INFORMATIKAI ALGORITMUSOK II.

Iványi Antal

alkotó szerkesztő

INFORMATIKAI

ALGORITMUSOK II.

ELTE Eötvös Kiadó, Budapest, 2005

A könyv az Oktatási Minisztérium támogatásával, a
Felsőoktatási Tankönyv- és Szakkönyvtámogatási
Pályázat keretében jelent meg.

Alkotó szerkesztő: Iványi Antal

A második kötet szerzői: Ivanyos Gábor és Rónyai Lajos (18. fejezet), Kása Zoltán (19.),
Csörnyei Zoltán (20.), Gács Péter (21.), Farkas Gábor és Kátai Imre (22.),
Burkhard Englert, Darius Kowalski, Grzegorz Malewicz és Alexander Shvartsman (23.),
Horváth Zoltán és Tejfel Máté (24.), Illés Tibor, Nagy Marianna és Terlaky Tamás (25.),
Lakatos László, Szeidl László és Telek Miklós (26.), Imreh Csanád (27.)
Bodon Ferenc (28.), Fogaras Dániel és Lukács András (29.),
Demetrovics János és Sali Attila (30.), Kiss Attila (31.)

A második kötet szakmai lektorai: Pethő Attila (18. fejezet), Fülöp Zoltán (19.),
Dömösi Pál (20.), Gál Anna (21.), Járai Antal (22.), Majzik István (23.),
Pataricza András (24.), Mayer János (25. és 26.), Vizvári Béla (27.) Rónyai Lajos (28.)
Benczúr A. András (29.), Kiss Attila (30.), Benczúr András (31.)

Nyelvi lektor: Biró Gabriella

Fordító: Lencse Zsolt (23. fejezet)

A könyv címoldalán – a © HUNGART engedélyével és az ELTE Informatikai Karának
támogatásával – Vasarely Victor *Kubtuz* című festménye látható. A borítóhoz felhasznált
filmet a © Goma RT. bocsátotta a rendelkezésünkre, a borítót Iványi Antal tervezte.

ISBN: 963 463 775 2

ELTE Eötvös Kiadó
1051 Budapest, Szerb utca 21–23., Telefon: 411-6740, Fax 485-52-26
Honlap: http://www.elte.hu/szervezet/eotvos_kiado.html
Villámcím: eotvoskiado@ludens.elte.hu
Felelős kiadó: Pándi András

Nyomás és kötés: Debreceni Kinizsi Nyomda
Felelős vezető: Bördős János

Tartalomjegyzék

Előszó

Nagy örömmel ajánlom az Olvasók figyelmébe az *Informatikai Algoritmusokat,* Iványi Antal gondos szerkesztésében. A számítógépes algoritmusok az informatika igen fontos és igen gyorsan fejlődő területét alkotják. Hatalmas hálózatok tervezése és üzemeltetése, nagyméretű tudományos számítások és szimulációk, gazdasági tervezés, adatvédelmi módszerek, titkosítás és még sok más alkalmazás igényel hatékony, gondosan tervezett és pontosan elemzett algoritmusokat.

Sok évvel ezelőtt Gács Péterrel írtunk egy kis könyvecskét *Algoritmusok* címmel. Az *Informatikai algoritmusok* két kötete mutatja, hogy milyen sokrétű és szerteágazó területté fejlődött ez a téma. Külön örömet jelent, hogy a magyar informatika ilyen sok kiváló képviselője fogott össze, hogy ez a könyv létrejöjjön. Nyilvánvaló számomra, hogy diákok, kutatók és alkalmazók egyik legfontosabb forrásmunkája lesz hosszú ideig.

Redmond, 2005. április 15.

Lovász László

Bevezetés

Az informatikai algoritmusok magyar nyelvű szakirodalma az utóbbi huszonöt évben alakult ki. Az első szakkönyvet Lovász László és Gács Péter írta 1978-ban [232]. Ezt a könyvet fordítások követték: 1982-ben Aho, Hopcroft és Ullman [9] könyve, 1987-ben Knuth háromkötetes monográfiája [205, 206, 207], majd 1987-ben Cormen, Leiserson és Rivest műve [70]. 1999-ben újra hazai szerzők következtek – Rónyai Lajos, Ivanyos Gábor és Szabó Réka [297] – majd 2002-ben megjelent Lynch *Osztott algoritmusok* című monográfiája [236].

Ezt 2003 tavaszán Iványi Antal *Párhuzamos algoritmusok* című könyve [174], majd 2003 őszén – *Új algoritmusok* címmel – Cormen, Leiserson, Rivest és Stein tankönyvének [71] fordítása követte.

A magyar informatikus hallgatók és gyakorlati szakemberek nagy érdeklődéssel fogadták az *Új algoritmusokat* – néhány hónap alatt a kiadott 2000 példány fele gazdára talált. Ez ösztönözte ennek a könyvnek a hazai szerzőit, hogy – külföldi kollégáik segítségével – további informatikai területek algoritmusait is összefoglalják.

2004 októberében jelent meg az *Informatikai algoritmusok 1* [175], majd 2005 májusában elkészült ez a második kötet is.

A könyv tartalmát hat részre tagoltuk: *Alapok, Hálózatok, Diszkrét optimalizálás, Folytonos optimalizálás, Adatbázisok* és *Alkalmazások*.

Az első kötetbe azok a fejezetek (17) kerültek, amelyek 2004 áprilisáig elkészültek. Ez a második kötet további 14 fejezetet tartalmaz.

Minden fejezet bemutat egy alkalmazási vagy elméleti szempontból lényeges területet és azokhoz kapcsolódó algoritmusokat. Az algoritmusok többségét szóban és olyan pszeudokóddal is megadjuk, amely a programozási tapasztalattal rendelkező olvasók számára könnyen érthető.

Az első kötet 247 ábrát, 157 pszeudokódot és 133 példát tartalmaz, amelyek elősegítik a tárgyalt algoritmusok működésének megértését. A második kötetben 175 ábra, 137 pszeudokód és 106 példa van. Az önálló tanulást az alfejezetek végén lévő gyakorlatok (az első kötetben összesen 269, a második kötetben 298), az egyes témákban való elmélyülést pedig a fejezetek végén lévő (az első kötetben összesen 66, a másodikban 41) feladatok segítik.

A fejezetek anyagával kapcsolatos friss és kiegészítő ismeretekre való utalások találhatók a fejezetek végén lévő *Megjegyzések a fejezethez* című részben. Az *Irodalomjegyzékben* megadjuk egyrészt a felhasznált szakirodalom bibliográfiai adatait, másrészt – teljességre törekedve – felsoroljuk a magyar nyelvű forrásokat. Az irodalomjegyzék számos eleme

felhasználható a megfelelő honlapra való ugráshoz. A könyvet *Névmutató* és *Tárgymutató* zárja.

Az algoritmusok bemutatása az igényelt erőforrások – elsősorban futási idő és memória – elemzését is magában foglalja. A szakirodalomban szokásos módon felső korlátokat adunk meg a legrosszabb esetre jellemző erőforrásigényre, és esetenként a megoldandó probléma erőforrásigényére jellemző alsó korlátot is levezetünk.

A könyv kéziratát LaTeX kiadványszerkesztő eszköz segítségével készítettük, amelyet az elmúlt hat év során Belényesi Viktorral és Locher Kornéllal fejlesztettünk ki, és korábban már öt könyv kéziratának előállítására használtunk. Az ábrák többségét Locher Kornél rajzolta. Az irodalomjegyzéket Iványi Anna tette élővé.

Garey és Johnson klasszikus művét [127] követve mindazon algoritmusok futási idejét exponenciálisnak nevezzük, amelyekre nem adható polinomiális felső korlát.

Az *Új algoritmusok* példáját követve tizedespontot használunk.

Mindig különös gondot fordítunk könyveink külsejére. Az adott esetben olyan megoldást kerestünk, amely

- tükrözi a könyv tartalmi gazdagságát (az első kötet 17 és a második kötet 14 fejezetét)
- és az alkotók szoros kötődését mind Magyarországhoz, mind pedig Európához.

Úgy gondoljuk, hogy a pécsi születésű Vásárhelyi Viktor – aki francia festőként Victor Vasarely néven vált világhírűvé – képeire jellemző a formák és színek gazdagsága, életútja pedig tükrözi kultúránk európai kötődését.

A budapesti és pécsi múzeumokban összesen közel 500 Vasarely-alkotás van. Ezek a művész ajándékai – a szülőföld iránti hála és tisztelet szimbólumai. Vasarely gazdag életművéből a könyv alkotói és majdani olvasói segítségével választottuk ki a *Dirac* és a *Kubtuz* című festményeket, amelyeken szakaszokból kör alakul ki – szemléltetve az informatika azon alapvető tulajdonságát, hogy a folytonos valós világot diszkrét objektumokkal (bitekkel) írja le.

Közismert, hogy az elmúlt évszázadban nemcsak művészeink, hanem sok kiváló tudósunk is külföldön ért fel a csúcsra. Nagy részükre azonban folyamatosan számíthat a hazai oktatás és tudományos élet. A hálózati szimulációs fejezet szerzője Gyires Tibor (Illinois Egyetem), a játékelméleti fejezetet pedig Szidarovszky Ferenc (Arizonai Műszaki Egyetem) írta. A második kötetben a megbízhatóságról szóló fejezetet Gács Péter (Bostoni Egyetem) írta, a belsőpontos módszerekről szóló fejezet egyik szerzője pedig Terlaky Tamás (McMaster Egyetem). Ma mind a négy szerző az adott terület vezető kutatója, amerikai egyetemek professzora – egykor magyar egyetemen tanultak, majd tanítottak.

A rekurziót (első kötet), valamint az automatákat és formális nyelveket (második kötet) tárgyaló fejezet szerzője Kása Zoltán (Babeş-Bolyai Tudományegyetem), a szisztolikus rendszerekről szóló fejezetet Szakács Laura (Babeş-Bolyai Tudományegyetem) fordította németről magyarra. Részvételük a könyv megszületésében a határainkon túli magyar nyelvű oktatással való szoros kapcsolatunk része.

Könyvünk tartalmi gazdagsága jó külföldi – elsősorban német – kapcsolatainknak is köszönhető. Az első kötet kriptográfiai és bonyolultságelméleti fejezetét Jörg Rothe (Düsseldorfi Egyetem), szisztolikus rendszerekkel foglalkozó fejezetét Eberhard Zehendner (Friedrich Schiller Egyetem) írta. Az adattömörítési fejezet szerzője Ulrich Tamm (Chemnitzi Egyetem), a párhuzamos programozásról szóló fejezet egyik szerzője Claudia Leopold (Kasseli Egyetem), az ember-gép kapcsolatokkal foglalkozó fejezet szerzői Ingo Althöfer

és Stefan Schwarz (Friedrich Schiller Egyetem). A második kötet osztott algoritmusokat ismertető fejezetét lengyel és német kollégák – Burkhard Englert, Darius Kowalski, Grzegorz Malewicz és Alexander Shvartsman – írták.

Az alkotók (szerzők, lektorok, fordítók és segítőtársaik) többsége a hazai informatikai felsőoktatás meghatározó intézményeinek – Budapesti Corvinus Egyetem, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapesti Műszaki Főiskola, Debreceni Egyetem, Eötvös Loránd Tudományegyetem, Miskolci Egyetem, Pécsi Tudományegyetem, Szegedi Tudományegyetem – oktatója.

Az *Oktatási Minisztérium* támogatásának köszönhetően ezen tankönyv első kötete nagyon kedvező áron kapható a kiadóban. Ugyancsak az Oktatási Minisztérium támogatásának köszönhető, hogy 2005 márciusától az első kötet elektronikus változata is mindenki számára szabadon hozzáférhető az ELTE Informatikai Karának a

http://elek.inf.elte.hu/

címen lévő könyvtárában. Ugyanitt találják meg Olvasóink a nyomtatott első kötet kiegészítését, amely többek között a szerzők és lektorok által javasolt kiegészítéseket, valamint az eddig megtalált hibák jegyzékét tartalmazza.

Ugyancsak az OM támogatásának köszönhetően 2005 tavaszán elkészültek az első digitális informatikai tankönyvek: Fóthi Ákos és Horváth Zoltán *Bevezetés a programozásba* [123], Lovász László *Kombinatorikai problémák és feladatok* [233], valamint Stoyan Gisbert és Takó Galina *Numerikus módszerek 2* [324] című műve. Ugyanekkor vált hozzáférhetővé a *Párhuzamos algoritmusok* [174] című elektronikus tankönyv is.

Egyelőre csak az ELTE hallgatói férnek hozzá az „Encyclopedia of Information Science and Technology” angol nyelvű nyomtatott és elektronikus változatához [197].

Az alábbi kollégáknak köszönjük, hogy a tervezett könyv mindkét formáját támogatták: Fazekas Gábor egyetemi docens (Debreceni Egyetem Informatikai Koordinációs Kutató Központjának igazgatója), Imreh Balázs tanszékvezető egyetemi docens (Szegedi Egyetem), Kása Zoltán egyetemi tanár (BBTE Matematikai és Informatikai Karának dékánhelyettese), Kozma László egyetemi docens (ELTE Informatikai Karának dékánja), Jörg Rothe egyetemi tanár (Heinrich Heine Universität, Düsseldorf), Sima Dezső egyetemi tanár (Budapesti Műszaki Főiskola Neumann János Informatikai Karának főigazgatója), Sidló Csaba PhD hallgató (ELTE Informatikai Doktori Iskola), Szeidl László egyetemi tanár (Pécsi Tudományegyetem Matematikai és Informatikai Intézet igazgatója), Szidarovszky Ferenc egyetemi tanár (Arizonai Műszaki Egyetem), Szirmay-Kalos László egyetemi tanár (BME Villamosmérnöki és Informatikai Kara), Terlaky Tamás egyetemi tanár (McMaster Egyetem, Hamilton).

Ugyancsak köszönjük azoknak a kollégáinknak a segítőkészségét, akiknek a lektori véleményét csatolni tudtuk a pályázathoz: Fekete István egyetemi docens (*Rekurziók* című fejezet), Fridli Sándor egyetemi docens (*Adattömörítés*), Gonda János egyetemi docens (*Kriptográfia*), Hunyadvári László egyetemi docens és Katsányi István PhD hallgató (*Bioinformatika*), Kiss Attila egyetemi docens (*Relációs adatbázisok tervezése*), Tőke Pál egyetemi docens (*Hálózatok szimulációja*), Vida János egyetemi docens (*Grafika*).

Köszönet illeti azokat – Bánsághi Anna programtervező matematikus hallgató (ELTE IK), Belényesi Viktor programtervező matematikus hallgató (ELTE IK), Benyó Tamás programtervező matematikus hallgató (ELTE IK), Biró Gabriella (programtervező matematikus), Csörnyei Zoltán egyetemi docens, (ELTE IK), Gyires Tibor egyetemi tanár (Illinois Egyetem), Imrényi Katalin tanszéki előadó (ELTE IK), Iványi Anna program koordinátor (KVVM), Iványi Antal (villamosmérnök), Kása Zoltán egyetemi tanár (BBTE), Kiss Attila egyetemi docens (ELTE IK), Kurucz Miklós programtervező matematikus hallgató (ELTE IK), Locher Kornél programtervező matematikus hallgató (ELTE IK), Rét Anna szerkesztő (Műszaki Könyvkiadó), Rónyai Lajos egyetemi tanár (BME VIK), Sali Attila tudományos főmunkatárs (MTA Rényi Intézet), Sima Dezső egyetemi tanár (BMF Neumann János Informatikai Kara), Szabados Kristóf programtervező matematikus hallgató (ELTE IK), Szendrei Rudolf programtervező matematikus hallgató (ELTE IK), Szidarovszky Ferenc egyetemi tanár (Arizonai Műszaki Egyetem), Szirmay-Kalos László egyetemi tanár (BME VIK), Takács Dániel programtervező matematikus hallgató (ELTE IK) – akik észrevételeikkel segítettek az *Informatikai algoritmusok 1* hibalistájának összeállításában.

Ugyancsak köszönet illeti az ELTE Informatikai Karának azon hallgatóit, akik egy-egy fejezet véleményezésével, hibáinak javításával segítettek az elektronikus első kötet létrehozásában – a fejezetek sorrendjében Burcsi Péter (2. fejezet), Szabados Kristóf (6. fejezet), Szendrei Rudolf (10. fejezet), Benyó Tamás (11. fejezet) és Bánsághi Anna (13. fejezet) – valamint a második kötet kéziratának javításában: Hevér Andrea (20. és 31. fejezet), Germán László (22. fejezet), Szabó Gábor (23. fejezet), Sleinitz István (24. fejezet).

A későbbiekben szeretnénk mind az első nyomtatott kiadás, mind az elektronikus kiadás hibáit kijavítani. Ezért kérjük a könyv Olvasóit, hogy javaslataikat, észrevételeiket küldjék el a

tony@inf.elte.hu

címre – levelükben lehetőleg pontosan megjelölve a hiba előfordulási helyét, és megadva a javasolt szöveget.

Olvasóink javaslataikkal, kérdéseikkel megkereshetik a könyv alkotóit is (címük megtalálható a kolofonoldalon).

Budapest, 2005. május 29.

Iványi Antal

alkotó szerkesztő

VII. ALAPOK

Bevezetés

Az első kötet négy alapozó témakörét (rekurzív egyenletek, komputeralgebra, kriptográfia, bonyolultságelmélet) most további öt követi.

A tizennyolcadik fejezet – magyar nyelvű tankönyvben először – a legfontosabb *algebrai* algoritmusokat tekinti át.

A következő fejezet – több friss idegen nyelvű tankönyv példáját követve – egységes szerkezetben tárgyalja az *automaták és formális nyelvek* elméletét.

A huszadik fejezet a mai számítógépek felhasználáshoz nélkülözhetetlen *fordítóprogramok* szerkezetét, valamint a lexikális és szintaktikus elemzés gyakorlatban használt módszereit foglalja össze.

Ezután a *megbízható számítások* feltételeinek és költségeinek elemzése következik.

Ezt a részt a *számelmélet* néhány – az informatikai felhasználhatóság szempontjai szerint kiválasztott – aktuális részterületének vizsgálata követi.

18. Algebra

A fejezetben először néhány algebrai alapfogalmat ismertetünk (18.1. alfejezet). Elsősorban a konstruktív alkalmazások szempontjából központi jelentőségű egyváltozós polinomgyűrű tulajdonságaival foglalkozunk. Ezután a véges testek elméletének alapjait körvonalazzuk, különös figyelmet szentelve a véges testek konstrukciójának (18.2. alfejezet) és a felettük értelmezett polinomok felbontásának (18.3. alfejezet). Majd rácsokkal foglalkozunk, és ismertetjük a Lenstra–Lenstra–Lovász-algoritmust, amivel rövid vektorokat kereshetünk rácsokban (18.4. alfejezet). Bemutatjuk a módszer első nevezetes alkalmazását jelentő algoritmust, ami polinom idejű módszert ad racionális együtthatós polinomok felbontására (18.5. alfejezet).

18.1. Testek, vektorterek, polinomok

Ebben az alfejezetben a gyűrűkkel és a polinomokkal kapcsolatos alapfogalmakat tekintjük át.

18.1.1. Gyűrűkkel kapcsolatos alapfogalmak

Emlékeztetünk néhány – az *Új algoritmusok* 31. és 32. fejezetében bevezetett – fogalomra. A továbbiakban a 31. és 32. fejezettel kapcsolatos hivatkozások mindegyike ugyanerre a tankönyvre vonatkozik.

A legalább két elemű S halmazt ***gyűrűnek*** nevezzük, ha értelmezve van rajta két kétváltozós művelet, az *összeadás*, aminek jele $+$, és a *szorzás*, amit $\cdot$ jelöl. Az S elemei az összeadásra nézve kommutatív csoportot alkotnak. Az S a szorzás műveletével egységelemes félcsoportot alkot, amelynek egységelemét 1 jelöli. Feltesszük, hogy $0 \neq 1$. Teljesülnek továbbá a disztributív szabályok: tetszőleges $a, b, c \in S$ elemekre

$$a \cdot (b + c) = a \cdot b + a \cdot c \text{ és}$$

$$(b + c) \cdot a = b \cdot a + c \cdot a \ .$$

Az összeadásra vonatkozó feltétel azt jelenti, hogy a művelet asszociatív, kommutatív, van egységeleme (amit 0 jelöl), és erre az egységelemre nézve minden elemnek van inverze. Pontosabban fogalmazva ezek a követelmények a következők:
asszociatív tulajdonság: minden $a, b, c \in S$ elemhármasra érvényes $(a + b) + c = a + (b + c)$,

kommutatív tulajdonság: minden $a, b \in S$ elempárra $a + b = b + a$,
egységelem létezése: az S halmaz 0 elemére és bármelyik a elemére $a + 0 = 0 + a = a$,
inverz létezése: minden $a \in S$ elemhez létezik olyan $b \in S$, amelyre $a + b = 0$ teljesül.
Belátható, hogy minden $a \in S$ elemnek egyetlen inverze van az összeadásra vonatkozóan. Az inverz szokásos jelölése $-a$.

A szorzással szemben az asszociatív tulajdonságot és az egységelem létezését követeljük meg. A gyűrű ***egységelemének*** a szorzásra vonatkozó, azaz multiplikatív egységelemet nevezzük. Az additív egységelem szokásos elnevezése ***nullelem,*** továbbá az összeadásra vonatkozó inverz pontosabb elnevezése ***additív inverz.*** A szorzást legtöbbször a $\cdot$ mellőzésével, a tényezők egymás mellé fűzésével írjuk le, pl. $a \cdot b$ helyett ab-t írunk.

18.1. példa. *Gyűrűk.*
1. $\mathbb{Z}$ az egészek halmaza a szokásos + és $\cdot$ műveletekkel.
2. A modulo m maradékosztályok $\mathbb{Z}_m$ halmaza a maradékosztályok között értelmezett összeadással és szorzással,
3. $\mathbb{R}^{n\times n}$ azaz n-szer n-es valós mátrixok a mátrixösszeadás és a mátrixszorzás műveleteivel.

Legyenek S_1 és S_2 gyűrűk. A $\phi : S_1 \to S_2$ leképezés ***homomorfizmus,*** ha művelettartó abban az értelemben, hogy $\phi(a \pm b) = \phi(a) \pm \phi(b)$ és $\phi(ab) = \phi(a)\phi(b)$ teljesül minden $a, b \in S_1$ elempárra. A ϕ homomorfizmust ***izomorfizmusnak*** nevezzük, ha ϕ kölcsönösen egyértelmű leképezés, és az inverze is homomorfizmus. Az S_1, S_2 gyűrűk ***izomorfak,*** ha van közöttük izomorfizmus. Az izomorfia szokásos jelölése: $S_1 \cong S_2$. Az izomorf gyűrűk algebrai szempontból azonosnak tekinthetők.

Homomorfizmus például az a $\phi : \mathbb{Z} \to \mathbb{Z}_6$ leképezés, amely egy egészhez a 6-tal való osztási maradékát rendeli: $\phi(13) = 1$, $\phi(5) = 5$, $\phi(22) = 4$ stb.

Hasznos és fontos gyűrűkonstrukció a ***direkt összeg:*** az S_1 és S_2 gyűrűk direkt összegét $S_1 \oplus S_2$ jelöli. Ennek alaphalmaza $S_1 \times S_2$, vagyis az (s_1, s_2) alakú párok halmaza, ahol $s_i \in S_i$. A műveleteket komponensenként végezzük. Legyen $s_i, t_i \in S_i$. Ekkor

$$(s_1, s_2) + (t_1, t_2) := (s_1 + t_1, s_2 + t_2) \text{ és}$$

$$(s_1, s_2) \cdot (t_1, t_2) := (s_1 \cdot t_1, s_2 \cdot t_2) \,.$$

Egyszerű számolással ellenőrizhető, hogy $S_1 \oplus S_2$ gyűrű lesz az itt definiált két művelettel. A konstrukció könnyen általánosítható $k \geq 2$ gyűrűre. Ekkor a gyűrűelemek k-komponensű vektorok lesznek és ezekkel a műveleteket komponensenként végezzük.

Testek

Az $\mathbb{F}$ gyűrű ***test,*** ha a nullától (vagyis az összeadás egységelemétől) különböző elemei a szorzásra nézve kommutatív csoportot alkotnak. Az $a \in \mathbb{F}\backslash\{0\}$ szorzásra vonatkozó (másként mondva: multiplikatív) inverzének szokásos jelölése $1/a$ vagy a^{-1}.

Jól ismert példák testekre a racionális számok, a valós számok és a komplex számok a szokásos műveletekkel. Jelölésük rendre $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$.

Fontos további példát jelentenek az $\mathbb{F}_p$ testek, ahol p egy prímszám. Az $\mathbb{F}_p$ elemei a modulo p maradékosztályok, a műveletek pedig a maradékosztályok körében értelmezett szorzás és összeadás. A disztributív szabály egyszerűen következik az egészekre érvényes disztributivitásból. A 33.12. tétel szerint $\mathbb{F}_p$ csoport az összeadásra nézve, a 33.13. tétel pedig azt mutatja, hogy $\mathbb{F}_p$ nem nulla elemeinek $\mathbb{F}_p^*$ halmaza csoport a szorzás műveletével.

Az utóbbi igazolásához szükség van arra a tényre, hogy p prímszám.

Karakterisztika, prímtest
Tetszőleges $\mathbb{F}$ testben nézhetjük az $m \cdot 1$ alakú, azaz az egységelem m példányából képzett $1 + \cdots + 1$ összegként előálló elemeket, ahol m pozitív egész. Két eset lehetséges:
(a) Az $m \cdot 1$ elemek egyike sem nulla,
(b) van olyan m, amelyre $m \cdot 1$ az $\mathbb{F}$ nulleleme.

Az (a) esetben $\mathbb{F}$-et ***nulla karakterisztikájú*** testnek nevezzük. A (b) esetben az $\mathbb{F}$ ***karakterisztikája*** a legkisebb m, amelyre $m \cdot 1 = 0$. Ez az m érték szükségképpen prímszám, ugyanis ha $m = rs$, akkor $0 = m \cdot 1 = rs \cdot 1 = (r \cdot 1)(s \cdot 1)$, tehát vagy $r \cdot 1 = 0$, vagy pedig $s \cdot 1 = 0$.

Jelölje P az $\mathbb{F}$-ben az 1-et tartalmazó legkisebb résztestet. A P az $\mathbb{F}$ ***prímteste.*** Az (a) esetben P az $(m \cdot 1)(s \cdot 1)^{-1}$ alakú elemek halmaza, ahol m tetszőleges, s pedig pozitív egész. Ekkor P izomorf $\mathbb{Q}$-val, a racionális számok testével. A megfeleltetés a kézenfekvő $(m \cdot 1)(s \cdot 1)^{-1} \leftrightarrow m/s$.

A (b) esetben a karakterisztika egy p prímszám, és P az $m \cdot 1$ elemek halmaza, ahol $0 \le m < p$. Ebben az esetben P izomorf a modulo p maradékosztályok $\mathbb{F}_p$ testével.

Vektorterek
Legyen $\mathbb{F}$ test. Az (additívan írt) V kommutatív csoportot $\mathbb{F}$ feletti ***vektortérnek,*** vagy $\mathbb{F}$-vektortérnek nevezzük, ha minden $a \in \mathbb{F}$ testelem és $v \in V$ esetében értelmezve van az $av \in V$ elem (tehát $\mathbb{F}$ hat V-n), és teljesülnek a következő azonosságok:

$$a(u + v) = au + av, \quad (a + b)u = au + bu \,,$$

$$a(bu) = (ab)u, \quad 1u = u \,.$$

Itt a és b az $\mathbb{F}$, u és v a V tetszőleges elemei, 1 pedig az $\mathbb{F}$ multiplikatív egységeleme.

Vektorterekre fontos példát szolgáltat az $\mathbb{F}$ test feletti m-szer n-es mátrixok tere. Ezek tulajdonságaival foglalkozik a 31. fejezet.

Az $\mathbb{F}$ feletti V vektortér ***véges dimenziós,*** ha létezik a V-nek véges sok $v_1, \ldots, v_n$ eleme úgy, hogy minden $v \in V$ megkapható $v = a_1 v_1 + \cdots + a_n v_n$ alakú ***lineáris kombinációként*** alkalmas $a_1, \ldots, a_n \in \mathbb{F}$ elemekkel. Az ilyen tulajdonságú $\{v_i\}$ vektorrendszert a V generátorrendszerének nevezzük. A legkisebb generátorrendszer elemszáma a V ***dimenziója*** $\mathbb{F}$ felett, ennek a jele $\dim_{\mathbb{F}} V$. A V véges dimenziós vektortér egy $\dim_{\mathbb{F}} V$ elemszámú generátorrendszerét ***bázisnak*** nevezzük.

A V vektortér elemeinek $\{v_1, \ldots, v_k\}$ halmaza ***lineárisan független***, ha a $0 \in V$ csak úgy kapható meg $0 = a_1 v_1 + \cdots + a_k v_k$ alakú lineáris kombinációként, hogy $a_1 = \cdots = a_k = 0$. Megmutatható, hogy a V minden bázisa lineárisan független vektorhalmaz is egyben. Fontos kapcsolódó tulajdonság, hogy a V vektortér bármely lineárisan független részhalmaza kiegészíthető bázissá. A $\dim_{\mathbb{F}} V$ dimenzió tehát megegyezik a legnagyobb V-beli lineárisan független részhalmaz méretével.

A nem üres $U \subseteq V$ egy ***altere*** V-nek, ha (additív) részcsoportja V-nek, és $au \in U$ teljesül minden $a \in \mathbb{F}$ és $u \in U$ esetében. Nyilvánvaló, hogy egy altér vektortér is egyben.

Vektorterek között is értelmezhető a homomorfizmus fogalma, itt viszont a szokásos neve ***lineáris leképezés.*** Legyenek V_1 és V_2 vektorterek az $\mathbb{F}$ felett. A $\phi : V_1 \to V_2$ leképezés

lineáris leképezés, ha minden $a, b \in \mathbb{F}$ és $u, v \in V_1$ esetén

$$\phi(au + bv) = a\phi(u) + b\phi(v) \ .$$

A ϕ lineáris leképezés (lineáris) ***izomorfizmus,*** ha ϕ kölcsönösen egyértelmű leképezés és az inverze is homomorfizmus. Két vektortér izomorf, ha van közöttük izomorfizmus.

18.1. lemma. *Legyen $\phi : V_1 \to V_2$ lineáris leképezés. Ekkor $U = \phi(V_1)$ altere V_2-nek. Ha ϕ injektív, akkor $\dim_{\mathbb{F}} U = \dim_{\mathbb{F}} V_1$. Ha itt még $\dim_{\mathbb{F}} V_1 = \dim_{\mathbb{F}} V_2 < \infty$ is teljesül, akkor $U = V_2$ és a ϕ izomorfizmus.*

Bizonyítás. A

$$\phi(u) \pm \phi(v) = \phi(u \pm v) \text{ és } a\phi(u) = \phi(au)$$

összefüggések mutatják, hogy U altér. Világos továbbá, hogy a V_1 generátorrendszerének a képe generátorrendszer lesz U-ban. Tegyük fel ezután, hogy ϕ injektív. Ekkor tetszőleges V_1-beli lineárisan független halmaz képe is lineárisan független. Az utóbbi két észrevétel alapján a V_1 bázisának képe bázis lesz U-ban, és ezért $\dim_{\mathbb{F}} U = \dim_{\mathbb{F}} V_1$. Ha mármost $\dim_{\mathbb{F}} V_2 = \dim_{\mathbb{F}} V_1$ is teljesül, akkor U egy bázisa a V_2-nek is bázisa, hiszen lineárisan független rendszer, és kiegészíthető V_2 bázisává. Tehát $U = V_2$ és a ϕ kölcsönösen egyértelmű leképezés. Könnyű látni – ezt az Olvasóra hagyjuk –, hogy ϕ^{-1} is lineáris leképezés. ■

A gyűrűkhöz hasonlóan vektorterek esetén is értelmes fogalom a ***direkt összeg.*** A V_1 és V_2 vektorterek direkt összegét $V_1 \oplus V_2$ jelöli. Az alaphalmaza $V_1 \times V_2$, az összeadást és $\mathbb{F}$ hatását komponensenként értelmezzük. Egyszerűen látható, hogy

$$\dim_{\mathbb{F}} (V_1 \oplus V_2) = \dim_{\mathbb{F}} V_1 + \dim_{\mathbb{F}} V_2 \ .$$

Test véges multiplikatív részcsoportja

Legyen $\mathbb{F}$ egy test és $G \subseteq \mathbb{F}$ véges multiplikatív részcsoport. A G tehát olyan nem nulla elemek halmaza, amely zárt a szorzásra és a multiplikatív inverz képzésére. A célunk annak az igazolása, hogy G ciklikus, azaz egy elemmel generálható. A ciklikus csoportokkal kapcsolatos alapfogalmak a 33.3.4. pontban találhatók meg. Emlékeztetünk, hogy az $a \in G$ elem rendje $\mathrm{ord}(a)$ a legkisebb pozitív egész k, amelyre $a^k = 1$.

Az a elem által generált ciklikus csoportot $\langle a \rangle$ jelöli. Teljesül, hogy $|\langle a \rangle| = \mathrm{ord}(a)$, és az a^i elem pontosan akkor generálja az $\langle a \rangle$ csoportot, ha i és n relatív prímek. A csoport generátorainak száma tehát $\phi(n)$ ahol ϕ az Euler-függvény (lásd a 33.3.2. pontot).

Szükségünk lesz még a következő, tetszőleges pozitív egész n-re érvényes azonosságra:

$$\sum_{d|n} \phi(d) = n \ .$$

Itt az összegezés az n pozitív osztóira értendő. Bizonyításként tekintsük az i/n törteket, ahol $1 \le i \le n$. Ezek száma éppen n. Egyszerűsítés után a törtek j/d alakúak lesznek, ahol d az n pozitív osztója. Egy rögzített d nevező nyilván éppen $\phi(d)$-szer fog előfordulni.

18.2. tétel. *Legyen $\mathbb{F}$ egy test és $G \subseteq \mathbb{F}$ véges multiplikatív részcsoportja. Ekkor van olyan $a \in G$, amelyre $G = \langle a \rangle$.*

Bizonyítás. Legyen $|G| = n$. Lagrange tétele (lásd 33.15. tétel) szerint minden $b \in G$ elem rendje osztója n-nek. Megmutatjuk, hogy tetszőleges d-re legfeljebb $\phi(d)$ eleme van $\mathbb{F}$-nek, amelynek a rendje d. A d-edrendű elemek gyökei az $x^d - 1$ polinomnak. Ha van $\mathbb{F}$-ben egyáltalán d-edrendű b elem, akkor a (kicsit később igazolandó) 18.5. lemma alapján $x^d - 1 = (x-b)(x-b^2)\cdots(x-b^d)$. Az $\mathbb{F}$ test d-edrendű elemei tehát mind benne vannak a $\langle b\rangle$ csoportban, amiben pedig éppen $\phi(d)$ darab d-edrendű elem van.

Ha a G-ben nem volna n-edrendű elem, akkor G minden elemének rendje valódi osztója volna n-nek. Ekkor az azonosság és $\phi(n) > 0$ figyelembevételével

$$n = |G| \le \sum_{d|n, d<n} \phi(d) < n\ ,$$

ami ellentmondás. ■

18.1.2. Polinomok

Legyen $\mathbb{F}$ test és $a_0, \dots, a_n \in \mathbb{F}$. Emlékeztetünk arra (lásd 32. fejezet), hogy az

$$f = f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

alakú kifejezés, ahol az x egy határozatlan, egy $\mathbb{F}$ feletti ***polinom.*** Az a_i elemek az f polinom ***együtthatói.*** A 0 polinom foka 0, míg a 0-tól különböző f ***foka*** a legnagyobb olyan j index, amelyre $a_j \neq 0$. Az f fokát $\deg f$ jelöli.

Az $\mathbb{F}$ feletti x határozatlanú polinomok halmazát $\mathbb{F}[x]$ jelöli. A legfeljebb n-edfokú $\mathbb{F}$ feletti

$$f = f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

és

$$g = g(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^n$$

polinomok összege a

$$h = h(x) = f + g = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$$

polinom, amelynek együtthatói $c_i = a_i + b_i$.

A fenti f és g polinomok fg szorzata pedig az a legfeljebb $2n$-edfokú

$$fg = d_0 + d_1 x + d_2 x^2 + \cdots + d_{2n} x^{2n}$$

polinom, amelynek együtthatóit a $d_j = \sum_{k=0}^{j} a_k b_{j-k}$ összefüggések adják meg. Itt a jobb oldalon az esetleges n-nél nagyobb indexű együtthatók nullának értendők. Egyszerű számolással megmutatható, hogy $\mathbb{F}[x]$ kommutatív gyűrűt alkot ezekkel a műveletekkel. Azonnal látható, hogy $\mathbb{F}[x]$ ***nullosztómentes*** gyűrű: ha $fg = 0$, akkor az f és a g valamelyike a 0 polinom.

Maradékos osztás, oszthatóság

Az $\mathbb{F}[x]$ polinomgyűrű sok tekintetben hasonlít az egészek $\mathbb{Z}$ gyűrűjéhez. Ilyen közös vonás, hogy polinomok körében is elvégezhető a maradékos osztás.

18.3. lemma. *Legyenek $f(x), g(x) \in \mathbb{F}[x]$ polinomok, $g(x) \neq 0$. Ekkor vannak olyan $q(x)$ és $r(x)$ polinomok, amelyekkel*

$$f(x) = q(x)g(x) + r(x)\,,$$

és itt vagy $r(x) = 0$, vagy pedig $\deg r(x) < \deg g(x)$. A feltételeknek eleget tevő q és r egyértelmű.

Bizonyítás. A lemma első felét az f foka szerinti indukcióval igazolhatjuk. Ha $f = 0$, vagy $\deg f < \deg g$, akkor az állítás nyilvánvaló. Tegyük fel tehát, hogy $\deg f \geq \deg g$. Ekkor a g alkalmas $q^*(x)g(x)$ többszörösét levonva f-ből elérhetjük, hogy az $f_1(x) = f(x) - q^*(x)g(x)$ foka kisebb $f(x)$ fokánál. Ekkor az indukciós feltevés szerint létezik egy

$$f_1(x) = q_1(x)g(x) + r_1(x)$$

felbontás, ahol vagy $r_1 = 0$, vagy $\deg r_1 < \deg g$. Látható, hogy ekkor $q(x) = q_1(x) + q^*(x)$ és $r(x) = r_1(x)$ teljesíti a követelményeket.

Egyértelműség: legyen $f(x) = Q(x)g(x) + R(x)$ egy másik, a lemma szerinti előállítás. Ekkor $(q(x) - Q(x))g(x) = R(x) - r(x)$. Ha itt a bal oldali polinom nem 0, akkor a foka legalább $\deg g$, míg a jobb oldali polinom foka kisebb, mint $\deg g$. Ez nem lehetséges. ■

Legyen az R egységelemes, kommutatív, nullosztómentes gyűrű és $R^* := R \setminus \{0\}$. Az R ***euklideszi gyűrű,*** ha van olyan $\phi : R^* \to \mathbb{N}$ függvény, amelyre $\phi(ab) \geq \phi(a)\phi(b)$, $a, b \in R^*$, továbbá ha $a \in R$, $b \in R^*$, akkor vannak olyan $q, r \in R$ elemek, amelyekre $a = qb + r$ és ha $r \neq 0$, akkor $\phi(r) < \phi(b)$. Az előző állítás mutatja, hogy $\mathbb{F}[x]$ euklideszi gyűrű, a ϕ szerepét a fok játszhatja.

Az egészek esetéhez hasonlóan $\mathbb{F}[x]$-ben is felépíthető az oszthatóság elmélete. A $g(x)$ polinom ***osztója*** az $f(x)$ polinomnak (jelöléssel: $g \mid f$), ha van olyan $q(x) \in \mathbb{F}[x]$, hogy $f(x) = q(x)g(x)$. Az $\mathbb{F}$ nem nulla elemei nyilván minden polinomnak osztói, ezek a ***triviális osztók*** vagy ***egységek.*** A nullától különböző $f(x) \in \mathbb{F}[x]$ polinom ***irreducibilis*** vagy *felbonthatatlan,* ha minden $\mathbb{F}[x]$-beli $f(x) = q(x)g(x)$ felbontás esetén vagy a q, vagy pedig a g egység.

Az $f, g \in \mathbb{F}[x]$ polinomok ***asszociáltak,*** ha van olyan $u \in \mathbb{F}^*$ hogy $f(x) = ug(x)$.

A 18.3. lemmát alkalmazva ugyanúgy igazolhatjuk az egyértelmű felbonthatóságról szóló tételt, mint az egészek körében (lásd a 33.1. alfejezetet). Az egész számok abszolút értékének szerepét itt a polinomok foka veszi át.

18.4. tétel. *Tetszőleges $0 \neq f \in \mathbb{F}[x]$ polinom felírható*

$$f(x) = uq_1(x)^{e_1} \cdots q_r(x)^{e_r}$$

alakban, ahol az $u \in \mathbb{F}^*$ egység, a $q_i \in \mathbb{F}[x]$ polinomok páronként nem asszociált irreducibilis polinomok, az e_i számok pedig pozitív egészek. A felbontás lényegileg egyértelmű: ha

$$f(x) = UQ_1(x)^{d_1} \cdots Q_s(x)^{d_s}$$

egy másik hasonló felbontás, akkor $r = s$, a Q_i tényezők esetleges cseréje után a q_i és a Q_i polinomok asszociáltak, és $d_i = e_i$ minden $1 \leq i \leq r$ esetén.

Két polinom ***relatív prím,*** ha nincs közös irreducibilis osztójuk.

Az $a \in \mathbb{F}$ elem ***gyöke*** az $f \in \mathbb{F}[x]$ polinomnak, ha $f(a) = 0$. Itt az $f(a)$ értéket úgy kapjuk, hogy $f(x)$-ben az x helyébe a-t írunk.

18.5. lemma. *Tegyük fel, hogy $a \in \mathbb{F}$ gyöke az $f(x) \in \mathbb{F}[x]$ polinomnak. Ekkor van olyan $g(x) \in \mathbb{F}[x]$, amellyel $f(x) = (x-a)g(x)$. Az f gyökeinek száma ezért legfeljebb* $\deg f$.

Bizonyítás. A 18.3. lemma szerint létezik $g(x) \in \mathbb{F}[x]$ és r, melyekre $f(x) = (x-a)g(x) + r$, és itt $r \in \mathbb{F}$. Az x helyébe a-t írva azt kapjuk, hogy $r = 0$. A második állítás innen fokszám szerinti indukcióval következik abból, hogy az f gyökei az a elem és a $g(x)$ gyökei. ■

A polinomműveletek költsége

Legyenek $f(x), g(x) \in \mathbb{F}[x]$ legfeljebb n-edfokú polinomok. Ekkor az $f(x) \pm g(x)$ polinom nyilvánvalóan kiszámítható $O(n)$ darab $\mathbb{F}$-beli aritmetikai művelettel. Az $f(x)g(x)$ szorzatot a definíciót közvetlenül követve $O(n^2)$ testművelettel kaphatjuk meg. Ha az $\mathbb{F}$ test felett elvégezhető a gyors Fourier-transzformáció, akkor a szorzás megvalósítható $O(n \lg n)$ testművelettel is (lásd 32.2. tétel). Általános testekre az aszimptotikusan leggyorsabb ismert polinomszorzó algoritmusok (pl. a Schönhage–Strassen-módszer) költsége $O(n \lg n \lg\lg n)$, tehát $\widetilde{O}(n)$ testművelet.

A maradékos osztás, vagyis azon $q(x)$ és $r(x)$ polinomok meghatározása, amelyekkel $f(x) = q(x)g(x) + r(x)$ és $r(x) = 0$, vagy $\deg r(x) < \deg g(x)$, a 18.3. lemma bizonyításában körvonalazott, kézenfekvő módon $O(n^2)$ testművelettel elvégezhető. Erre a feladatra is létezik azonban $\widetilde{O}(n)$ lépésszámú megoldás (Sieveking–Kung-algoritmus), ezt nem részletezzük.

Kongruencia, maradékosztálygyűrű

Legyen $f(x) \in \mathbb{F}[x]$, $\deg f = n > 0$, és $g, h \in \mathbb{F}[x]$. Azt mondjuk, hogy a g ***kongruens*** h-val modulo f, jelöléssel $g \equiv h \pmod f$, ha f osztója a $g - h$ polinomnak. Az így értelmezett kongruencia sok tekintetben hasonlít az egészek körében bevezetett kongruencia fogalmára (lásd a 33.3.2. pontot). Így a definíció alapján közvetlenül belátható, hogy a $\equiv$ reláció ekvivalenciareláció az $\mathbb{F}[x]$ halmazon. Jelölje $[g]_f$ (vagy egyszerűen csak $[g]$, ha f világos a szövegkörnyezetből) a g polinomot tartalmazó ekvivalenciaosztályt. A 18.3. lemmából azonnal adódik, hogy minden g -hez pontosan egy olyan $r \in \mathbb{F}[x]$ létezik, amelyre $[g] = [r]$ és vagy $r = 0$ (ha f osztja g-t), vagy pedig $\deg r < n$. Ezt az r polinomot a $[g]$ osztály ***reprezentánsának*** nevezzük. Az ekvivalenciaosztályok halmazát hagyományosan $\mathbb{F}[x]/(f)$ jelöli.

18.6. lemma. *Legyenek $f, f_1, f_2, g_1, g_2 \in \mathbb{F}[x]$ és $a \in \mathbb{F}$. Tegyük fel, hogy $f_1 \equiv f_2 \pmod f$ és $g_1 \equiv g_2 \pmod f$. Ekkor*

$$f_1 + g_1 \equiv f_2 + g_2 \pmod f ,$$

$$f_1 g_1 \equiv f_2 g_2 \pmod f ,$$

és

$$af_1 \equiv af_2 \pmod f .$$

Bizonyítás. Az első kongruencia

$$(f_1 + g_1) - (f_2 + g_2) = (f_1 - f_2) + (g_1 - g_2)$$

alapján világos, hiszen a jobb oldal nyilvánvalóan osztható f-fel. A második kongruencia hasonlóan adódik az

$$f_1g_1 - f_2g_2 = (f_1 - f_2)g_1 + (g_1 - g_2)f_2$$

azonosságból. Az utolsó kongruencia az $af_1 - af_2 = a(f_1 - f_2)$ folyománya. ■

A lemma módot kínál arra, hogy értelmezzük a kongruenciaosztályok összegét és szorzatát. A $[g]_f$ és $[h]_f$ osztályok összege legyen a $[g]_f + [h]_f := [g + h]_f$, a szorzata pedig a $[g]_f[h]_f := [gh]_f$. Az összeg/szorzat a lemma szerint független attól, hogy a kongruenciaosztályokat melyik elemükkel adtuk meg. Ugyanígy értelmezhetjük az $\mathbb{F}$ hatását az osztályokon: legyen $a[g]_f := [ag]_f$.

18.7. tétel. *Legyen $f(x) \in \mathbb{F}[x]$, $\deg f = n > 0$.*
1. Az $\mathbb{F}[x]/(f)$ az osztályokon értelmezett + és · műveletekkel egységelemes kommutatív gyűrűt alkot.
2. Az $\mathbb{F}[x]/(f)$ részgyűrűként tartalmazza az $\mathbb{F}$ testet és n-dimenziós vektortér az $\mathbb{F}$ felett, aminek az $[1], [x], \dots, [x^{n-1}]$ osztályok bázisát alkotják.
3. Ha f irreducibilis $\mathbb{F}[x]$-ben, akkor $\mathbb{F}[x]/(f)$ test.

Bizonyítás. 1. A gyűrűtulajdonságok egyszerűen következnek a polinomok körében érvényes hasonló tulajdonságokból. Példaként nézzük a disztributivitást:

$$[g]([h_1]+[h_2]) = [g][h_1+h_2] = [g(h_1+h_2)] = [gh_1+gh_2] = [gh_1]+[gh_2] = [g][h_1]+[g][h_2] \,.$$

Az összeadás egységeleme a $[0]$, a $[g]$ additív inverze a $[-g]$, a szorzás egységeleme pedig $[1]$ lesz. A részleteket az Olvasóra hagyjuk.

2. Az $[a]$ alakú osztályok halmaza ($a \in \mathbb{F}$) az $\mathbb{F}$-fel izomorf részgyűrűt alkot. A megfeleltetés a kézenfekvő $a \leftrightarrow [a]$. Az 1. alapján $\mathbb{F}[x]/(f)$ (additív) Abel-csoport, és az $\mathbb{F}$ hatására teljesülnek a vektortér-axiómák. Ez abból következik, hogy a polinomgyűrű maga vektortér az $\mathbb{F}$ felett. Példaként igazoljuk itt is az egyik disztributivitást:

$$a([h_1] + [h_2]) = a[h_1 + h_2] = [a(h_1 + h_2)] = [ah_1 + ah_2] = [ah_1] + [ah_2] = a[h_1] + a[h_2] \,.$$

Az $[1], [x], \dots, [x^{n-1}]$ osztályok lineárisan függetlenek. Ugyanis, ha

$$[0] = a_0[1] + a_1[x] + \cdots + a_{n-1}[x^{n-1}] = [a_0 + a_1x + \cdots a_{n-1}x^{n-1}] \,,$$

akkor $a_0 = \ldots = a_{n-1} = 0$, mert egy n-nél kisebb fokú polinom csak akkor lehet osztható f-fel, ha nulla. Másfelől tetszőleges g polinom esetén a $[g]$ reprezentánsának a foka kisebb n-nél, azaz a $[g]$ kifejezhető az $[1], [x], \dots, [x^{n-1}]$ osztályok lineáris kombinációjaként. Az $[1], [x], \dots, [x^{n-1}]$ osztályok tehát bázist alkotnak, és ennélfogva $\dim_{\mathbb{F}} \mathbb{F}[x]/(f) = n$.

3. Tegyük fel, hogy f irreducibilis. Először belátjuk, hogy $\mathbb{F}[x]/(f)$ nullosztómentes. Ha $[0] = [g][h] = [gh]$, akkor f osztja gh-t, tehát vagy osztja g-t, vagy pedig h-t, azaz $[g] = 0$ vagy $[h] = 0$ teljesül. Legyen most $g \in \mathbb{F}[x]$, $[g] \neq [0]$. A $[g][1], [g][x], \dots, [g][x^{n-1}]$ osztályok lineárisan függetlenek, ugyanis $[0] = a_0[g][1] + \cdots + a_{n-1}[g][x^{n-1}]$ esetén $[0] = [g][a_0 + \cdots + a_{n-1}x^{n-1}]$ és $a_0 = \ldots = a_{n-1} = 0$ következik. Ennélfogva a $[g][1], [g][x], \dots, [g][x^{n-1}]$ halmaz bázist alkot. Léteznek tehát $b_i \in \mathbb{F}$ együtthatók, amelyekkel

$$[1] = b_0[g][1] + \cdots + b_{n-1}[g][x^{n-1}] = [g][b_0 + \cdots + b_{n-1}x^{n-1}] \,.$$

Arra jutottunk, hogy bármely $[0] \neq [g]$-nek van multiplikatív inverze, ezért $\mathbb{F}[x]/(f)$ test. ■

A tétel 3. pontjában szereplő állítás megfordítását az Olvasóra bízzuk (18.1-1. gyakorlat).

18.2. példa. Az $\mathbb{F}[x]/(f)$ maradékosztálygyűrű elemeit hasznos a reprezentánsukkal megadni. Ezek, a 0 kivételével, éppen a deg f-nél kisebb fokú polinomok.

1. Legyen $\mathbb{F} = \mathbb{F}_2$, a kételemű test, és $f(x) = x^3 + x + 1$. Ekkor az $\mathbb{F}[x]/(f)$ nyolcelemű gyűrű, az elemei a

$$[0],\ [1],\ [x],\ [x+1],\ [x^2],\ [x^2+1],\ [x^2+x],\ [x^2+x+1]$$

osztályok.

Az összeadás tulajdonképpen a polinom-összeadás a reprezentánsok között. Például

$$[x^2+1] + [x^2+x] = [x+1]\ .$$

A szorzat számításakor a reprezentánsok szorzatát az f-fel való osztási maradékával helyettesítjük (más szóval redukáljuk). Így pl.

$$[x^2+1] \cdot [x^2+x] = [x^4+x^3+x^2+x] = [x+1]\ .$$

Az f irreducibilis $\mathbb{F}_2$ felett, mert harmadfokú, és nincs gyöke $\mathbb{F}_2$-ben. Ezért az $\mathbb{F}[x]/(f)$ maradékosztálygyűrű test.

2. Legyen $\mathbb{F} = \mathbb{R}$ és $f(x) = x^2 - 1$. A maradékosztálygyűrű elemei az $[ax+b]$ osztályok, ahol $a, b \in \mathbb{R}$. Az $\mathbb{F}[x]/(f)$ nem test, mert f nem irreducibilis. Például $[x+1][x-1] = [0]$.

18.8. lemma. *Legyen $\mathbb{L}$ egy az $\mathbb{F}$-et tartalmazó test és $\alpha \in \mathbb{L}$.*
1. Ha $\mathbb{L}$ véges dimenziós mint $\mathbb{F}$ feletti vektortér, akkor létezik olyan nem nulla $f \in \mathbb{F}[x]$ polinom, amelynek α gyöke.
2. Tegyük fel, hogy van olyan $0 \neq f \in \mathbb{F}[x]$, amelyre $f(\alpha) = 0$. Legyen g egy minimális fokú ilyen tulajdonságú polinom. Ekkor a g irreducibilis $\mathbb{F}[x]$-ben, és ha $h \in \mathbb{F}[x]$, $h(\alpha) = 0$, akkor g osztója h-nak.

Bizonyítás. 1. Kellően nagy n-re az $1, \alpha, \ldots, \alpha^n$ elemek lineárisan összefüggők $\mathbb{F}$ felett. Egy lineáris függés pedig éppen egy $0 \neq f \in \mathbb{F}[x]$ polinomot ad, amelyre $f(\alpha) = 0$.

2. Ha $g = g_1 g_2$, akkor $0 = g(\alpha) = g_1(\alpha) g_2(\alpha)$ miatt α gyöke a g_1-nek vagy g_2-nek. A g fokának minimalitása miatt ekkor g_1, g_2 egyike egység, tehát g irreducibilis. Végül pedig legyen $h \in \mathbb{F}[x]$, $h(\alpha) = 0$. Használjuk a 18.3. lemmát: alkalmas $q, r \in \mathbb{F}[x]$ polinomokkal $h(x) = q(x)g(x) + r(x)$. Ide az x helyébe α-t írva $r(\alpha) = 0$ adódik, ami csak úgy lehet, ha $r = 0$. ■

18.9. definíció. *A lemmabeli $g \in \mathbb{F}[x]$ polinomot α **minimálpolinomjának** nevezzük.*

A lemmából látszik, hogy az α minimálpolinomja $\mathbb{F}^*$-beli konstans szorzó erejéig egyértelmű. Gyakran hasznos feltenni, hogy a g főegyütthatója (a legmagasabb fokú tagjának együtthatója) 1.

18.10. következmény. *Legyen $\mathbb{L}$ az $\mathbb{F}$-et tartalmazó test és $\alpha \in \mathbb{L}$. Tegyük fel, hogy $f \in \mathbb{F}[x]$ irreducibilis és $f(\alpha) = 0$. Ekkor f minimálpolinomja α-nak.*

Bizonyítás. Legyen g az α minimálpolinomja. Az előző lemma szerint $g \mid f$ és g is irreducibilis. Ez csak úgy lehetséges, hogy f és g asszociáltak. ■

Legyen $\mathbb{L}$ az $\mathbb{F}$-et tartalmazó test és $\alpha \in \mathbb{L}$. Jelölje $\mathbb{F}(\alpha)$ a legszűkebb résztestet $\mathbb{L}$-ben, ami $\mathbb{F}$-et és α-t is tartalmazza.

18.11. tétel. *Legyen $\mathbb{L}$ az $\mathbb{F}$-et tartalmazó test és $\alpha \in \mathbb{L}$. Tegyük fel, hogy $f \in \mathbb{F}[x]$ az α minimálpolinomja. Ekkor az $\mathbb{F}(\alpha)$ test izomorf az $\mathbb{F}[x]/(f)$ testtel. Pontosabban létezik olyan $\phi : \mathbb{F}[x]/(f) \to \mathbb{F}(\alpha)$ izomorfizmus, amelynél $\phi(a) = a$ minden $a \in \mathbb{F}$ esetén, és $\phi([x]_f) = \alpha$. A ϕ egyben $\mathbb{F}$ feletti vektorterek izomorfizmusa is, tehát $\dim_{\mathbb{F}} \mathbb{F}(\alpha) = \deg f$.*

Bizonyítás. Tekintsük azt a $\psi : \mathbb{F}[x] \to \mathbb{L}$ leképezést, amelynél a $g \in \mathbb{F}[x]$ polinom képe $g(\alpha)$. Ez nyilvánvalóan gyűrűhomomorfizmus, és $\phi(\mathbb{F}[x]) \subseteq \mathbb{F}(\alpha)$. Most megmutatjuk, hogy $\psi(g) = \psi(h)$ pontosan akkor teljesül, ha $[g]_f = [h]_f$. Ugyanis $\psi(g) = \psi(h)$ akkor és csak akkor igaz, ha $\psi(g-h) = 0$, azaz ha $g(\alpha) - h(\alpha) = 0$, ami a 18.8. lemma szerint ekvivalens azzal, hogy $f \mid g - h$, vagyis $[g]_f = [h]_f$. Legyen ϕ a ψ által indukált $\mathbb{F}[x]/(f) \to \mathbb{F}(\alpha)$ leképezés: $\phi([g]_f) := \psi(g)$. Az előzőek szerint ϕ injektív. Rutin számolással adódik, hogy ϕ gyűrű- és vektortér-homomorfizmus is. $\mathbb{F}[x]/(f)$ test, tehát a homomorf képe $\phi(\mathbb{F}[x]/(f))$ is test. Ez tartalmazza $\mathbb{F}$-et és α-t is, így szükségképpen $\phi(\mathbb{F}[x]/(f)) = \mathbb{F}(\alpha)$. ■

Euklideszi algoritmus, legnagyobb közös osztó a polinomok körében

Legyenek $f(x), g(x) \in \mathbb{F}[x]$ polinomok, $g(x) \neq 0$. Legyen $f_0 = f$, $f_1 = g$ és képezzük az $\mathbb{F}[x]$-beli q_i és a további f_i polinomokat maradékos osztással a következők szerint:

$$f_0(x) = q_1(x) f_1(x) + f_2(x)\,,$$

$$f_1(x) = q_2(x) f_2(x) + f_3(x)\,,$$

$$\vdots$$

$$f_{k-2}(x) = q_{k-1}(x) f_{k-1}(x) + f_k(x)\,,$$

$$f_{k-1}(x) = q_k(x) f_k(x) + f_{k+1}(x)\,.$$

Itt $k > i > 1$ estén az f_{i+1} foka kisebb az f_i fokánál. A sorozatot addig képezzük, amíg nullát nem kapunk, vagyis $f_{k+1} = 0$. A 18.3. lemma szerint ez lehetséges. Legyen n az f és a g fokának a maximuma. A fokok csökkenése miatt $k \leq n+1$. Az itt vázolt számítássort szokás ***euklideszi algoritmusnak*** nevezni. Az egészek körében használható változata megtalálható a 33.2. alfejezetben.

Az $f(x), g(x) \in \mathbb{F}[x]$ polinomok legnagyobb közös osztója a $h(x) \in \mathbb{F}[x]$ polinom, ha $h(x) \mid f(x)$, $h(x) \mid g(x)$, és ha a $h_1(x) \in \mathbb{F}[x]$ osztója az f-nek és a g-nek is, akkor $h_1(x)$ osztója $h(x)$-nek is. Az $f(x)$ és a $g(x)$ legnagyobb közös osztójának a jele $\operatorname{lnko}(f(x), g(x))$. A 18.4. tételből világos, hogy $\operatorname{lnko}(f(x), g(x))$ létezik és $\mathbb{F}^*$-beli szorzó erejéig egyértelmű.

18.12. tétel. *Legyenek $f(x), g(x) \in \mathbb{F}[x]$ polinomok, $g(x) \neq 0$ és jelölje n az f és a g fokának a maximumát. Ekkor a fenti eljárás szerint definiált k számra és f_k polinomra*
1. $\operatorname{lnko}(f(x), g(x)) = f_k(x)$.

2. Vannak olyan legfeljebb n-edfokú $F(x), G(x)$ polinomok, amelyekkel

$$f_k(x) = F(x)f(x) + G(x)g(x)\ . \tag{18.1}$$

3. Adott f, g bemenettel az $F(x), G(x), f_k(x)$ polinomok $O(n^3)$ számú $\mathbb{F}$-beli alapművelettel kiszámíthatók.

Bizonyítás. 1. Az euklideszi sorozat mentén visszafelé lépdelve látható, hogy f_k osztója mindegyik f_i-nek, így f-nek és g-nek is. A sorozat elejétől a vége felé haladva látható, hogy ha $h(x)$ osztója f-nek és g-nek is, akkor, mindegyik f_i-nek osztója, így f_k-nak is. Ezzel beláttuk, hogy $\operatorname{lnko}(f(x), g(x)) = f_k(x)$.

2. Az állítás kézenfekvő, ha $f = 0$. Feltehetjük ezután, hogy $f \neq 0$. Az euklideszi sorozat elejétől indulva látható, hogy léteznek $F_i(x), G_i(x) \in \mathbb{F}[x]$ polinomok, amelyekkel

$$F_i(x)f(x) + G_i(x)g(x) = f_i(x) \tag{18.2}$$

igaz. Vegyük észre, hogy (18.2) akkor is fennáll, ha $F_i(x)$ helyébe az $F_i(x)$-nek a g-vel való $F_i^*(x)$ osztási maradékát írjuk és $G_i(x)$ helyébe a $G_i(x)$-nek az f-vel való $G_i^*(x)$ osztási maradékát írjuk. Ugyanis ekkor

$$F_i^*(x)f(x) + G_i^*(x)g(x) \equiv f_i(x) \pmod{f(x)g(x)},$$

és a kongruencia mindkét oldalán a polinomok foka kisebb, mint $\deg f \deg g$, amiből

$$F_i^*(x)f(x) + G_i^*(x)g(x) = f_i(x)$$

következik.

3. Ha már meghatároztuk az f_{i-1}, f_i, valamint az F_i^* és G_i^* polinomokat, akkor f_{i+1}, F_{i+1}^* és G_{i+1}^* megkapható $O(n^2)$ számú $\mathbb{F}$-beli aritmetikai művelettel. Kezdetben $F_1^* = 1$ és $G_2^* = -q_1$. Az állítás innen $k \leq n+1$ miatt már következik. ■

Megjegyzés. Hagyományosan az euklideszi algoritmus csak a legnagyobb közös osztót számolja ki. A (18.1) formulának eleget tevő $F(x)$ és $G(x)$ polinomokat is megadó változatot bővített euklideszi algoritmusnak nevezik. Az első kötet második fejezetében a polinomokra vonatkozó euklideszi algoritmust részletes tárgyalását találhatja az Olvasó. Viszonylag egyszerű megmutatni, hogy az $f_k(x)$ és alkalmas $F(x), G(x)$ polinomok valójában $O(n^2)$ művelettel is számíthatók. Az aszimptotikusan legjobb ismert módszerek költsége $\widetilde{O}(n)$.

Polinomok deriváltja

A polinomok többszörös tényezőinek vizsgálatakor gyakran hasznos a derivált polinom fogalma. Az

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \in \mathbb{F}[x]$$

polinom ***deriváltja*** az

$$f'(x) = a_1 + 2a_2x + \cdots + na_nx^{n-1}$$

polinom. A definícióból azonnal adódik, hogy az $f(x) \mapsto f'(x)$ egy $\mathbb{F}[x] \to \mathbb{F}[x]$ $\mathbb{F}$-lineáris leképezés: $(f(x) + g(x))' = f'(x) + g'(x)$ és $(af(x))' = af'(x)$ teljesül $(f(x), g(x) \in \mathbb{F}[x]$, $a \in \mathbb{F})$. Szorzat deriváltjára ad érdekes formulát a ***Leibniz-szabály*** tetszőleges $f(x), g(x) \in$

$\mathbb{F}[x]$ polinomok esetén $(f(x)g(x))' = f'(x)g(x)+f(x)g'(x)$. A szabály teljesülését a deriválás lineáris volta miatt elég az $f(x) = x^i$ és $g(x) = x^j$ alakú polinomokra ellenőrizni. Itt pedig nyilvánvalóan teljesül.

Az $f'(x)$ derivált polinom érzékeny az $f(x)$ többszörös tényezőire:

18.13. lemma. *Legyen $\mathbb{F}$ tetszőleges test és tegyük fel, hogy $f(x) \in \mathbb{F}[x]$ és $f(x) = u^k(x)v(x)$ alakú, ahol $u(x), v(x) \in \mathbb{F}[x]$. Ekkor $u^{k-1}(x)$ osztja $\mathbb{F}[x]$-ben az $f(x)$ polinom $f'(x)$ deriváltját.*

Bizonyítás. A k kitevő szerinti indukcióval a Leibniz-szabály segítségével látható, hogy $(u^k(x))' = ku^{k-1}(x)u'(x)$. Így, megint csak a Leibniz-szabály alkalmazásával $f'(x) = u^{k-1}(x)(ku'(x)v(x) + u^k(x)v'(x))$. Ezért $u^{k-1}(x) \mid f'(x)$. ■

Sok fontos esetben a megfordítás is igaz:

18.14. lemma. *Legyen $\mathbb{F}$ tetszőleges test és tegyük fel, hogy $f(x) \in \mathbb{F}[x]$ és $f(x) = u(x)v(x)$ alakú, ahol $u(x)$ és $v(x)$ relatív prímek, továbbá $u'(x) \neq 0$ (például F karakterisztikája 0 és $u(x)$ nem konstans). Ekkor az $f'(x)$ derivált polinom nem osztható $u(x)$-szel.*

Bizonyítás. A Leibniz-szabály szerint $f'(x) = u(x)v'(x)+u'(x)v(x) \equiv u'(x)v(x) \pmod{u(x)}$. Mivel $u'(x)$ foka kisebb $u(x)$ fokánál, $u'(x)$ nem osztható $u(x)$-szel, és így az $u'(x)v(x)$ szorzat sem, hiszen $u(x)$ és a $v(x)$ tényező relatív prímek. ■

A kínai maradéktétel polinomokra

A következő tétel szerint az $\mathbb{F}[x]/(f)$ gyűrű összerakható bizonyos $\mathbb{F}[x]/(g)$ alakú gyűrűkből, ahol $g \mid f$.

18.15. tétel. *(kínai maradéktétel polinomokra) Legyenek $f_1, \dots, f_k \in \mathbb{F}[x]$ pozitív fokú, páronként relatív prím polinomok, és $f = f_1 \cdots f_k$. Ekkor az $\mathbb{F}[x]/(f)$ és az $\mathbb{F}[x]/(f_1) \oplus \cdots \oplus \mathbb{F}[x]/(f_k)$ gyűrűk izomorfak. Az izomorfiát adó leképezés*

$$\phi : [g]_f \mapsto ([g]_{f_1}, \dots, [g]_{f_k}), \quad g \in \mathbb{F}[x] .$$

Bizonyítás. Először megjegyezzük, hogy ϕ definíciója korrekt. Ha $h \in [g]_f$, akkor $h = g + f^*f$, amiből látható, hogy a h és a g az f_i polinommal osztva ugyanazt a maradékot adják, vagyis $[h]_{f_i} = [g]_{f_i}$.

A ϕ nyilvánvalóan gyűrű homomorfizmus és lineáris leképezés a két $\mathbb{F}$ feletti vektortér között. A ϕ injektív: ha $\phi([g]) = \phi([h])$, akkor $\phi([g - h]) = (0, \dots, 0)$, vagyis $f_i \mid g - h$ $(1 \leq i \leq k)$, amiből $f \mid g - h$ és $[g] = [h]$ következik.

Az $\mathbb{F}[x]/(f)$ és az $\mathbb{F}[x]/(f_1)\oplus\cdots\oplus\mathbb{F}[x]/(f_k)$ vektorterek dimenziója megegyezik: mindkettő $\deg f$. A 18.1. lemma szerint ϕ vektortér izomorfizmus. Már csak annak a belátása maradt hátra, hogy ϕ^{-1} a szorzást is megtartja. Ennek az egyszerű ellenőrzését az Olvasóra bízzuk. ■

Gyakorlatok

18.1-1. Legyen $f \in \mathbb{F}[x]$ az $\mathbb{F}$ test feletti polinom. Igazoljuk, hogy ha az $\mathbb{F}[x]/(f)$ maradékosztálygyűrű nullosztómentes, akkor az f polinom irreducibilis.

18.1-2. Legyen R egységelemes, kommutatív gyűrű. Az $I \subseteq R$ halmaz ideál, ha I részcsoport az összeadásra nézve, és ha $a \in I$, $b \in R$, akkor $ab \in I$ is igaz. Mutassuk, meg, hogy R pontosan akkor test, ha az összes ideáljai $\{0\}$ és R.

18.1-3. Legyenek $a_1, \ldots, a_k \in R$. Jelölje $(a_1, \ldots, a_k)$ a legkisebb R-beli ideált, ami az a_i elemeket tartalmazza. Igazoljuk, hogy $(a_1, \ldots, a_k)$ mindig létezik és az elemei pontosan a $b_1a_1 + b_2a_2 + \cdots + b_ka_k$ alakú elemek, ahol $b_1, \ldots, b_k \in R$.

18.1-4. Az egységelemes, nullosztómentes, kommutatív R gyűrű ***főideálgyűrű,*** ha minden I ideáljához van olyan $a \in I$ elem, amellyel (az előző gyakorlat jelölését használva) $I = (a)$ teljesül. Mutassuk meg, hogy $\mathbb{Z}$ és $\mathbb{F}[x]$ ($\mathbb{F}$ test) is főideálgyűrű.

18.1-5. Legyen S egységelemes kommutatív gyűrű, I egy ideálja, és $a, b \in S$. Definíció szerint legyen $a \equiv b \pmod{I}$ akkor és csak akkor, ha $a - b \in I$. Igazoljuk a következőket:

a. A $\equiv$ reláció egy ekvivalenciareláció az S-en.

b. Jelölje $[a]_I$ az a elem ekvivalenciaosztályát, és S/I az ekvivalenciaosztályok halmazát. Legyen $[a]_I + [b]_I := [a + b]_I$, és $[a]_I[b]_I := [ab]_I$. Mutassuk meg, hogy ezekkel a műveletekkel S/I egységelemes kommutatív gyűrű. *Útmutatás.* Kövessük a 18.7. tétel bizonyítását.

18.1-6. Legyen $\mathbb{F}$ test, $f(x), g(x) \in \mathbb{F}[x]$, amelyekre $\operatorname{lnko}(f(x), g(x)) = 1$. Mutassuk meg, hogy ekkor létezik olyan $h(x) \in \mathbb{F}[x]$ polinom, hogy $h(x)g(x) \equiv 1 \pmod{f(x)}$. *Útmutatás.* Alkalmazzuk az euklideszi algoritmust.

18.2. Véges testek

A véges elemszámú testek – rövidebben véges testek – fontos szerepet játszanak a matematikában és annak több alkalmazási területén, így a számítások világában is. Sok fontos és hasznos konstrukció alapját jelentik. A következőkben ismertetjük a véges testek elméletének legfontosabb tételeit, figyelmet szentelve a konstrukciós kérdéseknek is.

Ebben részben p prímszámot jelöl, q pedig a p egy pozitív kitevőjű hatványa.

18.16. tétel. *Legyen $\mathbb{F}$ véges test. Ekkor létezik olyan p prímszám, hogy az $\mathbb{F}$ prímteste izomorf $\mathbb{F}_p$-vel (a modulo p maradékosztályok testével). Az $\mathbb{F}$ véges dimenziós vektortér $\mathbb{F}_p$ felett, és az elemszáma a p egy hatványa. Ha $\dim_{\mathbb{F}_p} \mathbb{F} = d$, akkor $|\mathbb{F}| = p^d$.*

Bizonyítás. $\mathbb{F}$ csak prím karakterisztikájú lehet, mert egy 0 karakterisztikájú testnek végtelen sok eleme van. A P prímtest tehát $\mathbb{F}_p$-vel izomorf, ahol p prím. Az $\mathbb{F}$ vektortér a P felett, mert P résztest. Legyen $\alpha_1, \ldots, \alpha_d$ bázisa $\mathbb{F}$-nek a P felett. A bázisban való felírás egyértelműsége miatt minden $\alpha \in \mathbb{F}$ pontosan egyféleképpen írható fel $\sum_{j=1}^{d} a_i\alpha_i$ alakban, ahol $a_i \in P$. Innen következik, hogy $|\mathbb{F}| = p^d$. ■

Az $\mathbb{F}$ test nem 0 elemeinek halmazát (multiplikatív csoportját) $\mathbb{F}^*$ jelöli. A 18.2. tételből azonnal adódik a következő

18.17. tétel. *Legyen $\mathbb{F}$ véges test. Az $\mathbb{F}^*$ multiplikatív csoport ciklikus.*

Az $\mathbb{F}^*$ csoport generátorelemét ***primitív elemnek*** nevezzük. Ha $|\mathbb{F}| = q$, és α primitív eleme $\mathbb{F}$-nek, akkor az $\mathbb{F}$ elemei $0, \alpha, \alpha^2, \ldots, \alpha^{q-1} = 1$.

18.18. következmény. *Legyen $\mathbb{F}$ véges test, $|\mathbb{F}| = p^d$ és α az $\mathbb{F}$ primitív eleme. Legyen $g \in \mathbb{F}_p[x]$ az α minimálpolinomja $\mathbb{F}_p$ felett. Ekkor g irreducibilis $\mathbb{F}_p[x]$-ben, a g foka d és $\mathbb{F}$ izomorf az $\mathbb{F}_p[x]/(g)$ testtel.*

Bizonyítás. Az α primitív eleme $\mathbb{F}$-nek, tehát $\mathbb{F} = \mathbb{F}_p(\alpha)$. Az állítások ezután közvetlenül adódnak a 18.8. lemmából és a 18.11. tételből. ■

18.19. tétel. *Legyen $\mathbb{F}$ véges test, $|\mathbb{F}| = q$. Ekkor*
*1. **(Kis Fermat-tétel.)** Minden $\beta \in \mathbb{F}^*$ elemre igaz, hogy $\beta^{q-1} = 1$.*
2. Ha $\beta \in \mathbb{F}$, akkor $\beta^q = \beta$.

Bizonyítás. 1. Legyen $\alpha \in \mathbb{F}^*$ primitív elem. Ekkor alkalmas i-re $\beta = \alpha^i$. Innen

$$\beta^{q-1} = (\alpha^i)^{q-1} = (\alpha^{q-1})^i = 1^i = 1 \ .$$

2. Az 1. alapján $\beta^q = \beta$ fennáll, ha $\beta \neq 0$. Nyilvánvalóan a $\beta = 0$ esetben is igaz az állítás. ■

18.20. tétel. *Legyen $\mathbb{F}$ egy q elemű test. Ekkor*

$$x^q - x = \prod_{\alpha \in \mathbb{F}} (x - \alpha) \ .$$

Bizonyítás. A 18.19. tétel és a 18.5. lemma alapján látjuk, hogy a jobb oldali szorzat osztója az $x^q - x \in \mathbb{F}[x]$ polinomnak. Innen a fokszámok és a főegyütthatók egyezése miatt adódik az állítás. ■

18.21. következmény. *Tetszőleges két azonos elemszámú véges test izomorf.*

Bizonyítás. Tegyük fel, hogy $q = p^d$, valamint $\mathbb{K}$ és $\mathbb{L}$ is q-elemű testek. Legyen β az $\mathbb{L}$ primitív eleme. Ekkor a 18.18. következmény szerint β-nak az $\mathbb{F}_p$ feletti minimálpolinomja egy d-edfokú $g(x) \in \mathbb{F}_p[x]$ ($\mathbb{F}_p[x]$-ben) irreducibilis polinom és $\mathbb{L} \cong \mathbb{F}_p[x]/(g(x))$. A g minimálpolinom a 18.8. lemma és a 18.19. tétel alapján osztója az $x^q - x$ polinomnak. A 18.20. tételt $\mathbb{K}$-ra alkalmazva azt kapjuk, hogy az $x^q - x$ polinom és így az ezt osztó $g(x)$ polinom lineáris tényezők szorzatára bomlik $\mathbb{K}[x]$-ben, következésképpen $g(x)$-nek van legalább egy α gyöke a $\mathbb{K}$ testben. Mivel $g(x)$ irreducibilis $\mathbb{F}_p[x]$-ben, α minimálpolinomja $g(x)$ (18.10. következmény), és így $\mathbb{F}_p(\alpha)$ is izomorf az $\mathbb{F}_p[x]/(g(x))$ testtel. Innen az elemszámokat összehasonlítva adódik, hogy $\mathbb{F}_p(\alpha) = \mathbb{K}$, továbbá hogy $\mathbb{K}$ és $\mathbb{L}$ is izomorfak. ■

Ezek után $\mathbb{F}_q$-val jelöljük a q elemű testet, amennyiben létezik. A létezés kérdésének tisztázásához hasznos lesz a következő két tény.

18.22. lemma. *Ha p egy prímszám és $0 < j < p$ egész, akkor $p \mid \binom{p}{j}$.*

Bizonyítás. A $\binom{p}{j}$ egyfelől egész. Másfelől $\binom{p}{j} = (p(p-1)\cdots(p-j+1))/j!$ egy olyan tört, aminek $0 < j < p$ esetén a számlálója osztható p-vel, a nevezője pedig nem. ■

18.23. lemma. *Legyen R egy kommutatív gyűrű, és tegyük fel, hogy $pr = 0$ minden $r \in R$ elemre. Ekkor a $\Phi_p : R \to R$, $\Phi_p : r \mapsto r^p$ leképezés egy gyűrű homomorfizmus.*

Bizonyítás. Legyenek $r, s \in R$. Nyilvánvaló hogy

$$\Phi_p(rs) = (rs)^p = r^p s^p = \Phi_p(r)\Phi_p(s) \ .$$

Az előző lemma alapján

$$\Phi_p(r+s) = (r+s)^p = \sum_{j=0}^{p} \binom{p}{j} r^{p-j} s^j = r^p + s^p = \Phi_p(r) + \Phi_p(s) \ .$$

Ugyanígy adódik, hogy $\Phi_p(r-s) = \Phi_p(r) - \Phi_p(s)$. ■

A lemmában szereplő Φ_p homomorfizmust az R gyűrű Frobenius-endomorfizmusának szokás nevezni.

18.24. tétel. *Tegyük fel, hogy a d pozitív egész számra az $\mathbb{F}_q[x]$-ben irreducibilis $g(x) \in \mathbb{F}_q[x]$ polinom osztója az $x^{q^d} - x$ polinomnak. Ekkor $g(x)$ foka osztója d-nek.*

Bizonyítás. Legyen $g(x)$ foka n és tegyük fel indirekt módon, hogy $d = tn + s$, ahol $0 < s < n$. A $g(x) \mid x^{q^d} - x$ feltevés másképpen fogalmazva azt jelenti, hogy $x^{q^d} \equiv x \pmod{g(x)}$. De ekkor tetszőleges $u(x) = \sum_{i=0}^{N} u_i x^i \in \mathbb{F}_q[x]$ polinomra

$$u(x)^{q^d} = \sum_{i=0}^{N} u_i^{q^d} x^{iq^d} = \sum_{i=0}^{N} u_i (x^{q^d})^i \equiv \sum_{i=0}^{N} u_i x^i = u(x) \pmod{g(x)}$$

is teljesül. Itt alkalmaztuk a 18.23. lemmát az $R = \mathbb{F}_q[x]/(g(x))$ gyűrűre, és a 18.19. tételt $\mathbb{F}$-re. Az $\mathbb{F}_q[x]/(g(x))$ maradékosztálygyűrű a q^n elemű $\mathbb{F}_{q^n}$ testtel izomorf. Legyen $u(x) \in \mathbb{F}_q(x)$ egy olyan polinom, amelyre $u(x) \pmod{g(x)}$ az $\mathbb{F}_{q^n}$ test egy primitív eleme: $u(x)^{q^n-1} \equiv 1 \pmod{g(x)}$, de $u(x)^j \not\equiv 1 \pmod{g(x)}$ $(j = 1, \dots, q^n - 2)$. Ugyanakkor

$$u(x) \equiv u(x)^{q^d} = u(x)^{q^{m+s}} = (u(x)^{q^{nt}})^{q^s} \equiv u(x)^{q^s} \pmod{g(x)} \ ,$$

tehát $u(x)(u(x)^{q^s-1} - 1) \equiv 0 \pmod{g(x)}$. Mivel az $\mathbb{F}_q[x]/(g(x))$ maradékosztálygyűrű test és $u(x) \not\equiv 0 \pmod{g(x)}$, ez csak úgy lehet, hogy $u(x)^{q^s-1} \equiv 1 \pmod{g(x)}$. Az $0 \le q^s - 1 < q^n - 1$ egyenlőtlenségek miatt ez ellentmond $u(x) \pmod{g(x)}$ primitív voltának. ■

18.25. tétel. *Tetszőleges p prímszámra és d pozitív egész számra létezik p^d-elemű test.*

Bizonyítás. Indukció d szerint. Az állítás $d = 1$-re nyilvánvaló. Az indukciós lépés: $d > 1$ esetén legyen r egy prímosztója d-nek. Az indukciós feltevés miatt létezik a $q = p^{(d/r)}$ elemű test. A 18.24. tétel miatt az $f(x) = x^{q^r} - x$ polinom tetszőleges $\mathbb{F}_q[x]$-ben irreducibilis faktora vagy első- vagy r-edfokú. Továbbá $f'(x) = (x^{q^r} - x)' = -1$, így $f(x)$ négyzetmentes a 18.13. lemma szerint. Az $\mathbb{F}_q$ felett elsőfokú faktorok száma legfeljebb q, ezek szorzata legfeljebb q-adfokú. Létezik tehát legalább $(q^r - q)/r \ge 1$ r-edfokú $\mathbb{F}_q[x]$-ben irreducibilis polinom. Legyen $g(x)$ egy ilyen polinom. Ekkor az $\mathbb{F}_q[x]/(g(x))$ a $q^r = p^d$ elemű testtel izomorf. ■

18.26. következmény. *Tetszőleges pozitív egész d esetében létezik $f \in F_p[x]$ irreducibilis d-edfokú polinom.*

Bizonyítás. Az $\mathbb{F}_{p^d}$ egy primitív elemének az $\mathbb{F}_p$ feletti minimálpolinomja éppen ilyen. ■

Kicsit később, a 18.31. tételben ennél erősebb állítást igazolunk: egy véletlen d-edfokú polinom jó eséllyel irreducibilis lesz.

Véges testek résztestei

A következő tétel pontos leírást ad egy véges test összes résztesteiről.

18.27. tétel. *Az $\mathbb{F} = \mathbb{F}_{p^n}$ test pontosan akkor tartalmaz $\mathbb{F}_{p^k}$-val izomorf résztestet, ha $k \mid n$. Ebben az esetben $\mathbb{F}$-nek pontosan egy $\mathbb{F}_{p^k}$-val izomorf részteste van.*

Bizonyítás. A $k \mid n$ feltétel szükséges, hiszen a nagyobb test vektortér a kisebb felett, így alkalmas l egésszel $p^n = (p^k)^l$ teljesül.

Fordítva, tegyük fel, hogy $k \mid n$, és legyen $f \in \mathbb{F}_p[x]$ irreducibilis, k-adfokú polinom. Ilyen a 18.26. következmény alapján létezik. Legyen $q = p^k$. A 18.19. tételt alkalmazva az $\mathbb{F}_p[x]/(f)$ testre kapjuk, hogy $x^q \equiv x \pmod f$, amiből $x^{p^n} = (x^{q^l}) \equiv x \pmod f$, tehát f osztója az $x^{p^n} - x$ polinomnak. A 18.20. tételt is figyelembe véve nyerjük, hogy f-nek van egy α gyöke $\mathbb{F}$-ben. A szokásos módon kapjuk innen, hogy az $\mathbb{F}_p(\alpha)$ résztest izomorf $\mathbb{F}_{p^k}$-val.

Az utolsó állítás azért igaz, mert az $\mathbb{F}_q$ elemei éppen az $x^q - x$ gyökei (18.20. tétel), ennek a polinomnak pedig bármely testben legfeljebb q gyöke lehet. ■

Az irreducibilis polinomok szerkezete

A következő állítás a véges testek feletti irreducibilis polinomok fontos tulajdonságait fogalmazza meg.

18.28. tétel. *Legyenek $\mathbb{F}_q \subseteq \mathbb{F}$ véges testek, $\alpha \in \mathbb{F}$. Legyen $f \in \mathbb{F}_q[x]$ az α elem 1 főegyüttható minimálpolinomja $\mathbb{F}_q$ felett, és* $\deg f = d$ *Ekkor*

$$f(x) = (x-\alpha)(x-\alpha^q)\cdots(x-\alpha^{q^{d-1}}).$$

Az $\alpha, \alpha^q, \ldots, \alpha^{q^{d-1}}$ elemek páronként különbözők.

Bizonyítás. Legyen $f(x) = a_0 + a_1 x + \cdots + x^d$. Ha $\beta \in \mathbb{F}$ és $f(\beta) = 0$, akkor a 18.23. lemmát és a 18.19. tételt alkalmazva

$$0 = f(\beta)^q = (a_0 + a_1\beta + \cdots + \beta^d)^q = a_0^q + a_1^q\beta^q + \cdots + \beta^{dq} = a_0 + a_1\beta^q + \cdots + \beta^{qd} = f(\beta^q),$$

vagyis β^q szintén gyöke f-nek.

Az $\alpha, \alpha^q, \ldots, \alpha^{q^{d-1}}$ elemek tehát valóban gyökei f-nek. Elég ezután megmutatnunk, hogy páronként különbözők. Tegyük fel indirekt módon, hogy $\alpha^{q^i} = \alpha^{q^j}$ és $0 \le i < j < d$. Legyen $\beta = \alpha^{q^i}$ és $l = j - i$. A feltevésünk szerint $\beta = \beta^{q^l}$, ami a 18.8. lemma szerint azt jelenti, hogy $f(x) \mid x^{q^l} - x$.

A 18.24. tétel alapján ekkor $d \mid l$, ami viszont ellentmondás, hiszen $l < d$. ■

A tétel rámutat, hogy a véges testek feletti irreducibilis polinomoknak nincs többszörös gyökük. Az f egy gyökéből az összes többi megkapható pusztán (q-adik) hatványozással.

Automorfizmusok

18.29. definíció. *Legyenek $\mathbb{F}_q \subseteq \mathbb{F}$ véges testek. A $\Psi : \mathbb{F} \to \mathbb{F}$ leképezés az $\mathbb{F}$ test $\mathbb{F}_q$-**automorfizmusa,** ha gyűrűizomorfizmus, és $\Psi(a) = a$ teljesül minden $a \in \mathbb{F}_q$ elemre.*

A $\Phi = \Phi_q : \mathbb{F} \to \mathbb{F}$ leképezést a következőképpen értelmezzük: $\Phi(\alpha) = \alpha^q$, ahol $\alpha \in \mathbb{F}$.

18.30. tétel. *Az $\mathbb{F} = \mathbb{F}_{q^d}$ test $\mathbb{F}_q$-automorfizmusai éppen a $\Phi, \Phi^2, \dots, \Phi^d = id$ leképezések.*

Bizonyítás. A 18.23. lemma alapján a Φ leképezés egy $\mathbb{F} \to \mathbb{F}$ gyűrűhomomorfizmus. A Φ nyilvánvalóan injektív, tehát izomorfizmus is. A 18.19. tételből következik, hogy Φ fixen hagyja az $\mathbb{F}_q$ elemeit. A Φ^j leképezések valóban $\mathbb{F}_q$-automorfizmusai $\mathbb{F}$-nek.

Legyen most $f(x) = a_0 + a_1 x + \dots + x^d \in \mathbb{F}_q[x]$, $\beta \in \mathbb{F}$, $f(\beta) = 0$, és Ψ egy $\mathbb{F}_q$-automorfizmusa $\mathbb{F}$-nek. Megmutatjuk, hogy ekkor $\Psi(\beta)$ is gyöke f-nek.

$$0 = \Psi(f(\beta)) = \Psi(a_0) + \Psi(a_1)\Psi(\beta) + \dots + \Psi(\beta)^d = f(\Psi(\beta)) \,.$$

Végül legyen β az $\mathbb{F}$ primitív eleme és $f \in \mathbb{F}_q[x]$ a β minimálpolinomja. A fenti észrevétel és a 18.28. tétel szerint $\Psi(\beta) = \beta^{q^j}$, ahol $0 \le j < d$, vagyis $\Psi(\beta) = \Phi^j(\beta)$. A Ψ és a Φ^j automorfizmusok ugyanoda képezik az $\mathbb{F}$ generátorelemét, amiből következik, hogy $\Psi = \Phi^j$. ■

Véges testek konstrukciója

Legyen $q = p^n$. A 18.7. tétel és a 18.26. következmény alapján $\mathbb{F}_q$ megkapható $\mathbb{F}[x]/(f)$ alakban, ahol $f \in \mathbb{F}[x]$ egy n-edfokú irreducibilis polinom. A számítástudományi alkalmazások területén ez a véges testek megadásának leggyakoribb módja. A 18.2. példabeli $f(x) = x^3 + x + 1$ segítségével az $\mathbb{F}_8$ testet kapjuk. A következő tétel szerint véletlen választással elég jó esélyünk van arra, hogy irreducibilis polinomot kapjunk.

18.31. tétel. *Legyen $f(x) \in \mathbb{F}_q[x]$, 1-főegyüttható, k-adfokú ($k > 1$), az egyenletes eloszlás szerint választott véletlen polinom (tehát egy f választásának a valószínűsége $1/q^k$). Ekkor az f legalább $1/k - 1/q^{k/2}$ valószínűséggel irreducibilis lesz $\mathbb{F}_q$ felett.*

Bizonyítás. Először becsüljük meg, hogy hány olyan $\alpha \in \mathbb{F}_{q^k}$ elem van, amelyre $\mathbb{F}_q(\alpha) = \mathbb{F}_{q^k}$. Az ilyen elemek száma legalább

$$|\mathbb{F}_{q^k}| - \sum_{r|k} |\mathbb{F}_{q^{k/r}}| \,,$$

ahol az összegezés a k különböző prímosztóira értendő. Valóban, ha α nem generálja $\mathbb{F}_q$ felett az $\mathbb{F}_{q^k}$ testet, akkor benne van az utóbbi test valamelyik maximális résztestében, ezek pedig a 18.27. tétel alapján éppen az $\mathbb{F}_{q^{k/r}}$ alakú testek. A k különböző prímosztóinak száma legfeljebb $\log_2 k$, így a kérdéses α elemek száma legalább $q^k - (\log_2 k)q^{k/2}$. Az ilyen α elemek (1-főegyütthatós) $\mathbb{F}_q$ feletti minimálpolinomja k-adfokú és irreducibilis. Egy polinomhoz éppen k különböző α tartozik (18.28. tétel). A különböző 1-főegyütthatós, k-adfokú, $\mathbb{F}_q[x]$-beli irreducibilis polinomok száma ezért legalább

$$\frac{q^k}{k} - \frac{(\log_2 k)q^{k/2}}{k} \ge \frac{q^k}{k} - q^{k/2} \,,$$

ahonnan q^k-val osztva adódik az állítás. ■

Ha tehát az $\mathbb{F}_q$ test birtokában annak $\mathbb{F}_{q^k}$ bővítését szeretnénk megkonstruálni, akkor érdemes ***véletlen***

$$f(x) = a_0 + a_1 x + \cdots + a_{k-1}x^{k-1} + x^k \in \mathbb{F}_q[x]$$

polinomot választanunk: az $a_0, \ldots, a_{k-1} \in \mathbb{F}_q$ együtthatókat egyenletes eloszlás szerint egymástól teljesen függetlenül sorsoljuk. A kapott polinom jó eséllyel (a valószínűség legalább $1/k - \epsilon$, ha q^k nagy) irreducibilis lesz, és ekkor $\mathbb{F}_q[x]/(f) \cong \mathbb{F}_{q^k}$. Várhatóan mintegy k számú f választásával kapunk irreducibilis polinomot.

A testbővítéseket irreducibilis polinomok segítségével kaphatjuk meg. Sokszor hasznos, ha ezeknek az polinomoknak további jó tulajdonságaik is vannak. Ilyen polinomok létezéséről szól a következő lemma.

18.32. lemma. *Legyen r egy prímszám. A q-elemű $\mathbb{F}_q$ testben akkor és csak akkor létezik olyan elem, amely nem r-edik hatvány, ha $q \equiv 1 \pmod{r}$. Ha $b \in \mathbb{F}_q$ egy ilyen elem, akkor az $x^r - b$ polinom irreducibilis $\mathbb{F}_q[x]$-ben és így $\mathbb{F}_q[x]/(x^r - b)$ egy q^r elemű test.*

Bizonyítás. Ha $r \nmid q-1$, legyen s egy olyan pozitív egész szám, amelyre $sr \equiv 1 \pmod{q-1}$. Ha $b \in \mathbb{F}_q \setminus 0$, akkor $(b^s)^r = b^{sr} = bb^{sr-1} = b$, ha pedig $b = 0$ akkor $b = 0^r$. Tehát ez esetben $\mathbb{F}_q$ minden eleme r-edik hatvány. Ha $r \mid q-1$, akkor legyen a egy primitív elem $\mathbb{F}_q$-ban. Ekkor $\mathbb{F}_q$-ban az r-edik hatványokat pontosan a következő $1 + (q-1)/r$ elem adja: $0, (a^r)^0, (a^r)^1, \ldots, (a^r)^{(q-1)/r-1}$. Tegyük fel, hogy $r^s \mid q-1$, de $r^{s+1} \nmid q-1$. Ekkor $b \in \mathbb{F}_q \setminus \{0\}$ rendje akkor és csak akkor osztható r^s-sel, ha b nem r-edik hatvány. Legyen b egy ilyen elem, és legyen $g(x) \in \mathbb{F}_q[x]$ az $x^r - b$ polinom egy irreducibilis tényezője. Tegyük fel, hogy $g(x)$ foka d. Nyilván $d \le r$. Ekkor $\mathbb{K} = \mathbb{F}_q[x]/(g(x))$ egy q^d elemű test és $\mathbb{K}$-ban az $[x]^r = b$ és így $[x]$ rendje osztható r^{s+1}-gyel. Következésképpen $r^{s+1} \mid q^d - 1$. Mivel $q-1$ nem osztató r^{s+1}-gyel, $r \mid (q^d-1)/(q-1) = 1 + q + \cdots + q^{d-1}$. Más szóval $1+q+\cdots+q^{d-1} \equiv 0 \pmod{r}$. Ugyanakkor $q \equiv 1 \pmod{r}$ miatt $1+q+\cdots+q^{d-1} \equiv d \pmod{r}$, tehát $d \equiv 0 \pmod{r}$, ami a $0 < d \le r$ egyenlőtlenségek miatt csak úgy lehet, hogy $d = r$. ■

Az előző lemma feltételei mellett az általános esetnél (18.31. tétel) sokkal kedvezőbb valószínűséggel tudunk véletlen választással irreducibilis polinomot kapni. Egy véletlen $b \in \mathbb{F}_q^*$ elem ugyanis $r/(q-1)$ valószínűséggel lesz r-edik hatvány.

18.33. állítás. *Legyen r prím és $r \mid q-1$. Ekkor egy véletlen $b \in \mathbb{F}_q^*$ elemmel az $x^r - b$ polinom $1 - r/(q-1)$ valószínűséggel irreducibilis lesz $\mathbb{F}_q[x]$-ben.*

Megjegyzés. Ha b nem r-edik hatvány $\mathbb{F}_q$-ban (tehát speciálisan $r \mid (q-1)$), és $r = 2$ esetén $4 \mid (q-1)$, akkor $[x]$ nem r-edik hatvány az $\mathbb{F}_q[x]/(x^r - b) \cong \mathbb{F}_{q^r}$ testben.

Bizonyítás. Az előző tétel érvelését követve elég azt belátni, hogy $r^2 \nmid (q^r-1)/(q-1)$. Ez a feltételek alapján nyilvánvaló, ha $r = 2$. Tegyük fel most, hogy $r > 2$, és legyen $q \equiv 1 + rt \pmod{r^2}$. Ekkor minden $i \ge 0$ egészre $q^i \equiv 1 + irt \pmod{r^2}$ és így

$$\frac{q^r - 1}{q-1} = 1 + q + \cdots + q^{r-1} \equiv r + \frac{r(r-1)}{2}rt \equiv r \pmod{r^2}$$

a feltevések miatt. ■

Gyakorlatok

18.2-1. Mutassuk meg, hogy az $x^{q+1}-1$ polinom lineáris tényezők szorzatára bomlik az $\mathbb{F}_{q^2}$ testben.

18.2-2. Mutassuk meg, hogy az $f(x)=x^4+x+1$ polinom irreducibilis $\mathbb{F}_2$ felett, tehát $\mathbb{F}_2[x]/(f)\cong\mathbb{F}_{16}$. Mennyi az $[x]_f$ elem rendje a maradékosztálygyűrűben? Igaz-e, hogy $[x]_f$ primitív elem $\mathbb{F}_{16}$-ban?

18.2-3. Határozzuk meg az $x^{31}-1$ irreducibilis tényezőit az $\mathbb{F}_2$ test felett.

18.2-4. Határozzuk meg az $\mathbb{F}_{3^6}$ test összes résztesteit.

18.2-5. Legyenek a, b pozitív egészek. Mutassuk meg, hogy van olyan (az $\mathbb{F}_q$-t tartalmazó) $\mathbb{K}$ véges test, amelyre $\mathbb{F}_{q^a}\subseteq\mathbb{K}$ és $\mathbb{F}_{q^b}\subseteq\mathbb{K}$. Mit mondhatunk a $\mathbb{K}$ test elemszámáról?

18.2-6. Igazoljuk, hogy az $\mathbb{F}_q$ feletti 1-főegyütthatós, k-adfokú irreducibilis polinomok száma legfeljebb q^k/k.

18.2-7. *a.* Legyen $\mathbb{F}$ test, V egy n-dimenziós vektortér $\mathbb{F}$ felett, $A: V\to V$ egy lineáris transzformáció, amelynek a minimálpolinomja megegyezik a karakterisztikus polinomjával. Mutassuk meg, hogy ekkor van olyan $v\in V$ vektor, amelyre a $v, Av, \dots, A^{n-1}v$ vektorok lineárisan függetlenek.

b. Az $S=\{\alpha,\alpha^q,\dots,\alpha^{q^{d-1}}\}$ elemhalmaz ***normálbázisa*** $\mathbb{F}_{q^d}$-nek $\mathbb{F}_q$ felett, ha $\alpha\in\mathbb{F}_{q^d}$ és S lineárisan független halmaz $\mathbb{F}_q$ felett. Bizonyítsuk be, hogy létezik $\mathbb{F}_{q^d}$-nek $\mathbb{F}_q$ feletti normálbázisa. *Útmutatás.* Mutassuk meg, hogy az $\mathbb{F}_q$-lineáris $\Phi:\mathbb{F}_{q^d}\to\mathbb{F}_{q^d}$ leképezés minimálpolinomja x^d-1, és alkalmazzuk az a. állítást.

18.3. Polinomok felbontása véges testek felett

Az algebrai kifejezések kezelésével kapcsolatos feladatok egyike a ***szorzattá alakítás.*** Ilyenkor a kifejezést, amivel dolgozunk, egyszerűbbek szorzataként állítjuk elő. Ennek a hasznossága nyilvánvaló a számításokat végző ember számára. Az erre szolgáló módszerek – ***felbontó algoritmusok*** – közül itt azokkal foglalkozunk, amelyek véges testek feletti egyváltozós polinomok felbontására alkalmasak.

A ***felbontási feladat*** bemenete egy $f(x)\in\mathbb{F}_q[x]$ polinom. A cél az f polinom

$$f=f_1^{e_1}f_2^{e_2}\cdots f_s^{e_s} \tag{18.3}$$

felbontásának meghatározása (kiszámítása), ahol $f_1,\dots,f_s$ páronként relatív prím, $\mathbb{F}_q$ felett irreducibilis (azaz felbonthatatlan) polinomok, és az e_i számok pozitív egészek. Tudjuk (18.4. tétel), hogy f lényegében egyértelműen meghatározza az f_i polinomokat és az e_i kitevőket is.

18.3. példa. Legyen $p=23$ és

$$f(x)=x^6-3x^5+8x^4-11x^3+8x^2-3x+1\,.$$

Ekkor, amint arról a tényezők modulo 23 összeszorzásával meggyőződhetünk,

$$f(x)=(x^2-x+10)(x^2+5x+1)(x^2-7x+7)\,.$$

Az x^2-x+10, x^2+5x+1, x^2-7x+7 tényezők egyikének sincs gyöke $\mathbb{F}_{23}$-ban, ezért szükségképpen irreducibilisek $\mathbb{F}_{23}[x]$-ben.

A felbontó algoritmusok fontos számítási eszközök, így a szimbolikus számítási rendszerek legtöbbje (Mathematica, Maple stb.) rendelkezik is ilyen eljárásokkal. A felbontó módszereket gyakran alkalmazzák a hibajavító kódolás és a kriptográfia területén.

A célunk azoknak az alapvető ötleteknek, algoritmikus építőköveknek a bemutatása, amelyek a feladat megoldásához használhatók. A hangsúlyt elsősorban a polinom idejű algoritmusok létezésére helyezzük. Terjedelmi okokból nem tudjuk részletesen tárgyalni a ma ismert legjobb megoldásokat.

18.3.1. Négyzetmentes felbontás

Az (18.3) felbontási feladat hatékonyan visszavezethető arra a speciális esetére, amikor a felbontandó f négyzetmentes, vagyis az e_i kitevők értéke 1. A visszavezetés alapja a 18.13. lemma és a következő állítás (az $f(x)$ polinom x szerinti deriváltját $f'(x)$ jelöli):

18.34. lemma. *Ha az $f(x) \in \mathbb{F}_q[x]$ polinomra $f'(x) = 0$, akkor létezik $g(x) \in \mathbb{F}_q[x]$, amelyre $f(x) = g(x)^p$.*

Bizonyítás. Legyen $f(x) = \sum_{i=0}^{n} a_i x^i$. Ekkor $f'(x) = \sum_{i=1}^{n} a_i i x^{i-1}$. Az $a_i i$ együttható akkor nulla az $\mathbb{F}_q$ testben, ha $a_i = 0$ vagy $p \mid i$. Tehát $f'(x) = 0$ esetén $f(x) = \sum_{j=0}^{k} b_j x^{pj}$ alakú. Legyen $q = p^d$, ekkor a $c_j = b_j^{p^{d-1}}$ választással $c_j^p = b_j^{p^d} = b_j$, így $f(x) = (\sum_{j=0}^{k} c_j x^j)^p$. ■

Ha $f'(x) = 0$, akkor az előző lemma alapján $f(x)$ négyzetmentes tényezőkre való felbontása a nála kisebb fokú $g(x)$ polinoméból nyerhető. Ha pedig $f'(x) \neq 0$, akkor a 18.13. lemma szerint az $f(x)/\mathrm{lnko}(f(x), f'(x))$ polinom már négyzetmentes és az $\mathrm{lnko}(f(x), f'(x))$ tényező vár további felbontásra. A polinomok osztása és a legnagyobb közös osztó számítása elvégezhető polinom időben (18.12. tétel). A $g(x)$ polinom számításakor szükségünk van $y^p = a$ alakú egyenletek megoldására $\mathbb{F}_q$-ban, ahol $a \in \mathbb{F}_q$. Ha $q = p^s$, akkor ennek az egyenletnek $y = a^{p^{s-1}}$ megoldása, ami ***gyors hatványozással*** (ismételt négyzetre emelésekkel, lásd 33.6.1. pont) polinom időben megkapható.

A kétféle egyszerűsítő lépés valamelyike mindig elvégezhető, ha f-nek van pozitív fokú négyzetosztója.

Egy polinom általában többféleképpen bontható fel négyzetmentes tényezőkre. Az egyértelműség kedvéért az $f \in \mathbb{F}[x]$ polinom ***négyzetmentes felbontásán*** a következő alakban történő előállítását értjük:

$$f = f_1^{e_1} \cdots f_s^{e_s} ,$$

ahol $e_1 < \cdots < e_s$ egészek, az f_i-k pedig páronként relatív prím négyzetmentes polinomok, más szóval f azonos multiplicitású irreducibilis tényezői egybe vannak gyűjtve. Az alábbi algoritmus kiszámítja az f polinom négyzetmentes felbontását. Az eddig vázolt észrevételeken túl a 18.14. lemmát használjuk még fel. Ez a 18.13. lemmával együtt azt garantálja, hogy egy véges test feletti f polinom egyszeres irreducibilis tényezőinek a szorzata $f/\mathrm{lnko}(f, f')$.

NÉGYZETMENTES-FELBONTÁS(f)

```
1  g ← f
2  S ← ∅
3  m ← 1
4  i ← 1
5  while deg g ≠ 0
6        do if g' = 0
7              then g ← ∛g
8                   i ← i · p
9              else h ← g/lnko(g, g')
10                  g ← g/h
11                  if deg h ≠ 0
12                    then S ← S ∪ (h, m)
13                  m ← m + i
14 return S
```

Mivel a ciklus minden egyes lefutása során g foka csökken és a felhasznált összetevők polinom időben megvalósíthatók, az eljárás polinom idejű.

18.3.2. Különböző fokú felbontás

Legyen f négyzetmentes polinom. Ebben a számítási fázisban f-et felbontjuk

$$f(x) = h_1(x)h_2(x)\cdots h_l(x) \tag{18.4}$$

alakba, ahol a $h_i(x) \in \mathbb{F}_q[x]$ polinom i-edfokú felbonthatatlanok szorzata. Ez a lépés nem feltétlenül szükséges a felbontási feladat megoldásához. Azért érdemes foglalkozni vele, mert vannak olyan módszerek, amelyek hatékonyan ki tudják aknázni a h_i polinomok sajátos szerkezetét. A különböző fokú felbontás kiindulópontja a következő tény:

18.35. tétel. *Az $x^{q^d} - x$ azon 1-főegyütthatós $f \in \mathbb{F}_q[x]$ irreducibilis polinomok egyszeres multiplicitással vett szorzata, amelyek foka osztója d-nek.*

Bizonyítás. $(x^{q^d} - x)' = -1$ tehát az irreducibilis tényezők egyszeresek. Ha $f \in \mathbb{F}_q[x]$ irreducibilis és osztója $x^{q^d} - x$-nek, akkor a 18.24. tétel alapján f foka osztója d-nek.

Fordítva, legyen $f \in \mathbb{F}_q[x]$ irreducibilis, f foka k és $k \mid d$. Ekkor a 18.27. tétel szerint f-nek van gyöke az $\mathbb{F}_{q^d}$ testben, amiből $f \mid x^{q^d} - x$ következik. ■

A tétel hatékony módszert kínál a $h_i(x)$ polinomok számítására. Először leválasztjuk f-ről a h_1-et, majd sorban a magasabb fokú tényezők szorzatait:

KÜLÖNBÖZŐ-FOKÚ-FELBONTÁS(f)

```
1  F ← f
3  for i ← 1 to deg f
4      do h_i ← lnko(F, x^{q^i} − x)
7         F ← F/h_i
8  return h_1, ..., h_{deg f}
```

Ha itt $F(x)$ konstans, akkor megállhatunk, a további lépések nem adnak érdemi új tényezőt. Az $x^{q^i}-x$ esetleges magas foka miatt az $\operatorname{lnko}(F(x), x^{q^i}-x)$ legnagyobb közös osztók számítása különös gondot igényel. Az érdemi gondolat itt az, hogy az $x^{q^i} \pmod{F(x)}$ maradékot gyors hatványozással célszerű számolni.

Az imént körvonalazott algoritmus alkalmas polinomok felbonthatatlanságának tesztelésére, ami a véges testek konstrukciójának egy fontos részfeladata. A különböző fokú felbontás itt bemutatott algoritmusa hatékonyan megoldja a feladatot. Nyilvánvaló ugyanis, hogy a k-adfokú f pontosan akkor irreducibilis, ha a (18.4) felbontásában $h_k(x) = f(x)$.

A következő változat valamivel hatékonyabb és a nem négyzetmentes bemeneteket is helyesen kezeli:

FELBONTHATATLANSÁG-TESZT(f)

```
1  n ← deg f
2  if x^{p^n} ≢ x (mod f)
3    then return "nem"
4  for n összes r prímosztójára
5      do if x^{p^{n/r}} ≡ x (mod f)
6           then return "nem"
7  return "igen"
```

A 2. és 5. sorban annak az ellenőrzése történik, hogy n-e az a legkisebb k természetes szám, amelyre f osztója az $x^k - x$ polinomnak. A 18.35. tétel miatt f pontosan ebben az esetben irreducibilis. Ha a 2. sorban a tesztet f túlélte, a 18.35. tételből tudjuk, hogy f négyzetmentes és a szóban forgó k csakis n osztója lehet. Összesen legfeljebb $\lg n+1$ (gyors) hatványozással modulo f eldönthető tehát f irreducibilitása.

18.36. tétel. *Ha $\mathbb{F}_q$ adott és $k > 1$ egész, akkor az $\mathbb{F}_{q^k}$ test $\lg q$-ban és k-ban polinom idejű Las Vegas-típusú véletlen algoritmussal megkonstruálható.*

Bizonyítás. Az algoritmus a következő:

VÉGES-TEST-KONSTRUKCIÓ(q^k)

```
1  for i ← 0 to k − 1
2      do a_i ← F_q egy véletlen eleme (egyenletes eloszlás szerint)
3  f ← x^k + Σ_{i=0}^{k−1} a_i x^i
4  if FELBONTHATATLANSÁG-TESZT(f) ="igen"
5    then return F_q[x]/(f)
6    else  return "nem sikerült"
```

Az 1–3. sorokban egyenletes eloszlás szerint választunk egy véletlen, 1-főegyüttható k-adfokú $f(x) \in \mathbb{F}_q[x]$ polinomot, majd a 4. sorban hatékonyan ellenőrizzük, hogy $f(x)$ irreducibilis-e. A 18.31. tétel alapján f jó eséllyel irreducibilis lesz. ■

18.3.3. A Cantor–Zassenhaus-algoritmus

Itt a felbontási feladatnak azzal az esetével foglalkozunk, amikor q páratlan és az $f(x) \in \mathbb{F}_q[x]$ polinom

$$f = f_1 f_2 \cdots f_s \tag{18.5}$$

alakú, ahol az f_i polinomok páronként relatív prím d-edfokú irreducibilis polinomok $\mathbb{F}_q[x]$-ben, továbbá $s \geq 2$. A négyzetmentes és a különböző fokú tényezőkre történő felbontás ugyanis visszavezeti az általános felbontási feladatot ilyen részfeladatokra. Páros q esetére Berlekampnak a következő részben bemutatandó determinisztikus módszere ad polinom idejű megoldást. Az itt tárgyalthoz hasonló, páros q-ra működő módszert vázolunk a 18-2. feladatban.

18.37. lemma. *Legyen q páratlan. Ekkor $(q^2-1)/2$ olyan $(c_1, c_2) \in \mathbb{F}_q \times \mathbb{F}_q$ pár van, amelyre $c_1^{(q-1)/2}$ és $c_2^{(q-1)/2}$ közül pontosan az egyik 1.*

Bizonyítás. Legyen a egy primitív elem $\mathbb{F}_q$-ban: $a^{q-1} = 1$, de $a^k \neq 1$, ha $0 < k < q-1$. Ekkor $\mathbb{F}_q \setminus \{0\} = \{a^s | s = 0, \ldots, q-2\}$, továbbá, mivel $\left(a^{(q-1)/2}\right)^2 = 1$, de $a^{(q-1)/2} \neq 1$, azt kapjuk, hogy $a^{(q-1)/2} = -1$. Ezért $a^{s(q-1)/2} = (-1)^s$, tehát a $c \in \mathbb{F}_q \setminus \{0\}$ elemek felére $c^{(q-1)/2} = 1$, másik felére pedig $c^{(q-1)/2} = -1$. A $c = 0$ esetben nyilván $c^{(q-1)/2} = 0$. Ezért $((q-1)/2)((q+1)/2)$ olyan (c_1, c_2) pár van, amelyre $c_1^{(q-1)/2} = 1$, de $c_2^{(q-1)/2} \neq 1$; és ugyanennyi olyan, amelyre fordított a helyzet. Tehát a feltételnek megfelelő párok száma $(q-1)(q+1)/2 = (q^2-1)/2$. ■

18.38. tétel. *Tegyük fel, hogy q páratlan, és az $f(x) \in \mathbb{F}_q[x]$ n-edfokú (18.5) alakú polinom. Válasszunk az egyenletes eloszlás szerint egy véletlen n-nél alacsonyabb fokú $u(x) \in \mathbb{F}_q[x]$ polinomot (azaz válasszunk egymástól függetlenül, egyenletes valószínűséggel n véletlen $u_0, \ldots, u_{n-1}$ elemet, és tekintsük az $u(x) = \sum_{i=0}^{n-1} u_i x^i$ polinomot)! Ekkor az*

$$\operatorname{lnko}(u(x)^{\frac{q^d-1}{2}} - 1, f(x))$$

legnagyobb közös osztó legalább $(q^{2d}-1)/(2q^{2d}) \geq 4/9$ valószínűséggel valódi osztója az $f(x)$ polinomnak.

Bizonyítás. Az $u(x) \pmod{f_i(x)}$ elem az $F[x]/(f_i(x)) \cong \mathbb{F}_{q^d}$ maradékosztálytest egy elemének felel meg. A kínai maradéktétel (18.15. tétel) alapján $u(x)$ egyenletes választása azt jelenti, hogy $u(x)$ maradékait az $f_i(x)$ tényezők szerint egymástól függetlenül, egyenletes valószínűséggel választjuk. A 18.37. lemma alapján annak valószínűsége, hogy az $u(x)^{(q^d-1)/2} - 1$ polinom $f_1(x)$, illetve $f_2(x)$ szerinti maradéka közül pontosan az egyik 0, éppen $(q^{2d}-1)(2q^{2d})$. Ebben az esetben a tételbeli legnagyobb közös osztó valódi osztója lesz f-nek. Ugyanis, ha például $u(x)^{(q^d-1)/2} - 1 \equiv 0 \pmod{f_1(x)}$, de ez a kongruencia nem

áll fenn modulo $f_2(x)$, akkor az $u(x)^{(q^d-1)/2} - 1$ polinom osztható az $f_1(x)$ tényezővel, de $f_2(x)$-szel nem, tehát az $f(x)$ polinommal vett legnagyobb közös osztója tényleg egy valódi osztó. A

$$\frac{q^{2d}-1}{2q^{2d}} = \frac{1}{2} - \frac{1}{2q^{2d}}$$

mennyiség monoton növő függvénye q^d-nek. A számunkra érdekes tartományban akkor lesz a legkisebb, ha q^d a legkisebb páratlan prímhatvány, azaz 3. A minimum tehát $1/2 - 1/18 = 4/9$. ■

A tétel Las Vegas típusú véletlent használó, polinom idejű módszert ad a (18.5) alakú polinomok két tényezőre bontására:

CANTOR–ZASSENHAUS-PÁRATLAN(f, d)

```
1  n ← deg f
2  for i ← 0 to n − 1
3      do u_i ← 𝔽_q egy véletlen eleme (egyenletes eloszlás szerint)
4  u ← Σ_{i=0}^{n−1} u_i x^i
5  g ← lnko(u^{(q^d−1)/2} − 1, f)
6  if 0 < deg g < deg f
7     then return(g, f/g)
8     else return "nem sikerült"
```

A kapott tényezők (amennyiben nem felbonthatatlanok) ugyancsak (18.5) alakúak, ezért rájuk az eljárás ismételhető. Ezáltal véletlent használó, polinom idejű eljárást kapunk az f teljes felbontására.

A legnagyobb közös osztó számításakor az $u(x)^{(q^d-1)/2} \pmod{f(x)}$ maradékot itt is célszerű gyors hatványozással számítani.

Az eddigiek alapján megállapíthatjuk, hogy az általános (18.3) felbontási feladat páratlan elemszámú testek felett véletlent használó algoritmussal polinom időben megoldható.

18.3.4. Berlekamp algoritmusa

Itt egy olyan felbontási algoritmust körvonalazunk, amely a feladatot lényegében az alaptestben, illetve a prímtestben való keresésre vezeti vissza. Legyen tehát

$$f(x) = f_1^{e_1}(x) \cdots f_s^{e_s}(x) \,,$$

ahol $f_i(x)$ $(i = 1, \ldots, s)$ páronként nem asszociált irreducibilis polinomok $\mathbb{F}_q[x]$-ben, és n az $f(x)$ polinom foka. A kínai maradéktétel (18.15. tétel) izomorfizmust létesít az $\mathbb{F}_q[x]/(f)$ és az

$$\mathbb{F}_q[x]/(f_1^{e_1}) \oplus \cdots \oplus \mathbb{F}_q[x]/(f_s^{e_s})$$

gyűrűk között. A megfeleltetés a következő:

$$[u(x)]_f \leftrightarrow ([u(x)]_{f_1^{e_1}}, \ldots, [u(x)]_{f_s^{e_s}}) \,,$$

ahol $u(x) \in \mathbb{F}_q[x]$.

A Berlekamp-algoritmus legfontosabb technikai eszköze az $\mathbb{F}_q[x]/(f(x))$ maradékosztálygyűrűben a p-edik (illetve q-adik) hatványra emelés. A kínai maradéktétel megfeleltetése a p-edik, illetve q-adik hatványra emelésre a következőképpen néz ki:

$$[u(x)]^p \quad \leftrightarrow \quad ([u(x)^p]_{f_1^{e_1}}, \ldots, [u(x)^p]_{f_s^{e_s}}) \,, \tag{18.6}$$

$$[u(x)]^q \quad \leftrightarrow \quad ([u(x)^q]_{f_1^{e_1}}, \ldots, [u(x)^q]_{f_s^{e_s}}) \,. \tag{18.7}$$

Az $f = f(x)$ polinom B_f ***Berlekamp-részalgebrája*** az $\mathbb{F}_q[x]/(f)$ maradékosztálygyűrűnek az a részgyűrűje, ami a q-adik hatványra emelés fixpontjaiból áll. Az A_f-fel jelölt ***abszolút Berlekamp-részalgebra*** pedig a p-edik hatványra emelés fixpontjaiból áll:

$$B_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) : \ [u(x)^q]_f = [u(x)]_f\} \,,$$

$$A_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) : \ [u(x)^p]_f = [u(x)]_f\} \,.$$

Nyilvánvaló, hogy $A_f \subseteq B_f$. A részalgebra elnevezést az indokolja, hogy mindkettő részgyűrűje az $\mathbb{F}_q[x]/(f(x))$ maradékosztálygyűrűnek (vagyis zártak a modulo $f(x)$ végzett összeadásra és szorzásra), valamint B_f ezen kívül egy $\mathbb{F}_q$ feletti lineáris altér is, azaz zárt az $\mathbb{F}_q$ elemeivel vett szorzásra is. Az A_f abszolút Berlekamp-részalgebra csak az $\mathbb{F}_p$ prímtest elemeivel való szorzásra zárt.

A B_f Berlekamp-részalgebra azért altér, mert az $u \mapsto u^q - u \pmod{f(x)}$ leképezés az $\mathbb{F}_q[x]/g(x)$ maradékosztálygyűrűnek egy önmagába való $\mathbb{F}_q$-lineáris leképezését adja a 18.23. lemma és a 18.19. tétel szerint. Így B_f (egy bázisa) egy $\mathbb{F}_q$ feletti homogén lineáris egyenletrendszer megoldásaként számítható ki, például a következőképpen:

Legyen tetszőleges $i \in \{0, \ldots, n-1\}$-re $h_i(x)$ az a legfeljebb $(n-1)$-edfokú polinom, amelyre $x^{iq} - x^i \equiv h_i(x) \pmod{f(x)}$. A h_i polinomok gyors hatványozás segítségével egyenként $O(\lg q)$ polinomszorzással és maradékos osztással kiszámíthatók. Legyen $h_i(x) = \sum_{j=0}^{n} h_{ij} x^j$. Az $u(x) = \sum_{i=0}^{n-1} u_i x^i$ n-nél alacsonyabb fokú polinom $[u]_f$ osztálya akkor és csak akkor esik bele a Berlekamp-részalgebrába, ha

$$\sum_{i=0}^{n-1} u_i h_i(x) = 0 \,,$$

ami $j = 0, \ldots, n-1$-re az x^j együtthatóját tekintve a következő n változós, n homogén lineáris egyenletből álló egyenletrendszerre vezet:

$$\sum_{i=0}^{n-1} h_{ij} u_i = 0, \quad (j = 0, \ldots, n-1) \,.$$

Hasonlóan, az abszolút Berlekamp-részalgebra (egy $\mathbb{F}_p$ feletti bázisának) kiszámítása megoldható egy nd változós nd homogén lineáris egyenletrendszer megoldásával az $\mathbb{F}_p$ prímtest felett. Egy kézenfekvő megközelítés a következő:

Az $\mathbb{F}_q$ test elemeit a szokásos módon reprezentáljuk, azaz d-nél alacsonyabb fokú $\mathbb{F}_p[y]$-beli polinomokként, és a műveleteket modulo $g(y)$ végezzük, ahol $g(y) \in \mathbb{F}_p[y]$ egy d-edfokú irreducibilis polinom az $\mathbb{F}_p$ prímtest fölött. Ekkor az $u[x] \in \mathbb{F}_q[x]$ n-nél alacsonyabb fokú

polinom

$$\sum_{i=0}^{n-1}\sum_{j=0}^{d-1} u_{ij}y^j x^i$$

alakba írható, ahol $u_{ij} \in \mathbb{F}_p$. Legyen $i \in \{0,\dots,n-1\}$-re és $j \in \{0,\dots,d-1\}$-re most $h_{ij}(x) \in \mathbb{F}_q[x]$ az a legfeljebb $(n-1)$-edfokú polinom, amelyre $h_{ij}(x) \equiv (y^j x^i)^p - y^j x^i \pmod{f(x)}$. A $h_{ij}(x)$ polinom $\sum_{k=0}^{n-1}\sum_{l=0}^{d-1} h_{ij}^{kl} y^l x^k$ alakba írható. Az $u[x] = \sum_{i=0}^{n-1}\sum_{j=0}^{d-1} u_{ij}y^j x^i$ polinomra az $[u]$ abszolút Berlekamp-részalgebrába tartozásának feltétele

$$\sum_{i=0}^{n-1}\sum_{j=0}^{d-1} u_{ij}h_{ij}(x) = 0\ ,$$

ami az $y^l x^k$ monomok együtthatóit tekintve a következő egyenletrendszerrel egyenértékű:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{d-1} h_{ij}^{kl}u_{ij} = 0 \quad (k = 0,\dots,n-1,\ l = 0,\dots,d-1)\ .$$

Ez pedig az u_{ij} változókban valóban egy homogén lineáris egyenletrendszer. A test feletti lineáris egyenletrendszerek polinom időben megoldhatók (lásd 31.4. alfejezet), az $\mathbb{F}_q[x]/(f(x))$ gyűrűben az alapműveletek hatékonyan elvégezhetők, továbbá a gyors hatványozás is megy polinom időben. Érvényes tehát a következő

18.39. tétel. *Legyen $f \in \mathbb{F}_q[x]$. Ekkor a $B_f \le \mathbb{F}_q[x]/(f(x))$ és az $A_f \le \mathbb{F}_q[x]/(f(x))$ Berlekamp-részalgebrák hatékonyan számíthatók abban az értelemben, hogy B_f egy $\mathbb{F}_q$-bázisa, illetve A_f egy $\mathbb{F}_p$-bázisa determinisztikus polinom időben megkapható.*

A (18.6) és a (18.7) képletek alapján

$$B_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) :\ [u^q(x)]_{f_i^{e_i}} = [u(x)]_{f_i^{e_i}}\ \ (i = 1,\dots,s)\} \tag{18.8}$$

és

$$A_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) :\ [u^p(x)]_{f_i^{e_i}} = [u(x)]_{f_i^{e_i}}\ \ (i = 1,\dots,s)\}\ . \tag{18.9}$$

A következő tétel azt állítja, hogy a Berlekamp-részalgebra elemei egyszerűen jellemezhetők a kínai maradékaikkal.

18.40. tétel.

$$B_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) :\ \exists c_i \in \mathbb{F}_q \text{ hogy } [u(x)]_{f_i^{e_i}} = [c_i]_{f_i^{e_i}}\ \ (i = 1,\dots,s)\}$$

és

$$A_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) :\ \exists c_i \in \mathbb{F}_p \text{ hogy } [u(x)]_{f_i^{e_i}} = [c_i]_{f_i^{e_i}}\ \ (i = 1,\dots,s)\}\ .$$

Bizonyítás. A kínai maradéktétel és a (18.8), illetve (18.9) formulák alapján elegendő azt igazolni, hogy tetszőleges $g(x), u(x) \in \mathbb{F}_q[x]$ irreducibilis polinomokra és e pozitív egészre

$$u^q(x) \equiv u(x) \pmod{g^e(x)} \iff \exists c \in \mathbb{F}_q \text{ amelyre } u(x) \equiv c \pmod{g^e(x)}\ ,$$

illetve

$$u^p(x) \equiv u(x) \pmod{g^e(x)} \Longleftrightarrow \exists c \in \mathbb{F}_p \text{ amelyre } u(x) \equiv c \pmod{g^e(x)} .$$

Mindkét esetben a $\Leftarrow$ irányú implikáció egyszerű következménye a 18.19. tételnek. Az $\mathbb{F}_p = \{a \in \mathbb{F}_q | a^p = a\}$ egyenlőség miatt az abszolút Berlekamp-részalgebrára vonatkozó $\Rightarrow$ irányú implikáció következik a Berlekamp-részalgebrára vonatkozóból, elegendő tehát ez utóbbival foglalkozni.

Az $\mathbb{F}_q[x]/(g(x))$ maradékosztálygyűrű test, így az $x^q - x$ polinomnak legfeljebb q gyöke van $\mathbb{F}_q[x]/(g(x))$-ben. A 18.19. tételből ismerünk is q különböző gyököt: ezek az $\mathbb{F}_q$ test elemei (a konstans polinomok modulo $g(x)$). Tehát

$$u^q(x) \equiv u(x) \pmod{g(x)} \Longleftrightarrow \exists c \in \mathbb{F}_q \text{ amelyre } u(x) \equiv c \pmod{g(x)} .$$

Ezért ha $u^q(x) \equiv u(x) \pmod{g^e(x)}$, akkor $u(x) = c + h(x)g(x)$ alakú, ahol $h(x) \in \mathbb{F}_q[x]$. Legyen most N egy tetszőleges pozitív egész szám. Ekkor

$$u(x) \equiv u^q(x) \equiv u^{q^N}(x) \equiv (c + h(x)g(x))^{q^N} \equiv c + h(x)^{q^N} g(x)^{q^N} \equiv c \pmod{g^{q^N}(x)} .$$

Ha itt N-et akkorának választjuk, hogy $q^N \geq e$ teljesüljön, akkor a fenti kongruencia miatt $u(x) \equiv c \pmod{g^e(x)}$ is igaz. ■

A B_f, illetve A_f egy $[u(x)]_f$ elemét ***nemtriviálisnak*** nevezzük, ha nem létezik olyan $c \in \mathbb{F}_q$ elem, amelyre $u(x) \equiv c \pmod{f(x)}$. Az előző tétel és a kínai maradéktétel alapján ez éppen akkor teljesül, ha van olyan i, j, hogy $c_i \neq c_j$. Ehhez pedig nyilvánvalóan szükséges feltétel, hogy $s > 1$, azaz $f(x)$-nek legyen legalább két különböző irreducibilis tényezője.

18.41. lemma. *Legyen $[u(x)]_f$ a B_f Berlekamp-részalgebra egy nemtriviális eleme. Ekkor létezik olyan $c \in \mathbb{F}_q$, amelyre* lnko$(u(x) - c, f(x))$ *az $f(x)$ polinom valódi osztója. Ha $[u(x)]_f \in A_f$, akkor létezik az $\mathbb{F}_p$ prímtestbe eső ilyen c elem is.*

Bizonyítás. Legyenek $1 \leq i, j \leq n$ és $c_i \neq c_j \in \mathbb{F}_q$, amelyekre $u(x) \equiv c_i \pmod{f_i^{e_i}(x)}$, illetve $u(x) \equiv c_j \pmod{f_j^{e_j}(x)}$. Ekkor a $c = c_i$ választással az $u(x) - c$ polinom osztható $f_i^{e_i}(x)$-szel, de nem osztható $f_j^{e_j}(x)$-szel. Ha $u(x) \in A_f$, akkor $c = c_i \in \mathbb{F}_p$ is teljesül. ■

Tegyük fel, hogy a kezünkben van A_f egy bázisa. A báziselemek közül legfeljebb egy lehet triviális, ugyanis a triviális elemek éppen az egységelem skalárszorosai. Ha $f(x)$ nem egy irreducibilis polinom hatványa, akkor biztosan lesz nemtriviális $[u(x)]_f$ báziselem is, így az $f(x)$ az előző lemmában megfogalmazott ötletet követve két tényezőre bontható. A $c \in \mathbb{F}_p$ elemeket sorra véve számoljuk az lnko$(u(x) - c, f(x))$ legnagyobb közös osztót.

18.42. tétel. *Az $f(x) \in \mathbb{F}_q[x]$ polinom felbontható olyan determinisztikus algoritmussal, aminek az időköltsége polinomiális a p,* deg f *és* log q *paraméterekben.*

Bizonyítás. Elég belátni, hogy f felbontható *két* tényező szorzatára az adott időkorláton belül. Az eljárás ugyanis ismételhető a kapott tényezőkkel.

BERLEKAMP-DETERMINISZTIKUS(f)

```
1  S ← A_f egy bázisa
2  if |S| > 1
3     then u ← S egy nemtriviális eleme
4     for c ∈ F_p
5          do g ← lnko(u − c, f)
6             if 0 < deg g < deg f
7                then return (g, f/g)
8     else return "irreducibilis hatványa"
```

Először (1. sor) meghatározzuk az A_f abszolút Berlekamp-részalgebra egy bázisát. Ennek a költsége polinomiális a $\deg f$ és $\lg q$ paraméterekben.

A második fázisban (2–7. sorok) egy nemtriviális $[u(x)]_f$ báziselemet véve sorra számítjuk az $\text{lnko}(u(x)-c, f(x))$ legnagyobb közös osztókat ($c \in \mathbb{F}_p$). Ennek költsége polinomja a p és $\deg f$ mennyiségeknek.

Ha nincsen nemtriviális báziselem, akkor A_f 1-dimenziós és f az irreducibilis f_1 polinom e_1-edik hatványa, ahol f_1 és e_1 például az $f_1 = f/\text{lnko}(f, f')$ és $e_1 = \deg f/\deg f_1$ formulák segítségével határozható meg. ■

A kapott időkorlát *nem polinomiális* a bemenet hosszához mérve, mert abban p szerepel $\log p$ helyett. Ha viszont a p kicsi a többi paraméterhez képest (például a kódelmélet szempontjából fontos $p = 2$ esetben), akkor a korlát már polinomiális lesz a bemenet méretében.

18.43. következmény. *Tegyük fel, hogy a p korlátozható a $\deg f$ és $\lg q$ paraméterek egy polinomjával. Ekkor az f felbontása determinisztikus polinom időben megkapható.*

Az előző két alapvető eredmény E. R. Berlekamptól származik. A témakör legfontosabb nyitott kérdése, hogy létezik-e determinisztikus polinom idejű módszer a felbontási feladat megoldására. A kérdés elsősorban elméleti jelentőségű, ugyanis a gyakorlatban igen hatékonynak bizonyultak a véletlent használó polinom idejű módszerek, így például a Cantor-Zassenhaus–algoritmus.

Berlekamp véletlenített algoritmusa

A Berlekamp-részalgebrák segítségével is kaphatunk hatékony véletlenített algoritmust. Tegyük fel, hogy q páratlan, és mint előbb, $f \in \mathbb{F}_q[x]$ a felbontandó polinom.

Legyen $[u(x)]_f$ véletlen elem a B_f Berlekamp-részalgebrából. A Cantor-Zassenhaus–algoritmus elemzéséhez hasonlóan belátható, hogy az $\text{lnko}(u(x)^{(q-1)/2} - 1, f(x))$ legnagyobb közös osztó legalább 4/9 valószínűséggel valódi osztója lesz $f(x)$-nek, feltéve, hogy $f(x)$-nek egynél több különböző irreducibilis tényezője van. Ennek az ötletnek egy olyan változatát mutatjuk be, amely egy kicsit takarékosabban bánik a véletlen bitekkel. A B_f véletlen eleme helyett csupán az $\mathbb{F}_q$-ból választunk véletlen elemet.

18.44. lemma. *Tegyük fel, hogy q páratlan és $a_1 \neq a_2 \in \mathbb{F}_q$. Ekkor legalább $(q-1)/2$ olyan $b \in \mathbb{F}_q$ elem van, amelyre az $(a_i + b)^{(q-1)/2}$ ($i \in \{1, 2\}$) elemek közül pontosan az egyik 1.*

Bizonyítás. A 18.37. lemma bizonyításának elején szereplő gondolatmenet segítségével látható, hogy az $\mathbb{F}_q\backslash\{1\}$ halmaznak $(q-1)/2$ olyan eleme van, amelynek $(q-1)/2$-edik hatványa -1.

Egyszerűen igazolható az is, hogy adott $c \in \mathbb{F}_q\backslash\{1\}$ elemhez pontosan egy olyan $b \neq -a_2$ létezik, amelyre $c = (a_1+b)(a_2+b)$. A keresett b ugyanis egy lineáris egyenlet megoldása.

A fentiek miatt $(q-1)/2$ olyan $b \in \mathbb{F}_q \setminus \{-a_2\}$ elem van, amelyre $((a_1+b)/(a_2+b))^{(q-1)/2} = -1$. Egy ilyen b esetén $(a_1+b)^{(q-1)/2}$ és $(a_2+b)^{(q-1)/2}$ közül az egyik $+1$, a másik -1. ■

18.45. tétel. *Tegyük fel, hogy q páratlan, és az $f(x) \in \mathbb{F}_q[x]$ polinomnak van legalább két különböző $\mathbb{F}_q[x]$-ben irreducibilis tényezője. Legyen $u(x)$ a B_f Berlekamp-részalgebra egy nemtriviális eleme. Ekkor, ha egyenletes valószínűséggel választunk egy $b \in \mathbb{F}_q$ elemet, akkor legalább $(q-1)/(2q) \geq 1/3$ valószínűséggel teljesül, hogy az* $\operatorname{lnko}((u(x)+b)^{(q-1)/2}-1, f(x))$ *legnagyobb közös osztó az $f(x)$ polinom valódi osztója.*

Bizonyítás. Legyen $f(x) = \prod_{i=1}^{s} f_i^{e_i}(x)$, ahol az $f_i(x)$ tényezők páronként különböző irreducibilis polinomok. Az $[u(x)]_f$ nemtriviális eleme a Berlekamp-részalgebrának, ezért léteznek olyan $0 < i, j \leq s$ indexek és $c_i \neq c_j \in \mathbb{F}_q$ elemek, amelyekre $u(x) \equiv c_i \pmod{f_i^{e_i}(x)}$ és $u(x) \equiv c_j \pmod{f_j^{e_j}(x)}$. Alkalmazzuk a 18.44. lemmát az $a_1 = c_i$, $a_2 = c_j$ szereposztással! Azt kapjuk, hogy ha b az $\mathbb{F}_q$ véletlen eleme, akkor legalább $(q-1)/(2q)$ valószínűséggel teljesül, hogy a $(c_i+b)^{(q-1)/2}-1$ és $(c_j+b)^{(q-1)/2}-1$ elemek közül pontosan az egyik 0. Ha például $(c_i+b)^{(q-1)/2}-1 = 0$, de $(c_j+b)^{(q-1)/2}-1 \neq 0$, akkor $(u(x)+b)^{(q-1)/2}-1 \equiv 0 \pmod{f_i^{e_i}(x)}$ de $(u(x)+b)^{(q-1)/2}-1 \not\equiv 0 \pmod{f_j^{e_j}(x)}$, vagyis az $(u(x)+b)^{(q-1)/2}-1$ polinom osztható $f_i^{e_i}(x)$-szel, de nem osztható $f_j^{e_j}(x)$-szel. Ezért az $\operatorname{lnko}(f(x), (u(x)+b)^{(q-1)/2}-1)$ legnagyobb közös osztó valódi osztója f-nek.

A $(q-1)/(2q) = 1/2 - 1/(2q)$ mennyiség monoton növekvő függvénye q-nak, ezért minimumát a legkisebb páratlan prímhatványnál, 3-nál veszi fel. A minimális érték 1/3. ■

A tétel alapján a következő algoritmus írható fel (ismét csak a két tényezőre bontásra adunk módszert):

BERLEKAMP-VÉLETLENÍTETT(f)

```
1  S ← B_f egy bázisa
2  if |S| > 1
3     then u ← S egy nemtriviális eleme
4          c ← F_q egy véletlen eleme (egyenletes eloszlás szerint)
5          g ← lnko((u − c)^((q−1)/2), f)
6          if 0 < deg g < deg f
7             then return (g, f/g)
8             else return "nem sikerült"
9     else return "irreducibilis hatványa"
```

Gyakorlatok

18.3-1. Legyen α egy adott eleme az $\mathbb{F}_p[x]/(f(x))$ testnek (itt $f(x) \in \mathbb{F}_p[x]$ egy irreducibilis

polinom). Adjunk polinom idejű algoritmust az α^{-1} kiszámítására. *Útmutatás.* Használjuk a 18.1-6. gyakorlat eredményét.

18.3-2. Legyen $f(x) = x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \in \mathbb{F}_2[x]$. A különböző fokú felbontás algoritmusával határozzuk meg az f polinom (18.4) felbontását.

18.3-3. A Cantor–Zassenhaus-algoritmussal bontsuk fel az $x^2 + 2x + 9 \in \mathbb{F}_{11}[x]$ polinomot.

18.3-4. Legyen $f(x) = x^2 - 3x + 2 \in \mathbb{F}_5[x]$. Mutassuk meg, hogy $\mathbb{F}_5[x]/(f(x))$ egybeesik az f abszolút Berlekamp-részalgebrájával: $A_f = \mathbb{F}_5[x]/(f(x))$.

18.3-5. Legyen $f(x) = x^3 - x^2 + x - 1 \in \mathbb{F}_7[x]$. A Berlekamp-algoritmussal határozzuk meg az f irreducibilis tényezőit: először keressük meg az A_f Berlekamp-részalgebra egy nemtriviális elemét, majd ezzel bontsuk f-et.

18.4. Rácsredukció

A fejezet hátralevő részében elsődleges célunk a racionális együtthatós polinomok felbontására szolgáló Lenstra–Lenstra–Lovász algoritmus ismertetése. Ehhez először egy önmagában is igen érdekes, geometriai jellegű kérdéskörrel, a rácsokban való rövid vektorok keresésével foglalkozunk. A legrövidebb nem-nulla rácsvektor megkeresése nehéz: Ajtai eredménye szerint ha ez véletlent használó, polinom időben megoldható lenne, akkor az összes NP-beli feladat megoldható lenne véletlent használó, polinom idejű algoritmussal. Az ebben a részben bemutatandó rácsredukciós eljárás polinom időben olyan rácsvektort állít elő, amelynek hossza $2^{(n-1)/4}$-szeres szorzótényezőn belül megközelíti a legrövidebb nem 0 rácsvektorét.

18.4.1. Rácsok

Szükségünk lesz néhány a valós vektorterekkel kapcsolatos alapfogalomra. Jelölje $\mathbb{R}^n$ az n-komponensű valós oszlopvektorok összességét. Könnyen látható, hogy $\mathbb{R}^n$ vektortér az $\mathbb{R}$ test felett. Az $\mathbb{R}^n$-beli $u = (u_1, \ldots, u_n)$ és $v = (v_1, \ldots, v_n)$ vektorok ***skalárszorzata*** az $(u, v) = u_1v_1 + u_2v_2 + \cdots + u_nv_n$ szám. Az $|u| = \sqrt{(u, u)}$ mennyiség az u vektor hossza. Az u, v vektorok ***merőlegesek,*** vagy ***ortogonálisak,*** ha $(u, v) = 0$. Az $\mathbb{R}^n$ egy $b_1, \ldots, b_n$ bázisa ***ortonormált***, ha $(b_i, b_i) = 1$ minden i-re, és $(b_i, b_j) = 0$, ha $i \neq j$.

A valós mátrixok rangjával, determinánsával, definit voltával kapcsolatos alapfogalmak megtalálhatók a 31.1. alfejezetben.

18.46. definíció. *Egy $L \subseteq \mathbb{R}^n$ halmazt **rácsnak** nevezünk, ha L részcsoport az összeadásra, továbbá L diszkrét abban az értelemben, hogy $\mathbb{R}^n$ minden korlátos tartományába L-nek csak véges sok pontja esik. Az L rács **rangja** az általa kifeszített altér dimenziója. Nyilvánvaló, hogy L rangja az L elemeiből kiválasztható maximális lineárisan független rendszer elemszáma. Ha ez n, akkor L-et **teljes** rácsnak nevezzük. Az L-be tartozó vektorokat **rácsvektoroknak** vagy **rácspontoknak** nevezzük.*

18.47. definíció. *Legyenek $b_1, \ldots, b_r$ lineárisan független elemek az $L \subseteq \mathbb{R}^n$ rácsból. Ha L minden eleme felírható a $b_1, \ldots, b_r$ elemek egész együtthatós lineáris kombinációjaként, akkor a $b_1, \ldots, b_r$ rendszert L egy **bázisának** nevezzük.*

Megjegyezzük, hogy a lineáris függetlenség miatt $\mathbb{R}^n$ minden eleme legfeljebb egyféleképp írható fel a $b_1, \ldots, b_r$ valós együtthatós lineáris kombinációjaként.

A következő tétel szerint az $\mathbb{R}^n$ additív részcsoportjai között a rácsok jellemezhetők a bázisok létezésével.

18.48. tétel. *Legyenek $b_1, \ldots, b_r$ lineárisan független vektorok $\mathbb{R}^n$-ből és álljon az L halmaz a $b_1, \ldots, b_r$ vektorok egész együtthatós lineáris kombinációiból. Ekkor L rács és $b_1, \ldots, b_r$ egy bázis L-ben.*
Fordítva, ha L egy rács $\mathbb{R}^n$-ben, akkor létezik bázisa.

Bizonyítás. Az első állításnak az a része nyilvánvaló, hogy L részcsoport, azaz zárt az összeadásra és a kivonásra. A diszkrétség bizonyításához feltehető, hogy $n = r$, mert a $b_1, \ldots, b_r$ által feszített altér izomorf $\mathbb{R}^r$-rel. Legyen tehát $n = r$. Ekkor a $\phi : (\alpha_1, \ldots, \alpha_n)^T \mapsto \alpha_1 b_1 + \ldots + \alpha_n b_n$ egy invertálható lineáris leképezés $\mathbb{R}^n$-ből $\mathbb{R}^n$-be. Következésképpen ϕ és ϕ^{-1} is folytonos. Ezért ϕ diszkrét halmazt diszkrét halmazba képez. Mivel $L = \phi(\mathbb{Z}^n)$, elég belátni, hogy $\mathbb{Z}^n$ diszkrét $\mathbb{R}^n$-ben. Ez pedig nyilvánvaló: ha K egy korlátos halmaz $\mathbb{R}^n$-ben, akkor létezik olyan ρ pozitív szám, hogy K minden elemének minden koordinátája legfeljebb ρ abszolút értékű. Ezért $\mathbb{Z}^n$-nek legfeljebb $(2\lfloor\rho\rfloor + 1)^n$ eleme eshet K-ba.

A második állítást n szerinti indukcióval igazoljuk. Ha $L = \{0\}$, akkor nincs mit bizonyítani. Egyébként a diszkrétség miatt létezik L-ben egy minimális hosszúságú origótól különböző vektor, legyen ez b_1. Belátjuk, hogy L-nek a $\{\lambda b_1 | \lambda \in \mathbb{R}\}$ egyenesre eső pontjai mind a b_1 vektor egész együtthatós többszörösei. Tegyük fel ugyanis, hogy $\lambda b_1 \in L$ valamely nem egész λ számra. A szokásos módon jelölje $\{\lambda\}$ a λ törtrészét. Ekkor $0 \neq |\{\lambda\}b_1| < |b_1|$, ugyanakkor $\{\lambda\}b_1 = \lambda b_1 - [\lambda]b_1$, azaz két L-beli vektor különbsége, tehát maga is L-beli. Ez pedig ellentmond annak, hogy b_1-et legrövidebbnek választottuk.

A most igazolt állítás egyben bizonyítja az $n = 1$ esetet. Tegyük fel ezután, hogy $n > 1$. Írjuk fel $\mathbb{R}^n$ elemeit b_1-gyel párhuzamos és b_1-re merőleges vektorok összegeként:

$$v = v^* + \frac{(v, b_1)}{(b_1, b_1)} b_1 \ .$$

Egyszerű számolás mutatja, hogy $(v^*, b_1) = 0$, és a $v \mapsto v^*$ leképezés lineáris. Legyen $L^* = \{v^* | v \in L\}$. Belátjuk, hogy L^* is rács a b_1-re merőleges vektorok által alkotott $H \cong \mathbb{R}^{n-1}$ altérben (ún. hipersíkban). A $v \mapsto v^*$ leképezés lineáris, így L^* nyilván zárt az összeadásra és kivonásra. A diszkrétség igazolásához legyen K egy korlátos tartomány H-ban. Meg kell mutatnunk, hogy K-ba L^*-nak csak véges sok pontja esik. Legyen $v \in L$ egy olyan vektor, amelyre $v^* \in K$. Legyen λ a $(v, b_1)/(b_1, b_1)$ számhoz legközelebb eső egész szám és legyen $v' = v - \lambda b_1$. Nyilvánvaló, hogy $v' \in L$ és $v'^* = v^*$. Teljesül továbbá az is, hogy $|(v', b_1)/(b_1, b_1)| = |(v - \lambda b_1, b_1)/(b_1, b_1)| \leq 1/2$, tehát a v' vektor a szintén korlátos $K \times \{\mu b_1 : \ -1/2 \leq \mu \leq 1/2\}$, tartományban van. Ide csak véges sok $v' \in L$ vektor esik, ennélfogva K-ba is csak véges sok a $v^* = v'^* \in L^*$ eshet.

Beláttuk tehát, hogy L^* egy rács H-ban, következésképpen az indukciós feltevés miatt van bázisa (esetleg üres, ha L rangja 1). Legyenek $b_2, \ldots, b_r \in L$ olyan rácsvektorok, amelyekre $b_2^*, \ldots, b_r^*$ az L^* rács egy bázisát alkotják. Ekkor tetszőleges $v \in L$ rácsvektorra v^* előáll $\sum_{i=2}^r \lambda_i b_i^*$ alakban, ahol a λ_i együtthatók egész számok. Ekkor $v' = v - \sum_{i=2}^r \lambda_i b_i \in L$ és a $v \mapsto v^*$ leképezés linearitása miatt $v'^* = 0$. Ez pedig azt jelenti, hogy v' a λb_1 egyenes egy rácsvektora, következésképpen $v' = \lambda_1 b_1$ valamely λ_1 egész számra, és így $v = \sum_{i=1}^r \lambda_i b_i$, vagyis a $b_1, \ldots, b_r$ vektorok egy egész együtthatós lineáris kombinációja. A $b_1, \ldots, b_r$ rendszer tehát bázisa L-nek. ■

Egy L rács mindig teljes rács az általa kifeszített altérben. Ezért nem jelent lényeges megszorítást, ha csak teljes rácsokkal foglalkozunk. A továbbiakban rács alatt mindig *teljes rácsot* értünk.

18.4. példa. Két ismerős rács $\mathbb{R}^2$-ben:
1. A ***négyzetrács,*** aminek $b_1 = (1, 0)$, $b_2 = (0, 1)$ egy bázisa.
2. A ***háromszögrács,*** aminek $b_1 = (1, 0)$, $b_2 = (1/2, (\sqrt{3})/2)$ egy bázisa.

A következő egyszerű tényt többször fogjuk használni.

18.49. lemma. *Legyen L egy rács $\mathbb{R}^n$-ben, $b_1, \ldots, b_n$ pedig L egy bázisa. Ha a $b_1, \ldots, b_n$ rendszerben felcseréljük a bázisvektorok sorrendjét, vagy ha az egyik bázisvektorhoz a többinek egy egész együtthatós lineáris kombinációját hozzáadjuk, a kapott rendszer szintén bázisa L-nek.*

Bizonyítás. Nyilvánvaló. ■

Legyen $b_1, \ldots, b_n$ az L rács egy bázisa. A $b_1, \ldots, b_n$ bázis Gram-mátrixa, az a $B = (B_{ij})$ mátrix, amelynek elemei $B_{ij} = (b_i, b_j)$. A B mátrix pozitív definit, ugyanis $A^T A$ alakú, ahol A teljes rangú mátrix (lásd 31.6. tétel). Ebből következően a determinánsa egy pozitív valós szám.

18.50. lemma. *Legyen $b_1, \ldots, b_n$, illetve $w_1, \ldots, w_n$ két bázisa ugyanannak az L rácsnak. Legyenek B, illetve W a $B_{ij} = (b_i, b_j)$, illetve $W_{ij} = (w_i, w_j)$ elemekből álló mátrixok. Ekkor B és W determinánsa megegyezik.*

Bizonyítás. Minden $i = 1, \ldots, n$ indexre $w_i = \sum_{j=1}^{n} \alpha_{ij} b_j$ alakban írható fel, ahol az α_{ij} együtthatók egész számok. Legyen A az $A_{ij} = \alpha_{ij}$ elemekből álló mátrix. Ekkor

$$(w_i, w_j) \quad = \quad (\sum_{k=1}^{n} \alpha_{ik} b_k, \sum_{l=1}^{n} \alpha_{jl} b_l) = \sum_{k=1}^{n} \alpha_{ik} \sum_{l=1}^{n} (b_k, b_l) \alpha_{jl}$$

miatt $W = ABA^T$, így $\det W = \det B (\det A)^2$, tehát $\det W / \det B = (\det A)^2$ egy nemnegatív egész szám, mivel A egy egész elemekből álló mátrix. Ugyanez a gondolatmenet (a bázisok megcserélésével alkalmazva) mutatja, hogy $\det B / \det W$ is egy nemnegatív egész szám. Ez a két feltétel együtt csak úgy teljesülhet, hogy $\det B = \det W$. ■

18.51. definíció (rács determinánsa). *Az L rács determinánsa* $\det L = \sqrt{\det B}$, *ahol B az L egy $b_1, \ldots, b_n$ bázisának a Gram-mátrixa.*

Az előző lemma értelmében $\det L$ független a $b_1, \ldots, b_n$ bázis választásától. A $\det L$ mennyiség geometriailag is interpretálható: $\det L$ a $b_1, \ldots, b_n$ vektorok által meghatározott $\{\sum_{i=1}^{n} \alpha_i b_i : \ 0 \le \alpha_1, \ldots, \alpha_n \le 1\}$ test (ún. paralelepipedon) térfogata.

18.52. megjegyzés. *Tegyük fel, hogy a b_i vektorok koordinátái $\mathbb{R}^n$ egy ortonormált bázisában felírva $\alpha_{i1}, \ldots, \alpha_{in}$ ($i = 1, \ldots, n$). Ekkor a $b_1, \ldots, b_n$ rendszer B Gram-mátrixa $B = AA^T$, ahol A az $A_{ij} = \alpha_{ij}$ elemekből álló mátrix. Következésképpen, ha $b_1, \ldots, b_n$ az L rács egy bázisa, akkor* $\det L = |\det A|$.

Bizonyítás. A $(b_i, b_j) = \sum_{k=1}^{n} \alpha_{ik}\alpha_{jk}$ egyenlőségek alapján az állítás nyilvánvaló. ■

18.4.2. Rövid rácsvektorok

Szükségünk lesz a konvex geometria egy alapvető eredményére. Ennek előkészítéséhez bevezetünk néhány egyszerű fogalmat. Legyen $H \subseteq \mathbb{R}^n$. A H halmaz ***centrálisan szimmetrikus*** az origóra nézve, ha $v \in H$ esetén $-v \in H$ is igaz. A H ***konvex,*** ha $u, v \in H$ esetén $\lambda u + (1-\lambda)v \in H$ is igaz, minden $0 \le \lambda \le 1$ valós számmal.

18.53. tétel (Minkowski konvex test tétele). *Legyen L egy rács $\mathbb{R}^n$-ben és legyen K az origóra centrálisan szimmetrikus, korlátos, zárt és konvex halmaz. Tegyük fel, hogy K térfogata legalább $2^n \det L$. Ekkor $K \cap L \neq \{0\}$.*

Bizonyítás. A feltétel szerint az $(1/2)K := \{(1/2)v : \ v \in K\}$ alakzat térfogata legalább $\det L$. Legyen $b_1, \ldots, b_n$ az L rács egy bázisa és legyen $P = \{\sum_{i=1}^{n} \alpha_i b_i : \ 0 \le \alpha_1, \ldots, \alpha_n < 1\}$ a megfelelő félig nyílt paralelepipedon. Ekkor $\mathbb{R}^n$ minden vektora egyértelműen írható fel $x + z$ alakban, ahol $x \in L$ és $z \in P$. Egy tetszőleges $x \in L$ rácsvektorra legyen

$$K_x = (1/2)K \cap (x + P) = (1/2)K \cap \{x + z : \ z \in P\} \ .$$

Mivel $(1/2)K$ és P korlátosak, az

$$(1/2)K - P = \{u - v : \ u \in (1/2)\cdot K, \ v \in P\}$$

halmaz is az, tehát ez utóbbiba L diszkrét volta miatt L-nek csak véges sok pontja esik. Másképpen fogalmazva, $K_x = \emptyset$ véges sok $x \in L$ kivételével. Ezek szerint az $S = \{x \in L : \ K_x \neq \emptyset\}$ egy véges halmaz, továbbá $(1/2)K$ a K_x $(x \in S)$ halmazok diszjunkt egyesítése. Ezért ezen halmazok össztérfogata legalább $\det L$. Adott $x \in S$-re legyen $P_x = K_x - x = \{z \in P : \ x + z \in (1/2)K\}$. Tekintsük P, illetve a P_x halmazok lezártját:

$$\overline{P} = \left\{ \sum_{i=1}^{n} \alpha_i b_i : \ 0 \le \alpha_1, \ldots, \alpha_n \le 1 \right\}$$

és $\overline{P_x} = \left\{ z \in \overline{P} : \ x + z \in (1/2)K \right\}$. A $\overline{P_x} \subseteq \overline{P}$ zárt halmazok össztérfogata legalább akkora, mint $\overline{P}$ térfogata, ezért nem lehetnek diszjunktak: létezik $x \neq y \in S$ és $z \in \overline{P}$ amelyekre $z \in \overline{P_x} \cap \overline{P_y}$, vagyis $x+z \in (1/2)K$ és $y+z \in (1/2)K$. Mivel $(1/2)\cdot K$ az origóra szimmetrikus, $-y-z \in (1/2)\cdot K$ is teljesül. $(1/2)K$ konvexitása miatt pedig $(x-y)/2 = ((x+z)+(-y-z))/2 \in (1/2)K$ is igaz. Ezért $x - y \in K$. Ugyanakkor, lévén két különböző rácsvektor különbsége, $x - y \in L \setminus \{0\}$ is teljesül. ■

Minkowski tétele éles. Legyen ugyanis $\epsilon > 0$ tetszőlegesen kicsiny pozitív szám, $L = \mathbb{Z}^n$ az egész koordinátájú pontokból álló rács $\mathbb{R}^n$-ben. Legyen továbbá K azon $(v_1, \ldots, v_n) \in \mathbb{R}^n$ vektorok halmaza, amelyekre $-1 + \epsilon \le v_i \le 1 - \epsilon$ teljesül $(i = 1, \ldots, n)$. Ekkor K korlátos, zárt, konvex, az origóra szimmetrikus, térfogata $(1-\epsilon)^n 2^n \det L$, ugyanakkor $L \cap K = \{0\}$.

18.54. következmény. *Legyen L egy rács $\mathbb{R}^n$-ben. Ekkor van L-nek olyan $v \neq 0$ rácsvektora, amelynek a hossza legfeljebb $\sqrt{n}\sqrt[n]{\det L}$.*

Bizonyítás. Legyen K a következő origó-középpontú, $s = 2\sqrt[n]{\det L}$ élhosszúságú kocka:

$$K = \{(v_1, \ldots, v_n) \in \mathbb{R}^n : \ -s/2 \le v_i \le s/2, \ i = 1, \ldots, n\} \ .$$

A K kocka térfogata éppen $2^n \det L$, tehát tartalmaz nem-nulla rácsvektort. Ugyanakkor a K-beli vektorok hossza legfeljebb $\sqrt{n}\sqrt[n]{\det L}$. ■

Megjegyezzük, hogy az $n > 1$ esetben rövidebb rácsvektor is van. A bizonyításhoz kocka helyett kellően nagy térfogatú gömböt célszerű választani.

18.4.3. Gauss algoritmusa a kétdimenziós esetre

Olyan algoritmust szeretnénk, amely egy rácsban rövid (nem nulla) vektort talál. Itt a legegyszerűbb nemtriviális esettel, a síkbeli rácsok esetével foglalkozunk. Ekkor elegáns, szemléletes és hatékony algoritmussal találhatunk legrövidebb rácsvektort. A megoldás a magasabb dimenziós módszerek alapjául is szolgál. Legyen L a b_1, b_2 bázisával adott rács $\mathbb{R}^2$-ben.

GAUSS(b_1, b_2)

```
1  (a, b) ← (b_1, b_2)
2  forever
3        do b ← a b − λa egyenesen fekvő legrövidebb rácsvektor
4        if |b| < |a|
5          then b ↔ a
6          else return (a, b)
```

Az eljárás elemzéséhez hasznosak lesznek a következő tények:

18.55. lemma. *Tegyük fel, hogy a és b két lineárisan független vektor az $\mathbb{R}^2$ síkban, és legyen L az általuk meghatározott rács. A b akkor és csak akkor az egyik legrövidebb L-beli vektor a $b - \lambda a$ egyenesen, ha*

$$|(b, a)/(a, a)| \le 1/2 \ . \tag{18.10}$$

Bizonyítás. Írjuk fel b-t a-val párhuzamos és a-ra merőleges vektorok összegeként:

$$b = (b, a)/(a, a)a + b^* \ . \tag{18.11}$$

Ekkor az a és b^* merőlegessége miatt

$$|b - \lambda a|^2 = \left|\left(\frac{(b, a)}{(a, a)} - \lambda\right)a + b^*\right|^2 = \left(\frac{(b, a)}{(a, a)} - \lambda\right)^2 |a|^2 + |b^*|^2 \ .$$

Ez a mennyiség arra a λ egész számra a legkisebb, amely a lehető legközelebb áll a $(b, a)/(a, a)$ számhoz. Eszerint $\lambda = 0$ pontosan akkor adja a minimális értéket, ha (18.10) teljesül. ■

18.56. lemma. *Tegyük fel, hogy a lineárisan független a és b vektorok az $L \subseteq \mathbb{R}^2$ rács bázisát alkotják, és teljesül rájuk a (18.10) egyenlőtlenség. Tegyük továbbá fel, hogy*

$$|b|^2 \geq (3/4)|a|^2 \ . \tag{18.12}$$

Írjuk fel b-t a (18.11) formula szerint az a vektorral párhuzamos $((b,a)/(a,a))a$ és az a-ra merőleges $b^* = b - ((b,a)/(a,a))a$ vektorok összegeként. Ekkor

$$|b^*|^2 \geq (1/2)|a|^2 \ , \tag{18.13}$$

továbbá L-ben vagy b, vagy pedig a az egyik legrövidebb nem-nulla vektor.

Bizonyítás. A feltevések miatt

$$|a|^2 \leq (4/3)|b|^2 = (4/3)|b^*|^2 + (4/3)\left((b,a)/(a,a)\right)^2 |a|^2 \leq (4/3)|b^*|^2 + (1/3)|a|^2 \ .$$

Innen átrendezéssel adódik, hogy $|b^*|^2 \geq (1/2)|a|^2$.

Egy $0 \neq v = \alpha a + \beta b \in L$ vektor hosszára

$$|\alpha a + \beta b|^2 = |\beta b^*|^2 + (\alpha + \beta(b,a)/(a,a))^2 |a|^2 \geq \beta^2 |b^*|^2 \geq (1/2)\beta^2 |a|^2$$

teljesül, amiből $|\beta| \geq 2$ esetén $|v| > |a|$. Ha $\beta = 0$ és $\alpha \neq 0$, akkor $|v| = |\alpha| \cdot |a| \geq |a|$. Hasonlóan, $\alpha = 0$ és $\beta \neq 0$ esetén $|v| = |\beta| \cdot |b| \geq |b|$. Marad tehát az eset, amikor $\alpha \neq 0$ és $\beta = \pm 1$. Mivel $|-v| = |v|$, feltehető, hogy $\beta = 1$. De ekkor $v = b - \lambda a$ alakú ($\lambda = -\alpha$,) és a 18.55. lemmából tudjuk, hogy ezen az egyenesen b az egyik legrövidebb rácsvektor. ■

18.57. tétel. *Legyen v az L rács (egyik) legrövidebb 0-tól különböző vektora. Ekkor a* Gauss*-algoritmus $O(1 + \lg(|b_1|/|v|))$ iterációs menetben véget ér, és az eredményül kapott a vektor az L rács (egyik) legrövidebb vektora.*

Bizonyítás. Először belátjuk, hogy az a és a b vektorok az algoritmus során L-nek mindig egy bázisát alkotják. Ha a 2. sorban kicseréljük b-t valamely $b' = b - \lambda a$ alakú vektorral, akkor $b = b' + \lambda a$ miatt az a, b' pár is az L egy bázisa. A 4. sorban történő esetleges csere pedig csak a bázisvektorok sorrendjét érinti. Tehát a és b valóban mindig az L egy bázisát alkotja.

A ciklus első lépése (2. sor) után a 18.55. lemma alapján teljesül a (18.10), ezért a második lépése (3–4. sor) előtti helyzetre alkalmazható a 18.56. lemma. Eszerint ha a és b egyike sem legrövidebb rácsvektor, akkor $|b|^2 \leq (3/4)|a|^2$. Tehát – kivéve esetleg a legutolsó kört – a ciklus második lépésében végrehajtott csere az a vektor hosszát $\sqrt{3/4}$-szeresére, vagy még rövidebbre zsugorítja. Innen adódik korlát a menetek számára. A 18.56. lemmából következik az is, hogy végül az a vektor egy legrövidebb nem nulla vektor L-ben. ■

A Gauss algoritmus hatékony, polinom idejű módszert ad az $L \subseteq \mathbb{R}^2$ rács egy legrövidebb vektorának a kiszámítására. Az algoritmus elemzése egy érdekes elméleti következményt is szolgáltat:

18.58. következmény. *Legyen L rács $\mathbb{R}^2$-ben, a pedig az L egyik legrövidebb nem nulla vektora. Ekkor $|a|^2 \leq (2/\sqrt{3}) \det L$.*

Bizonyítás. Legyen b egy a-tól lineárisan független vektora L-nek, amelyre a (18.10) teljesül. Ekkor

$$|a|^2 \le |b|^2 = |b^*|^2 + \left(\frac{(b,a)}{(a,a)}\right)^2 |a|^2 \le |b^*|^2 + \frac{1}{4}|a|^2 ,$$

amiből $(3/4)|a|^2 \le |b^*|^2$ adódik. Az alapparalelogramma területére a jól ismert

$$\text{terület} = \text{alap} \cdot \text{magasság}$$

képletet használva kapjuk, hogy $\det L = |a||b^*|$. A $|b^*|$ itt alulról becsülhető az előző, a $|b^*|$-ra vonatkozó egyenlőtlenség alapján. ■

18.4.4. A Gram–Schmidt-ortogonalizáció és a gyenge redukció

Legyen $b_1, \dots, b_n$ egy n lineárisan független vektorból álló rendszer $\mathbb{R}^n$-ben. Egy $i \in \{1, \dots, n\}$ indexre jelölje b_i^* a b_i vektornak a $b_1, \dots, b_{i-1}$ vektorok által kifeszített altérre merőleges összetevőjét. Tehát

$$b_i = b_i^* + \sum_{j=1}^{i-1} \lambda_{ij} b_j ,$$

ahol

$$(b_i^*, b_j) = 0 \quad (j = 1, \dots, i-1) .$$

Értelemszerűen $b_1^* = b_1$. A $b_1^*, \dots, b_{i-1}^*$ vektorok ugyanazt az alteret feszítik ki, mint $b_1, \dots, b_{i-1}$, ezért alkalmas μ_{ij} együtthatókkal

$$b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{ij} b_j^* , \tag{18.14}$$

alakú, és itt

$$(b_i^*, b_j^*) = 0, \quad \text{ha } j \ne i .$$

Az utóbbi összefüggések szerint $b_1^*, \dots, b_{i-1}^*, b_i^*$ egy ortogonális rendszer, ezért

$$\mu_{ij} = \frac{(b_i, b_j^*)}{(b_j^*, b_j^*)} \quad (j = 1, \dots, i-1) . \tag{18.15}$$

A $b_1^*, \dots, b_n^*$ rendszert a $b_1, \dots, b_n$ ***Gram–Schmidt-ortogonalizáltjának*** szokás nevezni.

18.59. lemma. *Legyen az $L \subseteq \mathbb{R}^n$ rács egy bázisa $b_1, \dots, b_n$. Ekkor*

$$\det L = \prod_{i=1}^{n} |b_i^*| .$$

Bizonyítás. Legyen $\mu_{ii} = 1$ és $\mu_{ij} = 0$, ha $j > i$. Ekkor $b_i^* = \sum_{k=1}^n \mu_{ik} b_k$, így $(b_i^*, b_j^*) = \sum_{k=1}^n \mu_{ik} \sum_{l=1}^n (b_k, b_l) \mu_{jl}$, azaz $B^* = MBM^T$, ahol B a $b_1, \dots, b_n$ rendszer Gram-mátrixa, B^*

a $b_1^*, \ldots, b_n^*$ rendszeré, M pedig a μ_{ij} elemekből álló mátrix. Az M alsó háromszögmátrix, aminek a főátlójában minden érték 1, ezért $\det M = \det M^T = 1$. Így, mivel B^* diagonális mátrix, $\prod_{i=1}^n |b_i^*|^2 = \det B^* = \det M \det B \det M^T = \det B$. ■

18.60. következmény (Hadamard-egyenlőtlenség). $\prod_{i=1}^n |b_i| \geq \det L$.

Bizonyítás. A b_i előáll, mint a b_i^* és egy b_i^*-ra merőleges vektor összege, tehát $|b_i^*| \leq |b_i|$. ■

A b_i^* a b_i vektornak az $b_1, \ldots, b_{i-1}$ vektorok által kifeszített altérre merőleges összetevője. Ezért b_i^* nem változik, ha b_i-ből levonjuk a $b_1, \ldots, b_{i-1}$ vektorok egy lineáris kombinációját. Ha ebben a kombinációban az együtthatók egészek, akkor az új $b_1, \ldots, b_n$ rendszer ugyanannak a rácsnak lesz bázisa, mint az eredeti. A Gauss algoritmusban szereplő ciklus első lépéséhez hasonló módon elérhető, hogy a (18.15) képletben szereplő μ_{ij} számok kicsik legyenek. A következő eljárás bemenete az L rács $b_1, \ldots, b_n$ bázisa.

Gyenge-redukció$(b_1, \ldots, b_n)$

1 **for** $j \leftarrow n-1$ **downto** 1
2 **do for** $i \leftarrow j+1$ **to** n
3 $b_i \leftarrow b_i - \lambda b_j$, ahol λ a $(b_i, b_j^*)/(b_j^*, b_j^*)$ számhoz legközelebbi egész
4 **return** $(b_1, \ldots, b_n)$

18.61. definíció (Gyengén redukált bázis). *A $b_1, \ldots, b_n$ bázist **gyengén redukáltnak** nevezzük, ha a (18.15) formulában szereplő μ_{ij} számokra*

$$|\mu_{ij}| \leq \frac{1}{2}, \quad (1 \leq j < i \leq n) .$$

18.62. lemma. *A* Gyenge-redukció *eljárás eredményeként kapott bázis gyengén redukált.*

Bizonyítás. Az algoritmus előtt tett megjegyzés rámutat, hogy $b_1^*, \ldots, b_n^*$ sosem változik: b_i-ből csak nála kisebb indexű bázisvektorok kombinációit vonjuk le. Ezért a belső utasítás nem változtatja meg a (b_k, b_l^*) értékeket, ahol $k \neq i$. A (b_i, b_l^*) értékek sem változnak $l > j$ esetén. Ugyanakkor az utasítás eléri, hogy az új b_i-vel $|\mu_{ij}| \leq 1/2$ teljesüljön:

$$|(b_i - \lambda b_j^*, b_j^*)| = |(b_i, b_j^*) - \lambda(b_j^*, b_j^*)| = |(b_i, b_j^*) - \lambda(b_j^*, b_j^*)| \leq \frac{1}{2}(b_j^*, b_j^*) .$$

A fenti észrevételek alapján ez az egyenlőtlenség az eljárás futása során később is érvényben marad. ■

18.4.5. A Lovász-redukció

Először megadjuk – tetszőleges dimenzióban – a számunkra hasznos bázis fogalmát. A definíció elég technikai természetű. Később látni fogjuk, hogy ezek a bázisok tényleg érdekesek abban az értelemben, hogy viszonylag rövid vektorokból állnak. Ez utóbbi tulajdonságuk teszi őket széles körben alkalmazhatóvá.

18.63. definíció. *Az L rács $b_1, \ldots, b_n$ bázisát* ***(Lovász-)redukáltnak*** *nevezzük, ha*

- *gyengén redukált,*

továbbá a Gram–Schmidt-ortogonalizációnál bevezetett jelölésekkel

- $|b_i^*|^2 \le (4/3)|b_{i+1}^* + \mu_{i+1,i} b_i^*|^2$ *minden* $1 \le i < n$ *esetén.*

Figyeljük meg a párhuzamot a Gauss algoritmus kapcsán tárgyaltakkal! Az $i = 1$ esetben az $a = b_1$ és $b = b_2$ szereposztásban a gyengén redukáltság biztosítja, hogy b a $b - \lambda a$ egyenes legrövidebb rácsvektora, míg a második feltétel az elemzésnél használt $|b|^2 \ge (3/4)|a|^2$ feltétellel egyenértékű, csak itt a Gram–Schmidt-bázis segítségével fogalmaztuk meg. Általános i indexre ugyanez igaz abban a szereposztásban, hogy a a b_i vektornak, b pedig a b_{i+1} vektornak a $b_1, \ldots, b_{i-1}$ vektorok által feszített altérre merőleges összetevője.

Lovász-redukció$(b_1, \ldots, b_n)$

1 **forever**
2 **do** $(b_1, \ldots, b_n) \leftarrow$ Gyenge-redukció$(b_1, \ldots, b_n)$
3 keressünk egy olyan i indexet, amelyre a redukáltság második feltétele megsérül
4 **if** van ilyen
5 **then** $b_i \leftrightarrow b_{i+1}$
6 **else return** $(b_1, \ldots, b_n)$

18.64. tétel. *Tegyük fel, hogy az $L \subseteq \mathbb{R}^n$ rácsban bármely két vektor skaláris szorzata egész. Ekkor a* Lovász-redukció *eljárása során az 5. sorban leírt csere legfeljebb* $\log_{4/3}(B_1 \cdots B_{n-1})$*-szer kerül végrehajtásra, ahol B_i a $b_1, \ldots, b_n$ kiindulási bázis Gram-mátrixának a bal felső $(i \times i)$-es aldeterminánsa.*

Bizonyítás. A B_i determináns a $b_1, \ldots, b_i$ rendszer Gram-mátrixa, így a Gram–Schmidt-ortogonalizációnál megállapítottak miatt $B_i = \prod_{j=1}^{i} |b_j^*|^2$. Ebből persze az is következik, hogy $i > 1$ esetén $B_i = B_{i-1}|b_i^*|^2$. A korábban mondottak szerint a gyenge redukció nem változtatja meg a b_i^* vektorokat, ezért a $\prod_{j=1}^{n-1} B_j$ szorzatot sem. Tegyük fel, hogy az eljárás 3. pontjában egy $b_i \leftrightarrow b_{i+1}$ csere történik. Vegyük észre, hogy $j = i$ kivételével a $\{b_1, \ldots, b_j\}$ halmazok nem változnak, így a B_j determinánsok sem. A b_i^* vektor szerepét $b_{i+1}^* + \mu_{i,i+1} b_i$ veszi át. Ennek hossza a csere feltétele miatt az eredeti b_i^* hosszának legfeljebb $\sqrt{3/4}$-szerese, vagyis az új B_i a réginek legfeljebb 3/4 része. A fenti észrevétel alapján tehát a $B = \prod_{j=1}^{n-1} B_j$ mennyiség is (3/4)-szeresére, vagy még rövidebbre zsugorodik. Az állítás ezután következik abból, hogy a B végig pozitív egész marad. ■

18.65. következmény. *A tétel feltevéseivel élve, a Lovász-redukció költsége $O(n^5 \lg nC)$ aritmetikai művelet racionális számokkal, ahol C a 2 és a $|(b_i, b_j)|$ mennyiségek $(i, j = 1, \ldots, n)$ maximuma.*

Bizonyítás. Az Hadamard-egyenlőtlenség miatt

$$B_i \le \prod_{j=1}^{i} \sqrt{(b_1, b_j)^2 + \ldots + (b_i, b_j)^2} \le (\sqrt{i}C)^i \le (\sqrt{n}C)^n \ .$$

Így $B_1 \cdots B_{n-1} \leq (\sqrt{n}C)^{n(n-1)}$ és $\log_{4/3}(B_1 \dots B_{n-1}) = O(n^2 \lg nC)$. A tétel alapján ennyi a menetek száma. A Gram–Schmidt-ortogonalizáció költsége $O(n^3)$ művelet, a gyenge redukció költsége $O(n^2)$ skaláris szorzás, amelyek mindegyike elvégezhető $O(n)$ művelettel (ha a vektorokat egy ortogonális bázis szerinti koordinátákkal ábrázoljuk). ■

Megmutatható az is, hogy az algoritmus futása során fellépő egész számok (beleértve a Gram–Schmidt-ortogonalizációnál keletkező törtek számlálóit és nevezőit is) hossza polinomiális korlát alatt marad.

18.4.6. A redukált bázisok tulajdonságai

A 18.67. tételben összefoglaljuk a redukált bázisoknak azokat a tulajdonságait, amelyek az alkalmazásoknál hasznosak. Kiderül, hogy egy redukált bázis viszonylag rövid vektorokból áll. Nevezetesen a $|b_1|$ csak a dimenziótól függő (és ezen túl L-től független) szorzó erejéig megközelíti a rács legrövidebb nem nulla vektorának a hosszát.

18.66. lemma. *Tegyük fel, hogy a $b_1, \dots, b_n$ rendszer redukált bázisa az L rácsnak. Ekkor $1 \leq j \leq i \leq n$ esetén*

$$(b_i^*, b_i^*) \geq 2^{j-i}(b_j^*, b_j^*) \,. \tag{18.16}$$

Speciálisan

$$(b_i^*, b_i^*) \geq 2^{1-i}(b_1^*, b_1^*) \,. \tag{18.17}$$

Bizonyítás. A 18.56. lemma az $a = b_i^*$, $b = b_{i+1}^* + ((b_{i+1}, b_i^*))/((b_i^*, b_i^*)b_i^*)$ szereposztással adja, hogy minden $1 \leq i < n$-re

$$(b_{i+1}^*, b_{i+1}^*) \geq (1/2)(b_i^*, b_i^*) \,.$$

Innen a (18.16) egyenlőtlenség indukcióval adódik. ■

Ezután kimondhatjuk a redukált bázisokra vonatkozó alaptételt.

18.67. tétel. *Tegyük fel, hogy a $b_1, \dots, b_n$ rendszer redukált bázisa az L rácsnak. Ekkor:*

1. $|b_1| \leq 2^{(n-1)/4}(\det L)^{(1/n)}$.
2. $|b_1| \leq 2^{(n-1)/2}|b|$ *minden* $0 \neq b \in L$ *rácsvektorra. Azaz b_1 egy $2^{(n-1)/2}$-szeres közelítő legrövidebb rácsvektor.*
3. $|b_1| \cdots |b_n| \leq 2^{(n(n-1))/4} \det L$.

Bizonyítás. 1. A (18.17) egyenlőtlenséget használva

$$(\det L)^2 = \prod_{i=1}^{n}(b_i^*, b_i^*) \geq \prod_{i=1}^{n}(2^{1-i}(b_1, b_1)) = 2^{\frac{-n(n-1)}{2}}(b_1, b_1)^n \,,$$

tehát 1. teljesül.

2. Legyen $b = \sum_{i=1}^{n} z_i b_i \in L$, ahol $z_i \in \mathbb{Z}$. Tegyük fel, hogy z_j az utolsó nem 0 együttható. Legyen továbbá $b_j = b_j^* + v$, ahol v a $b_1, \ldots, b_{j-1}$ vektorok egy lineáris kombinációja. Így $b = z_j b_j^* + w$, ahol w a $b_1, \ldots, b_{j-1}$ vektorok által feszített altérbe esik. Mivel b_j^* merőleges erre az altérre,

$$(b,b) = z_j^2 (b_j^*, b_j^*) + (w,w) \geq (b_j^*, b_j^*) \geq 2^{1-j}(b_1,b_1) \geq 2^{1-n}(b_1,b_1)\ ,$$

ezért 2. teljesül.

3. Először megmutatjuk, hogy $(b_i,b_i) \leq 2^{i-1}(b_i^*, b_i^*)$. Ez nyilvánvaló az $i = 1$ esetben. Legyen ezután $i > 1$. A b_i vektor (18.14) felírását használva, és tekintetbe véve, hogy a bázis gyengén redukált,

$$\begin{aligned}(b_i,b_i) &= \sum_{j=1}^{i} \left(\frac{(b_i, b_j^*)}{(b_j^*, b_j^*)} \right)^2 (b_j^*, b_j^*) \leq (b_i^*, b_i^*) + \frac{1}{4} \sum_{j=1}^{i-1} (b_j^*, b_j^*) \leq (b_i^*, b_i^*) + \frac{1}{4} \sum_{j=1}^{i-1} 2^{i-j} (b_i^*, b_i^*) \\ &\leq (2^{i-2} + 1)(b_i^*, b_i^*) \leq 2^{i-1}(b_i^*, b_i^*)\ .\end{aligned}$$

A kapott egyenlőtlenségeket összeszorozva:

$$\prod_{i=1}^{n} (b_i,b_i) \leq \prod_{i=1}^{n} 2^{i-1}(b_i^*, b_i^*) = 2^{\frac{n(n-1)}{2}} \prod_{i=1}^{n} (b_i^*, b_i^*) = 2^{\frac{n(n-1)}{2}} (\det L)^2\ ,$$

ami éppen a 3. egyenlőtlenség. ■

A tétel 1. állítását érdemes összevetni a Minkowski-tétel 18.54. következményével. A b_1 hosszára itt gyengébb korlát adódik, ám ekkora vektort hatékony algoritmussal kaphatunk. A 3. állításbeli bázis létezését először Hermite mutatta meg lényegében a 18.48. és a 18.67. tételek bizonyításában bemutatott eszközök felhasználásával. Megjegyezzük még, hogy egy Lovász-redukált bázis segítségével egy n dimenziós rácsban a bemeneti adatok méretében és 3^{n^2}-ben polinomiális időben megkereshető a legrövidebb rácsvektor, lásd a 18.4-4. gyakorlatot.

Gyakorlatok

18.4-1. *A háromszögrács optimalitása.* Mutassuk meg, hogy a 18.58. következmény korlátja éles. Pontosabban: legyen $L \subseteq \mathbb{R}^2$ teljes rács, $0 \neq a \in L$ az L egy legrövidebb vektora. Az $|a|^2 = (2/\sqrt{3}) \det L$ egyenlőség akkor és csak akkor áll fenn, ha L hasonló a háromszögrácshoz.

18.4-2. *A Gram–Schmidt-számok nevezői.* Tegyük fel, hogy a $b_1, \ldots, b_n$ bázis Grammátrixa egész elemű. Mutassuk meg, hogy ekkor a (18.15) képletben szereplő μ_{ij} számok $\mu_{ij} = \zeta_{ij} / \prod_{k=1}^{j-1} B_k$ alakban írhatók, ahol ζ_{ij} egész és B_k a $b_1, \ldots, b_k$ rendszer Grammátrixának a determinánsa.

18.4-3. *Redukált bázis vektorainak hossza.* Legyen $b_1, \ldots, b_n$ redukált bázisa L-nek és tegyük fel, hogy a (b_i, b_i) számok egészek. Adjunk csak n-től és $\det L$-től függő felső becslést a b_i vektorok hosszára: mutassuk meg, hogy

$$|b_i| \leq 2^{\frac{n(n-1)}{4}} \det L\ .$$

18.4-4. *A legrövidebb rácsvektor koordinátái.* Legyen $b_1, \ldots, b_n$ redukált bázisa L-nek. Bizonyítsuk be, hogy L-nek (minden) legrövidebb vektora $\sum z_i b_i$ alakba írható, ahol $z_i \in \mathbb{Z}$ és $|z_i| \leq 3^n$. Következésképpen rögzített (kis) n-re polinom időben található legrövidebb nem 0 rácsvektor.

Útmutatás. Tegyük fel, hogy valamely $v = \sum z_i b_i$ rácsvektorra $|v| \leq |b_1|$. Írjuk fel v-t a $b_1^*, \ldots, b_n^*$ bázisban:

$$v = \sum_{j=1}^{n} (z_j + \sum_{i=j+1}^{n} \mu_{ij} z_i) b_j^* \,.$$

A feltevésből következik, hogy v minden (az ortogonalizált bázisban felírt) komponense legfeljebb olyan hosszú, mint $b_1 = b_1^*$:

$$\left| z_j + \sum_{i=j+1}^{n} \mu_{ij} z_i \right| \leq \frac{|b_1^*|}{|b_j^*|} \,.$$

Használjuk ezután a $|\mu_{ij}| \leq 1/2$ és a (18.17) egyenlőtlenségeket.

18.5. Polinomok felbontása $\mathbb{Q}[x]$-ben

Ebben az alfejezetben racionális együtthatós polinomok felbontásával foglalkozunk. A ***felbontási feladat*** bemenete egy $f(x) \in \mathbb{Q}[x]$ polinom. A célunk az f polinom

$$f = f_1^{e_1} f_2^{e_2} \cdots f_s^{e_s} \tag{18.18}$$

felbontásának kiszámítása, ahol $f_1, \ldots, f_s$ páronként relatív prím, $\mathbb{Q}$ felett irreducibilis polinomok, az e_i számok pedig pozitív egészek. A 18.4. tétel szerint f lényegében egyértelműen meghatározza az f_i polinomokat és az e_i kitevőket.

18.5.1. Előkészületek

Legelőször is visszavezetjük a (18.18) feladatot egy kényelmesebben kezelhető speciális esetére.

18.68. lemma. *Feltehető, hogy a felbontandó $f(x)$ egész együtthatós, 1-főegyütthatós polinom.*

Bizonyítás. Az együtthatók egy közös nevezőjével felszorozva elérhetjük, hogy $f(x) = a_0 + a_1 x + \cdots + a_n x^n \in \mathbb{Z}[x]$ teljesüljön. Ezután végezzük el az $y = a_n x$ helyettesítést. A

$$g(y) = {a_n}^{n-1} f\left(\frac{y}{a_n}\right) = y^n + \sum_{i=0}^{n-1} a_n^{n-i-1} a_i y^i$$

polinom nyilvánvalóan egész együtthatós, 1-főegyütthatós. A $g(y)$ felbontásából $f(x)$ felbontása hatékonyan megkapható. ■

Primitív polinomok, Gauss-lemma

18.69. definíció. *Az $f(x) \in \mathbb{Z}[x]$ polinomot **primitívnek** nevezzük, ha együtthatóinak a legnagyobb közös osztója 1.*

Bármely $f(x) \in \mathbb{Z}[x]\backslash\{0\}$ polinom egyértelműen írható fel egy pozitív egész szám és egy $\mathbb{Z}[x]$-beli primitív polinom szorzataként: legyen a az együtthatók legnagyobb közös osztója, és ekkor $f(x) = a(1/a)f(x)$. Nyilvánvaló, hogy $(1/a)f(x)$ egy primitív egész együtthatós polinom.

18.70. lemma (Gauss-lemma). *Legyenek $u(x), v(x) \in \mathbb{Z}[x]$ primitív polinomok. Ekkor az $u(x)v(x)$ szorzat is primitív.*

Bizonyítás. Tegyük fel indirekt módon, hogy p egy olyan prímszám, amely osztja uv összes együtthatóját. Legyen $u(x) = \sum_{i=0}^{n} u_i x^i$, $v(x) = \sum_{j=0}^{m} v_j x^j$ és legyen i_0, illetve j_0 az a legkisebb olyan index, amelyre $p \nmid u_{i_0}$, illetve $p \nmid v_{j_0}$. Legyen $k_0 = i_0 + j_0$ és tekintsük az x^{k_0} együtthatóját az $u(x)v(x)$ szorzatban. Ez az együttható

$$\sum_{i+j=k_0} u_i v_j = u_{i_0} v_{j_0} + \sum_{i=0}^{i_0-1} u_i v_{k_0-i} + \sum_{j=0}^{j_0-1} u_{k_0-j} v_j \ .$$

Az utóbbi két összeg osztható p-vel, míg $u_{i_0} v_{j_0}$ nem, tehát az $u(x)v(x)$ vizsgált együtthatója mégsem osztható p-vel. Ellentmondás. ■

18.71. állítás. *Tegyük fel, hogy $g(x), h(x) \in \mathbb{Q}[x]$ racionális együtthatós, 1-főegyütthatós polinomok, amelyekre a $g(x)h(x)$ szorzat egész együtthatós. Ekkor $g(x)$ és $h(x)$ is egész együtthatós polinomok.*

Bizonyítás. Szorozzuk meg $g(x)$-et, illetve $h(x)$-et a nevezőinek c_g, illetve c_h legkisebb közös többszörösével. Ekkor a $c_g g(x)$ és $c_h h(x)$ polinomok primitív egész együtthatós polinomok, így a Gauss-lemma miatt a $c_g c_h g(x)h(x) = (c_g g(x))(c_h h(x))$ szorzat is az. Ezen polinom minden együtthatója osztható a $c_g c_h$ számmal, hiszen $g(x)h(x)$ egész együtthatós. Tehát $c_g c_h = 1$, és így $c_g = c_h = 1$, ezért $g(x)$ és $h(x)$ tényleg egész együtthatós polinomok. ■

Hasonlóan igazolható, hogy egy $f(x) \in \mathbb{Z}[x]$ polinom $\mathbb{Z}[x]$-ben felbonthatatlan tényezőkre való felbontása egyenértékű az $f(x)$ primitív részének a $\mathbb{Q}[x]$-beli felbontásával és egy egész szám (az együtthatók legnagyobb közös osztója) prímtényezős felbontásával.

A Mignotte-korlát

Végtelen test felett dolgozunk, ezért több figyelmet kell fordítanunk a számítások eredményeinek a méretére. Ennek egyik eszközét vezetjük itt be.

18.72. definíció. *Egy $f(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{C}[x]$ komplex együtthatós polinom **normája** a $\|f(x)\| = \sqrt{\sum_{i=0}^{n} |a_i|^2}$ nemnegatív valós szám.*

A $\max_{i=0}^{n}|a_i| \leq \|f(x)\|$ egyenlőtlenség alapján, ha $f(x)$ egész együtthatós, akkor $O(n \lg \|f(x)\|)$ bittel ábrázolható.

18.73. lemma. *Legyen $f(x) \in \mathbb{C}[x]$ egy komplex együtthatós polinom. Ekkor tetszőleges $c \in \mathbb{C}$ komplex számra*

$$\|(x-c)f(x)\| = \|(\overline{c}x-1)f(x)\| ,$$

ahol $\overline{c}$ a c szám szokásos komplex konjugáltja.

Bizonyítás. Tegyük fel, hogy $f(x) = \sum_{i=0}^{n} a_i x^i$ és legyen $a_{n+1} = a_{-1} = 0$. Ekkor

$$(x-c)f(x) = \sum_{i=0}^{n+1}(a_{i-1} - ca_i)x^i ,$$

és így

$$\begin{aligned}\|(x-c)f(x)\|^2 &= \sum_{i=0}^{n+1}|a_{i-1}-ca_i|^2 = \sum_{i=0}^{n+1}(|a_{i-1}|^2 + |ca_i|^2 - a_{i-1}\overline{ca_i} - \overline{a_{i-1}}ca_i) \\ &= \|f(x)\|^2 + |c|^2\|f(x)\|^2 - \sum_{i=0}^{n+1}(a_{i-1}\overline{ca_i} + \overline{a_{i-1}}ca_i) .\end{aligned}$$

Hasonlóképpen,

$$(\overline{c}x-1)f(x) = \sum_{i=0}^{n+1}(\overline{c}a_{i-1} - a_i)x^i ,$$

és így

$$\begin{aligned}\|(\overline{c}x-1)f(x)\|^2 &= \sum_{i=0}^{n+1}|\overline{c}a_{i-1}-a_i|^2 = \sum_{i=0}^{n+1}(|\overline{c}a_{i-1}|^2 + |a_i|^2 - \overline{c}a_{i-1}\overline{a_i} - c\overline{a_{i-1}}a_i) \\ &= \|f(x)\|^2 + |c|^2\|f(x)\|^2 - \sum_{i=0}^{n+1}(a_{i-1}\overline{ca_i} + \overline{a_{i-1}}ca_i) ,\end{aligned}$$

csakúgy, mint a bal oldal kifejtésénél. ■

18.74. tétel (Mignotte). *Tegyük fel, hogy $f(x), g(x) \in \mathbb{C}[x]$ komplex együtthatós, 1-főegyütthatójú polinomok és $g(x)|f(x)$. Ha $g(x)$ foka m, akkor $\|g(x)\| \leq 2^m\|f(x)\|$.*

Bizonyítás. Az algebra alaptétele szerint $f(x) = \prod_{i=1}^{n}(x-\alpha_i)$, ahol $\alpha_1, \ldots, \alpha_n$ az $f(x)$ polinom multiplicitással együtt figyelembe vett komplex gyökei. Ekkor $g(x) = \prod_{i\in I}(x-\alpha_i)$ valamely $I \subseteq \{1,\ldots,n\}$ részhalmazra. Először belátjuk, hogy tetszőleges $J \subseteq \{1,\ldots,n\}$ halmazra

$$\prod_{i\in J}|\alpha_i| \leq \|f(x)\| . \tag{18.19}$$

Ha J-ben van olyan i index, amelyre $\alpha_i = 0$, akkor ez az egyenlőtlenség nyilván teljesül. Feltehető tehát, hogy $i \in J$ esetén $\alpha_i \neq 0$. Legyen $\overline{J} = \{1, \ldots, n\} \setminus J$ és $h(x) = \prod_{i \in \overline{J}} (x - \alpha_i)$. A 18.73. lemma többszöri alkalmazásával azt kapjuk, hogy

$$\|f(x)\| = \|\prod_{i \in J} (x - \alpha_i) h(x)\| = \|\prod_{i \in J} (\overline{\alpha_i} x - 1) h(x)\| = |\prod_{i \in J} \overline{\alpha_i}| \cdot \|u(x)\| \,,$$

ahol $u(x) = \prod_{i \in J} (x - 1/\overline{\alpha_i}) h(x)$. Mivel $u(x)$ főegyütthatója 1, $\|u(x)\| \geq 1$, és így

$$|\prod_{i \in J} \alpha_i| = |\prod_{i \in J} \overline{\alpha_i}| = \|f(x)\| / \|u(x)\| \leq \|f(x)\| \,.$$

Fejezzük ki ezután $g(x)$ együtthatóit a gyökei segítségével:

$$\begin{aligned} g(x) &= \prod_{i \in I} (x - \alpha_i) = \sum_{J \subseteq I} \left((-1)^{|J|} \prod_{j \in J} \alpha_j x^{m - |J|} \right) \\ &= \sum_{i=0}^{m} (-1)^{m-i} \left(\sum_{J \subseteq I, |J| = m-i} \prod_{j \in J} \alpha_j \right) x^i \,. \end{aligned}$$

Tetszőleges $t(x) = t_0 + \cdots + t_k x^k$ polinomra nyilván érvényes a $\|t(x)\| \leq |t_0| + \cdots + |t_k|$ összefüggés. Ezt és a (18.19) egyenlőtlenségeket alkalmazva

$$\begin{aligned} \|g(x)\| &\leq \sum_{i=0}^{m} \left| \sum_{J \subseteq I, |J| = m-i} \prod_{j \in J} \alpha_j \right| \\ &\leq \sum_{J \subseteq I} \left| \prod_{j \in J} \alpha_j \right| \leq 2^m \|f(x)\| \,. \end{aligned}$$

A bizonyítás ezzel teljes. ■

18.75. következmény. *Egy $f(x) \in \mathbb{Z}[x]$ 1-főegyütthatós, egész együtthatós polinom $\mathbb{Q}[x]$-beli irreducibilis faktorainak a bitmérete polinomiális $f(x)$ bitméretében.*

Rezultáns, jó redukció

Legyen $\mathbb{F}$ egy tetszőleges test, $f(x), g(x) \in \mathbb{F}[x]$ n-edfokú, illetve m-edfokú polinomok: $f = a_0 + a_1 x + \ldots + a_n x^n$, $g = b_0 + b_1 x + \ldots + b_m x^m$, ahol $a_n \neq 0 \neq b_m$. Emlékeztetünk a ***rezultáns*** fogalmára az első kötet 2. fejezetéből. Az f és g rezultánsa, $Res(f, g)$ a következő $m + n \times m + n$-es M mátrix (ún. Sylvester-mátrix) determinánsa (az üres helyeken nullák

vannak):

$$M = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & \cdots & a_n & & & \\ & a_0 & a_1 & a_2 & \cdots & a_{n-1} & a_n & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} & a_n \\ b_0 & b_1 & \cdots & b_{m-1} & b_m & & & & \\ & b_0 & b_1 & \cdots & b_{m-1} & b_m & & & \\ & & b_0 & b_1 & \cdots & b_{m-1} & b_m & & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & b_0 & b_1 & \cdots & b_{m-1} & b_m \end{pmatrix} \tag{18.20}$$

A rezultáns információt ad f és g közös tényezőiről. Különösen elegánsan kifejezhető a segítségével az, hogy a két polinom relatív prím:

$$\operatorname{lnko}(f(x), g(x)) = 1 \Leftrightarrow Res(f, g) \neq 0\ . \tag{18.21}$$

18.76. következmény. *Legyen $f(x) = a_0 + a_1 x + \cdots + a_n x^n \in \mathbb{Z}[x]$ egy ($\mathbb{Q}[x]$-ben) négyzetmentes nem-konstans polinom. Ekkor $Res(f(x), f'(x))$ egész szám. Legyen továbbá p egy prím, ami nem osztja az na_n számot. Ekkor az $f(x) \pmod p$ polinom akkor és csak akkor lesz négyzetmentes $\mathbb{F}_p[x]$-ben, ha p nem osztója a $Res(f(x), f'(x))$ számnak.*

Bizonyítás. Az $f(x)$-hez és $f'(x)$-hez tartozó Sylvester-mátrix elemei egészek, így a determinánsa is egész. Az f-nek nincs többszörös tényezője $\mathbb{Q}$ felett, tehát a 18.5-1. gyakorlat alapján $\operatorname{lnko}(f(x), f'(x)) = 1$, amiből (18.21)-re tekintettel $Res(f(x), f'(x)) \neq 0$ következik. Jelölje $F(x)$ az f modulo p redukáltját. Ekkor a feltételeinkből következik, hogy $Res(F(x), F'(x))$ éppen a $Res(f(x), f'(x))$ szám modulo p maradéka. Az $F(x)$ a 18.5-1. gyakorlat szerint pontosan akkor négyzetmentes, ha $\operatorname{lnko}(F(x), F'(x)) = 1$, ami ekvivalens azzal, hogy $Res(F(x), F'(x)) \neq 0$. Ez pedig éppen azt jelenti, hogy a p nem osztja a $Res(f(x), f'(x))$ egészet. ■

18.77. következmény. *Legyen $f(x) \in \mathbb{Z}[x]$ négyzetmentes, n-edfokú polinom. Ekkor létezik olyan $p = O((n \lg n + 2n \lg \|f\|)^2)$ (tehát f bitméretében polinomiális nagyságú) p prímszám, amelyre az $f(x) \pmod p$ polinom négyzetmentes $\mathbb{F}_p[x]$-ben.*

Bizonyítás. A nagy prímszámtétel (33.37. tétel) alapján az $[1, K]$ intervallumba eső prímszámok szorzata legalább $2^{(0.9K/\ln K)}$, ha K elég nagy.

Legyen $K = ((n+1)\log_2 n + 2n \log_2 \|f\|)^2$. Ha K elég nagy, akkor tehát

$$p_1 \cdots p_l \geq 2^{(0.9K/\ln K)} > 2^{\sqrt{K}} \geq n^{n+1} \|f\|^{2n} \geq n^{n+1} \|f\|^{2n-1} |a_n| \tag{18.22}$$

teljesül, ahol $p_1, \ldots, p_l$ a K-nál nem nagyobb prímek, a_n pedig f főegyütthatója.

Tegyük fel, hogy a $p_1, \ldots, p_l$ prímekre $f(x) \pmod{p_i}$ nem négyzetmentes $\mathbb{F}_{p_i}[x]$-ben. Ekkor a $p_1 \ldots, p_l$ szorzat osztója a $Res(f(x), f'(x)) \cdot na_n$ számnak, és így

$$p_1 \cdots p_l \leq |Res(f, f')| \cdot |na_n| \leq \|f\|^{n-1} \cdot \|f'\|^n \cdot |na_n| \leq n^{n+1} \|f\|^{2n-1} |a_n|\ .$$

(Az utolsó két egyenlőtlenséghez az Hadamard-egyenlőtlenséget, illetve azt használtuk, hogy $\|f'(x)\| \le n\|f(x)\|$.) Ez pedig ellentmond a K választásából adódó (18.22) egyenlőtlenségnek. ■

Megjegyezzük, hogy itt a nagy prímszámtétel pontosabb alkalmazásával egy erősebb – p-re vonatkozó – felső korlát is igazolható.

Hensel-felemelés

Itt egy olyan általános technikát ismertetünk, amellyel egy egész együtthatós polinom modulo p (p egy prím) felbontásából modulo p^N felbontást kaphatunk.

18.78. tétel (Hensel-lemma). *Tegyük fel, hogy az $f(x), g(x), h(x) \in \mathbb{Z}[x]$ egész együtthatós, 1-főegyütthatós polinomokra $f(x) \equiv g(x)h(x) \pmod p$, továbbá, hogy a $g(x) \pmod p$ és a $h(x) \pmod p$ relatív prímek $\mathbb{F}_p[x]$-ben. Ekkor tetszőleges t pozitív egész számra léteznek olyan $g_t(x), h_t(x) \in \mathbb{Z}[x]$ egész együtthatós, 1-főegyütthatójú polinomok, amelyekre*

- *$g_t(x)$ és $h_t(x)$ főegyütthatója 1,*
- *$g_t(x) \equiv g(x) \pmod p$ és $h_t(x) \equiv h(x) \pmod p$,*
- *$f(x) \equiv g_t(x)h_t(x) \pmod{p^t}$.*

Az is igaz, hogy a fenti feltételek a $g_t(x)$ és $h_t(x)$ polinomokat egyértelműen meghatározzák modulo p^t.

Bizonyítás. A főegyütthatókra vonatkozó feltételekből $\deg f(x) = \deg g(x) + \deg h(x)$, továbbá $\deg g_t(x) = \deg g(x)$ és $\deg h_t(x) = \deg h(x)$, legalábbis ha léteznek a megfelelő $g_t(x)$ és $h_t(x)$ polinomok. A létezést t szerinti indukcióval bizonyítjuk. A $t = 1$ kezdőesetben $g_1(x) = g(x), h_1(x) = h(x)$ megfelelő választás.

A $t \to t+1$ indukciós lépés: tegyük fel, hogy $g_t(x)$ és $h_t(x)$ olyan modulo p^t egyértelműen meghatározott polinomok, amelyek kielégítik a feltételeket. Ekkor, mivel $g_{t+1}(x)$ és $h_{t+1}(x)$ (amennyiben egyáltalán léteznek) kielégítik a $g_t(x)$-re és $h_t(x)$-re vonatkozó feltételeket, az utóbbi polinomok modulo p^t egyértelműsége miatt $g_{t+1}(x) = g_t(x) + p^t\delta_g(x)$ és $h_{t+1}(x) = h_t(x) + p^t\delta_h(x)$ alakba írhatók, ahol $\delta_g(x)$ és $\delta_h(x)$ egész együtthatós polinomok. A főegyütthatókra vonatkozó feltétel miatt az is igaz, hogy $\deg\delta_g(x) < \deg g(x)$, illetve $\deg\delta_h(x) < \deg h(x)$.

Az indukciós feltevés miatt $f(x) = g_t(x)h_t(x) + p^t\lambda(x)$, ahol $\lambda(x) \in \mathbb{Z}[x]$. A $g_t(x)$, illetve $h_t(x)$ polinom fokszámára vonatkozó megállapítások alapján $\lambda(x)$ foka kisebb $\deg f(x)$-nél.

$$\begin{aligned} g_{t+1}(x)h_{t+1}(x) - f(x) &= g_t(x)h_t(x) - f(x) + p^t h_t(x)\delta_g(x) + p^t g_t(x)\delta_h(x) + p^{2t}\delta_g(x)\delta_h(x) \\ &\equiv -p^t\lambda(x) + p^t h_t(x)\delta_g(x) + p^t g_t(x)\delta_h(x) \pmod{p^{2t}} . \end{aligned}$$

Mivel $2t > t+1$, a fenti kongruencia modulo p^{t+1} is fennáll. Tehát $g_{t+1}(x)$ és $h_{t+1}(x)$ akkor és csak akkor felel meg a feltételeknek, ha

$$p^t h_t(x)\delta_g(x) + p^t g_t(x)\delta_h(x) \equiv p^t\lambda(x) \pmod{p^{t+1}} ,$$

ez pedig (p^t-vel egyszerűsítve) a

$$h_t(x)\delta_g(x) + g_t(x)\delta_h(x) \equiv \lambda(x) \pmod p$$

kongruenciával ekvivalens. A $g_t(x) \equiv g(x) \pmod p$ és a $h_t(x) \equiv h(x) \pmod p$ kongruenciákat használva ez tovább egyenértékű a

$$h(x)\delta_g(x) + g(x)\delta_h(x) \equiv \lambda(x) \pmod p \tag{18.23}$$

kongruenciával. A fokszámokra vonatkozó feltételeket (a $\deg \delta_g(x) < \deg g_t(x)$ és $\deg \delta_h(x) < \deg h_t(x)$ egyenlőtlenségeket) is tekintetbe véve, mivel $\mathbb{F}_p[x]$-ben a $g(x) \pmod p$ és a $h(x) \pmod p$ polinomok relatív prímek, (18.23) egyértelműen megoldható $\mathbb{F}_p[x]$-ben. Legyenek ugyanis $u(x)$ és $v(x)$ az $u(x)g(x) + v(x)h(x) \equiv 1 \pmod p$ megoldásai (18.12. tétel). Ekkor

$$\delta_g(x) \quad = \quad v(x)\lambda(x) \pmod{g(x)}\,,$$

illetve

$$\delta_h(x) \quad = \quad u(x)\lambda(x) \pmod{h(x)}$$

polinomok (18.23) megoldásai lesznek. A megoldás egyértelműsége könnyen adódik a fokszámkorlátokból, és abból, hogy $g(x) \pmod p$ és $h(x) \pmod p$ relatív prímek. Ennek részleteit az Olvasóra bízzuk. ■

18.79. következmény. *Tegyük fel, hogy p, $f(x), g(x), h(x) \in \mathbb{Z}[x]$ eleget tesznek a Hensel-lemma feltételeinek. Legyen* $\deg f = n$ *és N egy pozitív egész. Ekkor a $g_N(x)$ és $h_N(x)$ polinomok megkaphatók $O(Nn^2)$ modulo p^N aritmetikai művelettel.*

Bizonyítás. A 18.78. tétel bizonyítása a következő algoritmust sugallja.

HENSEL-FELEMELÉS (f, g, h, p, N)

1 $(u(x), v(x)) \leftarrow$ az $u(x)g(x) + v(x)h(x) \equiv 1 \pmod p$ egy megoldása ($\mathbb{F}_p[x]$-ben)
2 $(G(x), H(x)) \leftarrow (g(x), h(x))$
3 **for** $t \leftarrow 1$ **to** $N - 1$
4 **do** $\lambda(x) \leftarrow (f(x) - G(x) \cdot H(x))/p^t$
5 $\delta_g(x) \leftarrow v(x)\lambda(x)$ redukáltja modulo $g(x)$ ($\mathbb{F}_p[x]$-ben)
6 $\delta_h(x) \leftarrow u(x)\lambda(x)$ redukáltja modulo $h(x)$ ($\mathbb{F}_p[x]$-ben)
7 $(G(x), H(x)) \leftarrow (G(x) + p^t\delta_g(x), H(x) + p^t\delta_h(x))$ $((\mathbb{Z}/(p^{t+1}))[x]$-ben)
8 **return** $(G(x), H(x))$

Az u, v polinomok $O(n^2)$ $\mathbb{F}_p$-beli művelettel adódnak (18.12. tétel, és az utána levő megjegyzés).

Ezek után egy $t \to t+1$ iterációs menet konstans sok polinomműveletből áll, egy menet költsége pedig $O(n^2)$ művelet (modulo p, illetve p^{t+1}). Az összköltség $t = N$-ig: $O(Nn^2)$ művelet. ■

18.5.2. A Berlekamp–Zassenhaus-algoritmus

A (18.18) felbontási feladatot hatékony redukcióval visszavezettük arra az esetre, amikor a felbontandó f egész együtthatós és 1-főegyütthatós polinom. Feltehetjük azt is, hogy

$f(x)$-nek nincs többszörös tényezője $\mathbb{Q}[x]$-ben. Ugyanis esetünkben $f'(x) \neq 0$, és így az f esetleges többszörös tényezői a véges testek feletti polinomok felbontásánál már alkalmazott ötlettel leválaszthatók: a 18.13. lemma szerint a $g(x) = f(x)/(f(x), f'(x))$ polinom már négyzetmentes, és a 18.14. lemma miatt nyilván elég ennek az irreducibilis faktorait megkeresni. A 18.71. állításból látjuk, hogy $g(x)$ is egész együtthatós és 1-főegyütthatós. A legnagyobb közös osztó és a polinomok hányadosa hatékonyan számítható, vagyis a visszavezetés polinom időben elvégezhető. (A legnagyobb közös osztó kiszámításakor a köztes tárrobbanás például az első kötet második fejezetében ismertetett moduláris technikákkal kerülhető el.)

A továbbiakban feltesszük, hogy a felbontandó

$$f(x) = x^n + \sum_{i=0}^{n-1} a_i x^i \in \mathbb{Z}[x]$$

egy egész együtthatós, 1-főegyüttható, négyzetmentes polinom.

A Berlekamp–Zassenhaus-algoritmus alapötlete az, hogy megpróbáljuk $f(x)$ irreducibilis tényezőit modulo p^N kiszámítani, ahol p egy alkalmas prímszám és N elég nagy. Például, ha $p^N > 2 \cdot 2^{n-1}\|f\|$ és egy tényező együtthatóit modulo p^N kiszámoltuk, akkor Mignotte tétele alapján a tényleges együtthatók is a rendelkezésünkre állnak.

Mostantól azt is feltesszük, hogy p egy olyan prímszám, amelyre az $f(x) \pmod p$ polinom négyzetmentes $\mathbb{F}_p[x]$-ben. Ilyen p prím (lineáris kereséssel) polinom időben található (18.77. következmény), sőt az is feltehető, hogy a p prím az $f(x)$ polinom bitméretében polinomiális nagyságú.

Ekkor az $f(x) \pmod p$ polinom $\mathbb{F}_p[x]$-beli irreducibilis faktorai Berlekamp determinisztikus módszerével polinom időben megtalálhatók (18.42. tétel). Legyenek tehát $g_1(x), \ldots, g_r(x) \in \mathbb{Z}[x]$ olyan 1-főegyütthatós polinomok, amelyekre $g_i(x) \pmod p$ az $f(x) \pmod p$ polinom irreducibilis tényezői $\mathbb{F}_p[x]$-ben.

A Hensel-lemma (18.78. tétel) módszerével az egész $g_1(x), \ldots, g_r(x)$ rendszer felemelhető modulo p^N. Egyszerűsítve a jelölést, ezek után feltesszük, hogy $g_1(x), \ldots, g_r(x) \in \mathbb{Z}[x]$ olyan 1-főegyütthatós polinomok, amelyekre

$$f(x) \equiv g_1(x) \cdots g_r(x) \pmod{p^N}$$

és $g_i(x) \pmod p$ az $f(x) \pmod p$ polinom irreducibilis faktorai $\mathbb{F}_p[x]$-ben.

Legyen $h(x) \in \mathbb{Z}[x]$ az $f(x)$ polinomnak egy (1-főegyütthatójú) irreducibilis tényezője $\mathbb{Q}[x]$-ben. Ekkor létezik olyan egyértelműen meghatározott $I \subseteq \{1, \ldots, r\}$ halmaz, amelyre

$$h(x) \equiv \prod_{i \in I} g_i(x) \pmod{p^N}\,.$$

Legyen N a legkisebb pozitív egész, amelyre $p^N \geq 2 \cdot 2^{n-1}\|f(x)\|$. A Mignotte-korlát mutatja, hogy a jobb oldalon álló $\prod_{i \in I} g_i(x) \pmod{p^N}$ polinom – ha az együtthatóit a legkisebb abszolút értékű maradékokkal reprezentáljuk – egyenlő h-val.

Arra jutottunk, hogy az $f(x)$ irreducibilis tényezőinek meghatározása egyenértékű az olyan minimális $I \subseteq \{1, \ldots, r\}$ részhalmazok megkeresésével, amelyekre létezik olyan $h(x) \in \mathbb{Z}[x]$ 1-főegyütthatójú polinom, hogy $h(x) \equiv \prod_{i \in I} g_i(x) \pmod{p^N}$, a $h(x)$ együtthatói legfeljebb $2^{n-1}\|f(x)\|$ abszolút értékűek, továbbá $h(x)$ osztója $f(x)$-nek. Ez legfeljebb 2^{r-1} darab I halmaz megvizsgálásával ellenőrizhető. Az egyetlen I-hez kapcsolódó munka

mennyisége polinomiális az f méretében.

Összefoglalva, a következő módszer adódik az $f(x)$ egész együtthatós, 1-főegyütthatós, négyzetmentes polinom $\mathbb{Q}[x]$-beli felbontására:

BERLEKAMP-ZASSENHAUS(f)

1 $p \leftarrow$ olyan p prím, amelyre $f(x) \pmod p$ négyzetmentes $\mathbb{F}_p[x]$-ben
és $p = O((n \lg n + 2n \lg \|f\|)^2)$
2 $\{g_1, \ldots, g_r\} \leftarrow$ az $f(x)$ modulo p polinom irreducibilis tényezői $\mathbb{F}_p[x]$-ben
(Berlekamp determinisztikus módszerével)
3 $N \leftarrow \lfloor \lg_p(2^{\deg f} \cdot \|f\|) \rfloor + 1$
4 $\{g_1, \ldots, g_r\} \leftarrow$ a $\{g_1, \ldots, g_r\}$ rendszer Hensel-felemeltje modulo p^N
5 $\mathcal{I} \leftarrow$ az $\{1, \ldots r\}$ összes minimális olyan $I \neq \emptyset$ részhalmaza, amelyre
$g_I \leftarrow \prod_{i \in I} g_i$ redukáltja modulo p^N osztója f-nek
6 **return** $\{\prod_{i \in I} g_i \,:\, I \in \mathcal{I}\}$

18.80. tétel. *Legyen $f(x) = x^n + \sum_{i=0}^{n-1} a_i x^i \in \mathbb{Z}[x]$ egész együtthatós, 1-főegyütthatós, négyzetmentes polinom, p egy olyan prímszám, amelyre az $f(x) \pmod p$ polinom négyzetmentes $\mathbb{F}_p[x]$-ben és $p = O((n \lg n + 2n \lg \|f\|)^2)$. Ekkor az f polinom $\mathbb{Q}[x]$-beli irreducibilis tényezői megkaphatók a Berlekamp–Zassenhaus-algoritmussal. Ennek költsége polinomiális az n, $\lg \|f(x)\|$ és 2^r paraméterekben, ahol r az $f(x) \pmod p$ polinom $\mathbb{F}_p[x]$-beli irreducibilis tényezőinek a száma.*

18.5. példa. (Swinnerton–Dyer-polinomok.) Legyen

$$f(x) = \prod (x \pm \sqrt{2} \pm \sqrt{3} \pm \cdots \pm \sqrt{p_l}) \in \mathbb{Z}[x] \,,$$

ahol $2, 3, \ldots, p_l$ az első l prímszám, és a szorzás kiterjed mind a 2^l lehetséges $\pm$-kombinációra. Az $f(x)$ foka $n = 2^l$, és megmutatható, hogy irreducibilis $\mathbb{Q}[x]$-ben. Másfelől minden p prímre $f(x) \pmod p$ legfeljebb másodfokú tényezők szorzata. Ezek a polinomok tehát nehéz esetnek számítanak a Berlekamp–Zassenhaus-algoritmus szempontjából, hiszen $\geq 2^{n/2}$ darab I halmaz megvizsgálása után derül csak ki, hogy f felbonthatatlan.

18.5.3. Az LLL-algoritmus

A célunk itt az $f(x) \in \mathbb{Q}[x]$ polinomok felbontására szolgáló Lenstra–Lenstra–Lovász-algoritmus (röviden: LLL-algoritmus) ismertetése. Ez volt az első polinom idejű módszer a $\mathbb{Q}$ feletti felbontási feladat megoldására. A Berlekamp–Zassenhaus-módszerhez hasonlóan az LLL-algoritmus is az f modulo p felbontásából indul ki, és Hensel-felemelést is használ. A munka végső fázisában erőteljes új ötlet alkalmazásával, rácsredukció segítségével találja meg az f egy valódi osztóját (amennyiben létezik). Úgy fogalmazhatunk, hogy a Berlekamp–Zassenhaus-algoritmus lassú (bizonyos esetekben exponenciális lépésszámú) keresőfázisát a hatékony rácsredukció helyettesíti.

Legyen továbbra is $f(x) \in \mathbb{Z}[x]$, $\deg f = n > 1$, egész együtthatós, 1-főegyütthatós, négyzetmentes polinom, p egy olyan prímszám, amelyre az $f(x) \pmod p$ polinom négyzetmentes $\mathbb{F}_p[x]$-ben, és $p = O((\lg n + 2n \lg \|f\|)^2)$.

18.81. lemma. *Tegyük fel, hogy $f(x) \equiv g_0(x)v(x) \pmod{p^N}$, ahol $g_0(x)$ és $v(x)$ 1-főegyüttható egész együtthatós polinomok. Legyen $g(x) \in \mathbb{Z}[x]$, $\deg g(x) = m < n$ és tegyük fel, hogy $g(x) \equiv g_0(x)u(x) \pmod{p^N}$ valamilyen egész együtthatós $u(x)$ polinomra, amelyre $\deg u(x) = \deg g(x) - \deg g_0(x)$. Tegyük továbbá fel, hogy $\|g(x)\|^n \|f(x)\|^m < p^N$. Ekkor a $\mathbb{Q}[x]$-ben $\operatorname{lnko}(f(x), g(x)) \neq 1$ teljesül.*

Bizonyítás. Legyen $d = \deg v(x)$. A feltevések miatt

$$f(x)u(x) \equiv g_0(x)u(x)v(x) \equiv g(x)v(x) \pmod{p^N} .$$

Legyen $u(x) = \alpha_0 + \alpha_1 x + \ldots + \alpha_{m-1}x^{m-1}$, $v(x) = \beta_0 + \beta_1 x + \ldots + \beta_{n-1}x^{n-1}$. (Tudjuk hogy $\beta_d = 1$. Ha $i > d$, akkor $\beta_i = 0$, és hasonlóan, ha $j > \deg u(x)$, akkor $\alpha_j = 0$.) Átírva a kongruenciát:

$$x^d g(x) + \sum_{j \neq d} \beta_j x^j g(x) - \sum_i \alpha_i x^i f(x) \equiv 0 \pmod{p^N} .$$

Az $x^j g(x)$, illetve $x^i f(x)$ polinomok $n+m$-hosszú együtthatóvektorait szemlélve ez azt jelenti, hogy ha a (18.20) formulában megadott M Sylvester-mátrix $m+d$-edik sorához hozzáadjuk a többi sor alkalmas skalárszorosát, akkor ez a sor csupa p^N-nel osztható elemből áll. Következésképpen $\det M \equiv 0 \pmod{p^N}$. Az Hadamard-egyenlőtlenség (18.60. következmény) miatt $|\det M| \leq \|f\|^m \|g\|^n < p^N$, ez pedig csak úgy lehet, hogy $\det M = 0$. Viszont $\det M$ éppen $Res(f(x), g(x))$, és így (18.21) alapján $\operatorname{lnko}(f(x), g(x)) \neq 1$. ■

A rácsredukció alkalmazása:
Legyen

$$N = \lceil \log_p (2^{2n^2} \|f(x)\|^{2n}) \rceil = O(n^2 + n \lg \|f(x)\|) .$$

Legyen továbbá $g_0(x) \in \mathbb{Z}[x]$ egy olyan 1-főegyüttható polinom, amelyre $g_0(x) \pmod{p^N}$ az $f(x) \pmod{p^N}$ polinomnak egy irreducibilis faktora modulo p^N. Legyen $d = \deg g_0(x) < n$. Definiáljuk a L halmazt a következőképpen:

$$L = \{g(x) \in \mathbb{Z}[x] : \ \deg g(x) \leq n-1, \ \exists h(x) \in \mathbb{Z}[x], \text{ amellyel } g \equiv h g_0 \pmod{p^N}\} . \quad (18.24)$$

Az L halmaz nyilván zárt a polinomok összeadására. Azonosítsuk az n-nél alacsonyabb fokú polinomokat az (n hosszú) együtthatóvektoraikkal. Ezzel a megfeleltetéssel L egy rács $\mathbb{R}^n$-ben: nem nehéz megmutatni (18.5-2. gyakorlat), hogy

$$p^N 1, p^N x, \ldots, p^N x^{d-1}, \ g_0(x), x g_0(x), \ldots, x^{n-d-1} g_0(x) ,$$

pontosabban fogalmazva az itt szereplő polinomok együtthatóvektorai az L bázisát alkotják.

18.82. tétel. *Legyen $g_1(x) \in \mathbb{Z}[x]$ egy olyan n-nél alacsonyabb fokú polinom, amely (pontosabban: amelynek együtthatóvektora) az L rács egy tetszőleges Lovász-redukált bázisának az első eleme. Ekkor az $f(x)$ pontosan akkor irreducibilis $\mathbb{Q}[x]$-ben, ha $(f(x), g_1(x)) = 1$.*

Bizonyítás. A $g_1(x) \neq 0$, ezért nyilvánvaló, hogy irreducibilis $f(x)$ esetén $(f(x), g_1(x)) = 1$. A másik irányú implikáció igazolásához tegyük fel, hogy $f(x)$ felbontható, és legyen $g(x)$ olyan valódi osztója $f(x)$-nek, amelyre $g(x) \pmod p$ osztható a $g_0(x) \pmod p$ polinommal $\mathbb{F}_p[x]$-ben. Ekkor a Hensel-lemmából (18.78. tétel) arra következtethetünk, hogy $g(x) \pmod{p^N}$ is osztható $g_0(x) \pmod{p^N}$-nel, azaz $g(x) \in L$. Mignotte tétele (18.74. tétel) szerint

$$\|g(x)\| \leq 2^{n-1}\|f(x)\| .$$

A redukált bázisok tulajdonságai (18.67. tétel, 2. állítás) miatt viszont

$$\|g_1(x)\| \leq 2^{(n-1)/2}\|g(x)\| < 2^n\|g(x)\| \leq 2^{2n}\|f(x)\| ,$$

és így

$$\|g_1(x)\|^n\|f(x)\|^{\deg g_1} \leq \|g_1(x)\|^n\|f(x)\|^n < 2^{2n^2}\|f(x)\|^{2n} \leq p^N .$$

Alkalmazható tehát a 18.81. lemma, amiből $(g_1(x), f(x)) \neq 1$. ■

Az előzőek alapján az LLL-algoritmus a következőképpen vázolható (itt is csak a két tényezőre bontás lépéseit adjuk meg). A bemenet egy $f(x) \in \mathbb{Z}[x]$ egész együtthatós, 1-főegyütthatós, négyzetmentes polinom, $\deg f = n > 1$.

LLL-POLINOMFELBONTÁS(f)

```
1  p ← olyan p prím, amelyre f(x) (mod p) négyzetmentes F_p[x]-ben,
       és p = O((n lg n + 2n lg ||f||)^2)
2  w(x) ← f(x) (mod p) egy irreducibilis faktora F_p[x]-ben
       (Berlekamp determinisztikus módszerével)
3  if deg w = n
4     then return "irreducibilis"
5     else N ← ⌈log_p((2^{2n^2}||f(x)||^{2n})⌉ = O(n^2 + n lg(||f(x)||))
6          (g_0, h_0) ← Hensel-felemelés(f, w, f/w (mod p), p, N)
7          (b_1, ..., b_n) ← a (18.24)-beli L ⊆ R^n rács bázisa
8          (g_1, ..., g_n) ← Lovász-redukció(b_1, ..., b_n)
9          f* ← lnko(f, g_1)
10         if deg f* > 0
11            then return (f*, f/f*)
12            else return "irreducibilis"
```

18.83. tétel. *Az LLL-algoritmus alkalmazásával az $f \in \mathbb{Q}[x]$ polinomok $\mathbb{Q}[x]$-beli irreducibilis tényezői determinisztikus polinom időben megkaphatók.*

Bizonyítás. Az általános felbontási feladat a Berlekamp–Zassenhaus-módszer kapcsán tárgyalt módon polinom időben visszavezethető a négyzetmentes, 1-főegyütthatós $f(x) \in \mathbb{Z}[x]$ esetére. Az ott taglaltak szerint polinom időben megvalósíthatók az 1–7. sorokban leírt lépések. A 9. sorban a Lovász-redukció is hatékonyan végezhető el (18.65. következmény). A 10. sorban az euklideszi algoritmus moduláris változatát használhatjuk a köztes tárrobbanás elkerülésére (lásd az első kötet 2. fejezetét).

A módszer helyességét a 18.82. tétel fogalmazza meg. Az LLL-algoritmus alkalmazható a kapott tényezők további felbontására. ■

Megmutatható, hogy a Hensel-felemelés $O(Nn^2) = O(n^4 + n^3 \lg \|f\|)$ alapműveletet jelent nem túl nagy egészekkel. A polinomokat két tényezőre bontó LLL-algoritmus összköltsége így $O(n^5 \lg(p^N)) = O(n^7 + n^6 \lg \|f\|)$ művelet lesz.

Gyakorlatok

18.5-1. Legyen $\mathbb{F}$ test, $0 \neq f(x) \in \mathbb{F}[x]$. Az $f(x)$-nek pontosan akkor nincs többszörös irreducibilis tényezője, ha $\mathrm{lnko}(f(x), f'(x)) = 1$. (*Útmutatás.* Az egyik irányban alkalmazható a 18.13. lemma, míg másik irányban a 18.14. lemma.)

18.5-2. Mutassuk meg, hogy a (18.24) rács bázisát alkotják a

$$p^N 1, p^N x, \ldots, p^N x^{d-1}, g_0(x), x g_0(x), \ldots, x^{n-d-1} g_0(x)$$

polinomok. (*Útmutatás.* Elég belátni, hogy a $p^N x^j$ polinomok ($d \leq j < n$) kifejezhetők a megadott polinomokkal. Osszuk el maradékosan $p^N x^j$-t a $g_0(x)$ polinommal.)

Feladatok

18-1. A nyom véges testekben

Legyenek $\mathbb{F}_{q^k} \supseteq \mathbb{F}_q$ véges testek. Az $\mathbb{F}_{q^k}$-n értelmezett $tr = tr_{k,q}$ leképezés definíciója a következő: az $\alpha \in \mathbb{F}_{q^k}$ elemre

$$tr(\alpha) = \alpha + \alpha^q + \cdots + \alpha^{q^{k-1}} .$$

a. Mutassuk meg, hogy a tr leképezés $\mathbb{F}_q$-lineáris, és a képtere pontosan $\mathbb{F}_q$. (*Útmutatás:* annak az igazolásához, hogy tr nem azonosan nulla, használjuk, hogy tr egy q^{k-1}-edfokú polinommal adott.)

b. Legyen (α, β) az egyenletes eloszlás szerint választott véletlen elempár $\mathbb{F}_{q^k} \times \mathbb{F}_{q^k}$-ból. Ekkor $1 - 1/q$ annak a valószínűsége, hogy $tr(\alpha) \neq tr(\beta)$.

18-2. A Cantor–Zassenhaus-algoritmus 2 karakterisztikájú testek esetén

Legyen $\mathbb{F} = \mathbb{F}_{2^m}$ és $f(x) \in \mathbb{F}[x]$ egy

$$f = f_1 f_2 \cdots f_s \tag{18.25}$$

alakú polinom, ahol az f_i polinomok páronként relatív prím d-edfokú irreducibilis polinomok $\mathbb{F}[x]$-ben, és $s \geq 2$.

a. Legyen az $u(x) \in \mathbb{F}[x]$ az egyenletes eloszlás szerint választott deg f-nél kisebb fokú polinom. Ekkor az

$$\mathrm{lnko}(u(x) + u^2(x) + \cdots + u^{2^{d-1}}(x), f(x))$$

legnagyobb közös osztó legalább 1/2 valószínűséggel valódi osztója az $f(x)$ polinomnak.

Útmutatás. Alkalmazzuk az előző feladatot a $q = 2^m$ és $k = d$ választással, és kövessük a 18.38. tétel érvelését.

b. Az a. alapján adjunk véletlenített polinom idejű algoritmust a (18.25) alakú polinomok $\mathbb{F}$ feletti felbontására.

18-3. Osztók és nullosztók

Legyen $\mathbb{F}$ test. Az R gyűrűt $\mathbb{F}$-algebrának (ha az $\mathbb{F}$ világos a szövegkörnyezetből, akkor röviden algebrának) nevezzük, ha R vektortér az $\mathbb{F}$ felett, és $(ar)s = a(rs) = r(as)$ igaz minden $r, s \in S$ és $a \in \mathbb{F}$ elemre. Az $\mathbb{F}[x]$ és $\mathbb{F}[x]/(f)$ gyűrűkről könnyű látni, hogy egyben $\mathbb{F}$-algebrák is.

Legyen R véges dimenziós $\mathbb{F}$-algebra. Tetszőleges $r \in R$ elemre tekinthetjük az $L_r : R \to R$ leképezést, amelyre $L_r(s) = rs$, ha $s \in R$. Az L_r egy $\mathbb{F}$ lineáris leképezés, így beszélhetünk az $m_r(x) \in \mathbb{F}[x]$ minimálpolinomjáról, a $k_r(x) \in \mathbb{F}[x]$ karakterisztikus polinomjáról, és a $Tr(r) = Tr(L_r)$ nyomáról is. Sőt, ha U ideál R-ben, akkor az U invariáns altere L_r-nek, így értelmes az U-ra megszorított L_r leképezés minimálpolinomja, karakterisztikus polinomja és nyoma is.

a. Legyen $f(x), g(x) \in \mathbb{F}[x]$, $\deg f > 0$. Mutassuk meg, hogy a $[g(x)]$ maradékosztály pontosan akkor nullosztó az $\mathbb{F}[x]/(f)$ gyűrűben, ha f nem osztója g-nek, és $(f(x), g(x)) \neq 1$.

b. Legyen R algebra $\mathbb{F}$ felett, és legyen az $r \in R$ minimálpolinomja $f(x)$. Igazoljuk, hogy ha f nem irreducibilis $\mathbb{F}$ felett, akkor van R-ben nullosztó. Pontosabban, ha $f(x) = g(x)h(x)$ nem triviális felbontás ($g, h \in \mathbb{F}[x]$), akkor $g(r)$ és $h(r)$ egy nullosztópár (azaz egyikük sem 0, de a szorzatuk igen).

18-4. Polinomok felbontása algebrai számtestek felett

a. Legyen $\mathbb{F}$ nulla karakterisztikájú test, R véges dimenziós $\mathbb{F}$-algebra. Tegyük fel, hogy $R = S_1 \oplus S_2$, ahol S_1 és S_2 nem nulla $\mathbb{F}$-algebrák. Legyen $r_1, \dots, r_k$ az R egy $\mathbb{F}$-bázisa. Mutassuk meg, hogy van olyan j, amelyre $m_{r_j}(x)$ felbontható $\mathbb{F}[x]$-ben.

Útmutatás. Ezt a feladatot a lineáris algebra elemeiben jártas Olvasónak szánjuk. Tegyük fel, hogy az r_j minimálpolinomja az $m(x) = x^d - a_1 x^{d-1} + \dots + a_d$ irreducibilis polinom. Legyen $k_i(x)$ az L_{r_j} karakterisztikus polinomja az U_i invariáns altéren ($i \in \{1, 2\}$). Itt U_1 az $(s_1, 0)$, U_2 pedig a $(0, s_2)$ alakú párok halmaza ($s_i \in S_i$). A feltételeink miatt alkalmas d_i kitevőkkel $k_i(x) = m(x)^{d_i}$ teljesül. Innen az L_{r_j} leképezés $T_i(r_j)$ nyoma az U_i altéren $T_i(r_j) = d_i a_1$ lesz. Legyen $e_i = \dim_{\mathbb{F}} U_i$. Nyilvánvalóan $e_i = d_i d$, amiből $T_1(r_j)/e_1 = T_2(r_j)/e_2$. Ha a feladat állítása hamis, akkor az utóbbi összefüggés minden j-re teljesül, így a nyom linearitása miatt minden $r \in R$ elemre fennáll. Ez pedig ellentmondáshoz vezet: legyen $r = (1, 0) \in S_1 \oplus S_2$. Itt 1 az S_1 egységeleme. Világos, hogy $T_1 r) = e_1$ és $T_2(r) = 0$.

b. Legyen $\mathbb{F}$ ***algebrai számtest,*** vagyis egy $\mathbb{Q}(\alpha)$ alakú test, ahol $\alpha \in \mathbb{C}$, és van olyan irreducibilis $g(x) \in \mathbb{Z}[x]$ polinom, amellyel $g(\alpha) = 0$. Legyen $f(x) \in \mathbb{F}[x]$ négyzetmentes és $R = \mathbb{F}[x]/(f)$. Mutassuk meg, hogy R véges dimenziós algebra $\mathbb{Q}$ felett. Pontosabban,

ha $\deg g = m$ és $\deg f = n$, akkor az $\alpha^i[x]^j$ alakú elemek ($0 \le i < m$, $0 \le j < n$) bázist alkotnak $\mathbb{Q}$ felett.

c. Mutassuk meg, hogy ha f felbontható $\mathbb{F}$ felett, akkor vannak olyan S_1, S_2 $\mathbb{Q}$-algebrák, amelyekkel $R \cong S_1 \oplus S_2$.

Útmutatás. Alkalmazzuk a kínai maradéktételt.

d. Tegyük fel, hogy (a g polinommal) adott $\mathbb{F}$ és az $f \in \mathbb{F}[x]$ polinom. Tegyük még fel, hogy f négyzetmentes és nem irreducibilis $\mathbb{F}$ felett. Ekkor f polinom időben felbontható két nem konstans $\mathbb{F}[x]$-beli polinom szorzatára.

Útmutatás. Az előzőek alapján az $\alpha^i[x]^j$ elemek ($0 \le i \le m$, $0 \le j \le n$) valamelyikének a $\mathbb{Q}$ feletti $m(x)$ minimálpolinomja nem irreducibilis a $\mathbb{Q}$ felett. Az $m(x)$ az LLL-algoritmussal hatékonyan felbontható a $\mathbb{Q}[x]$-ben. Az $m(x)$ felbontásából R-beli nullosztó kapható, ebből pedig az f egy valódi $\mathbb{F}[x]$-beli osztója nyerhető.)

Megjegyzések a fejezethez

Az itt tárgyalt absztrakt algebrai alapokról bőséges anyag található a témakör tankönyveiben, például Hungerford [165] művében. A magyar nyelvű szakkönyvek közül Fried Ervin munkáit [119, 120] ajánljuk.

A véges testek elméletével és a hozzájuk kapcsolódó algoritmusokkal foglalkoznak Lidl és Niederreiter [227], valamint Shparlinski [312] kitűnő könyvei.

A fejezet nagyobb algoritmikus témáit (polinomok felbontása véges testek és $\mathbb{Q}$ felett, rácsredukció) alaposan tárgyalja von zur Gathen és Gerhard [128]. A polinomok körében elvégzendő alapfeladatok megoldására szolgáló hatékony módszerek iránt érdeklődő Olvasónak szintén ezt a könyvet ajánljuk. Schönhage–Strassen-féle polinomszorzó módszer költségéről szól az idézett mű 8.23. tétele, míg az euklideszi algoritmus aszimptotikusan gyors implementációjának költségéről a 11.6. következményben olvashatunk.

Ajtai Miklósnak a rácsredukcióval kapcsolatos eredménye a [13] dolgozatban jelent meg.

A véges testek feletti polinomok felbontását illetően Kaltofen és Shoup módszere egyike a legjobb időkorláttal rendelkező véletlent használó algoritmusoknak. Algoritmusuk várhatóan $O(n^{1.815} \lg q)$ $\mathbb{F}_q$-beli művelet árán kapja meg a teljes felbontást (n az f polinom foka). További versenyképes véletlen módszereket javasoltak von zur Gathen és Shoup, valamint Huang és Pan. Az utóbbi műveletigénye $O(n^{1.80535} \lg q)$, ha $\lg q < n^{0.00173}$. A determinisztikus algoritmusok közül von zur Gathen és Shoup módszere a jelenlegi bajnok. A költsége $\widetilde{O}(n^2 + n^{3/2}s + n^{3/2}s^{1/2}p^{1/2})$ $\mathbb{F}_q$-beli művelet, ahol $q = p^s$. Fontos kapcsolódó feladat az $\mathbb{F}_{q^n}$ test explicit konstrukciója. A leggyorsabb (véletlent használó) módszer Shoup nevéhez fűződik, a lépésszáma $O^{\sim}(n^2 + n \lg q)$. A négyzetmentes felbontásra Yun adott $\widetilde{O}(n) + O(n \lg(q/p))$ $\mathbb{F}_q$-beli műveletet igénylő algoritmust.

A rácsredukció feladatára és a racionális együtthatós polinomok felbontására a legjobb módszerek moduláris és numerikus technikákat használnak. Előbbire – a redukáltság kicsit módosított definíciójával – Storjohann adott $\widetilde{O}(n^{3.381} \lg^2 C)$ *bitműveletet* használó algoritmust. (Mi itt az eredeti, Lenstra, Lenstra és Lovász [225] cikkében bevezetett definícióhoz ragaszkodtunk.) Egész együtthatós polinomok felbontására Schönhage $\widetilde{O}(n^6 + n^4 \lg^2 l)$ bit-

műveletet igénylő módszerét említjük (itt l az együtthatók hossza).

A racionális együtthatós polinomok felbontása mellett a rácsredukciót sikerrel alkalmazzák egy sor más probléma megoldására is: bizonyos hátizsák-alapú kriptorendszerek és lineáris kongruenciákon alapuló véletlenszám-generátorok feltörésére, szimultán diofantikus approximációra, valós számok közötti egész együtthatós lineáris függések felderítésére – ez a feladat fontos szerephez jut a matematikai azonosságokat kereső kísérletekben. Ezekről és több más kapcsolódó eredményről a [128] könyvben találhatunk jó áttekintést.

További izgalmas – Magyarországon is sikeresen művelt – alkalmazási terület a diofantikus egyenletek numerikus megoldása. Erről a témakörről magyar nyelven Pethő Attila [277] akadémiai doktori értekezésében, angolul pedig Smart [317] és Gaál István [124] könyvében olvashatunk. A legrövidebb rácsvektor keresésének nehézségét Ajtai igazolta a [13] cikkben.

Végül érdemes megjegyezni, hogy a rácsredukciót alkalmazó polinomidejű módszerek gyakorlati megvalósításai máig sem versenyképesek a rossz esetben exponenciális bonyolultságú Berlekamp–Zassenhaus-algoritmus megvalósításaival szemben. Ugyanakkor maga a bázisredukció a gyakorlatban igen hatékonyan működik: meglepő módon az elméletileg bizonyított sebességnél nagyságrendekkel gyorsabb. A fent felsorolt alkalmazási területeken bizonyos feladatok megoldására nem is rendelkezünk más használható eljárással.

A szerzők munkáját részben támogatták a T042481 és T042706 számú OTKA-szerződések.

19. Automaták és formális nyelvek

A fordítóprogramok tervezésében és megvalósításában az automaták és formális nyelvek elmélete fontos szerepet játszik. Az első alfejezetben értelmezzük a formális nyelv és a nyelvtan fogalmát. A Chomsky-féle felosztás alapján tárgyaljuk a különböző nyelv- és nyelvtantípusokat. A második alfejezetben részletesen foglalkozunk a véges automatákkal és az általuk felismert nyelvekkel, míg a harmadikban a veremautomatákkal és általuk felismert nyelvekkel. Végül felsorolunk néhány könyvet a gazdag szakirodalomból.

19.1. Nyelvek és nyelvtanok

Tetszőleges szimbólumok (jelek) nem üres, véges halmazát ***ábécének*** nevezzük. Az ábécé elemeit ***betűknek*** nevezzük, de sokszor használjuk a jel és szimbólum neveket is. Ábécé például a $\Sigma = \{a, b, c, d, 0, 1, \sigma\}$, amelynek betűi $a, b, c, d, 0, 1$ és σ.

Az ábécé betűiből szavakat képezünk. Ha $a_1, a_2, \ldots, a_n \in \Sigma, n \geq 0$, akkor $a_1 a_2 \ldots a_n$ a Σ ábécé betűiből képzett ***szó*** (az a_i-k nem feltétlenül különbözők). A szót alkotó betűk száma, multiplicitással számolva, a szó ***hossza***. Ha $w = a_1 a_2 \ldots a_n$, akkor a w szó hossza $|w| = n$. Ha $n = 0$, akkor a szó egyetlen betűt sem tartalmaz, és ***üres szónak*** nevezzük. Jelölése: ε (sok könyvben λ). A Σ betűiből képezhető szavak halmazának jelölésére a Σ^* jelet használjuk:

$$\Sigma^* = \{a_1 a_2 \ldots a_n \mid a_1, a_2, \ldots, a_n \in \Sigma,\ n \geq 0\} .$$

A nem üres szavak halmaza $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. Az n hosszúságú szavak halmazát Σ^n-nel jelöljük, és természetesen $\Sigma^0 = \{\varepsilon\}$. Ekkor

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \cdots \cup \Sigma^n \cup \cdots \qquad \text{és} \qquad \Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \cdots \cup \Sigma^n \cup \cdots .$$

Az $u = a_1 a_2 \ldots a_m$ és $v = b_1 b_2 \ldots b_n$ szavak egyenlők (azaz $u = v$), ha $m = n$ és $a_i = b_i$, $i = 1, 2, \ldots, n$.

A Σ^*-on bevezetünk egy bináris műveletet, a konkatenációt. Az $u = a_1 a_2 \ldots a_m$ és $v = b_1 b_2 \ldots b_n$ szavak ***konkatenációján*** (illesztésén, szorzatán, összefűzésén) az $uv = a_1 a_2 \ldots a_m b_1 b_2 \ldots b_n$ szót értjük. Nyilvánvaló, hogy $|uv| = |u| + |v|$. Ez a művelet asszociatív, de nem kommutatív. Van egységeleme, az ε, mivel $\varepsilon u = u\varepsilon = u$ tetszőleges $u \in \Sigma^*$ szóra. Tehát Σ^* ezzel a konkatenáció művelettel monoidot (egységelemes félcsoportot) képez.

Bevezetjük a szavak hatványozását. Ha $u \in \Sigma^*$, akkor $u^0 = \varepsilon$, továbbá ha $n \geq 1$, akkor $u^n = u^{n-1} u$.

Az $u = a_1 a_2 \dots a_n$ szó ***tükörképe*** $u^{-1} = a_n a_{n-1} \dots a_1$. Az u tükörképének jelölésére néha használják még a következőket is: u^R, $\tilde{u}$. Nyilvánvaló, hogy $\left(u^{-1}\right)^{-1} = u$ és $(uv)^{-1} = v^{-1}u^{-1}$.

Azt mondjuk, hogy v ***kezdőszelete*** (prefixe) az u szónak, ha létezik a z szó úgy, hogy $u = vz$. Ha $z \neq \varepsilon$, akkor v valódi kezdőszelete u-nak. Hasonlóképpen v ***végszelete*** (szuffixe) az u szónak, ha létezik az x szó úgy, hogy $u = xv$, és analóg módon értelmezhető a valódi végszelet is. A v szó ***részszava*** az u szónak, ha léteznek a p és q szavak úgy, hogy $u = pvq$. Ha $pq \neq \varepsilon$, akkor v ***valódi részszava*** u-nak. A kezdő- és végszeletek egyben részszavak is.

A Σ^* tetszőleges L részhalmazát a Σ ábécé feletti ***nyelvnek*** nevezzük. Mivel a szavaknak nem tulajdonítunk értelmet, gyakran ***formális nyelvről*** beszélünk, hogy megkülönböztessük a természetes és a mesterséges nyelvektől, amelyekben a szavakhoz valamilyen értelem is kapcsolódik. Megjegyezzük, hogy $\emptyset$ az üres nyelv, míg $\{\varepsilon\}$ az üres szóból álló nyelv.

19.1.1. Műveletek nyelvekkel

Ha L, L_1, L_2 egy-egy Σ feletti nyelv, akkor értelmezzük a következő műveleteket:

- *egyesítés*
$$L_1 \cup L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ vagy } u \in L_2\} ,$$
- *metszet*
$$L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ és } u \in L_2\} ,$$
- *különbség*
$$L_1 \setminus L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ és } u \notin L_2\} ,$$
- *komplementum (komplemens* vagy *komplementer nyelv)*
$$\overline{L} = \Sigma^* \setminus L ,$$
- *szorzat*
$$L_1 L_2 = \{uv \mid u \in L_1, v \in L_2\} ,$$
- *nyelv hatványa*
$$L^0 = \{\varepsilon\}, \qquad L^n = L^{n-1}L, \text{ ha } n \geq 1 ,$$
- *nyelv iteráltja*
$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L \cup L^2 \cup \dots \cup L^i \cup \dots ,$$
- *tükrözés*
$$L^{-1} = \{u^{-1} \mid u \in L\}$$

Szoktuk még használni az L^+ jelölést is:

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L \cup L^2 \cup \dots \cup L^i \cup \dots .$$

Az egyesítés, szorzás, iteráció műveleteket ***reguláris műveleteknek*** nevezzük.

19.1.2. Nyelvek megadása

Nyelveket többféleképpen adhatunk meg. Például:

1) felsoroljuk a szavait,

2) megadunk egy tulajdonságot, amellyel a nyelv minden szava rendelkezik, de más szavak nem,

3) nyelvtan segítségével.

Nyelvek megadása elemeik felsorolásával

Nyelvek például:

$L_1 = \{\varepsilon, 0, 1\}$,

$L_2 = \{a, aa, aaa, ab, ba, aba\}$.

Habár végtelen halmazokat nem tudunk felsorolni, a végtelen nyelvek is megadhatók felsorolással abban az esetben, ha az első néhány szó megadásából kiderül, hogyan kell a felsorolást folytatni. Ilyen például a következő nyelv:

$L_3 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \ldots\}$.

Nyelvek megadása tulajdonság segítségével

Nyelvek a következő halmazok is:

$L_4 = \{a^n b^n \mid n = 0, 1, 2, \ldots\}$,

$L_5 = \{uu^{-1} \mid u \in \Sigma^*\}$,

$L_6 = \{u \in \{a, b\}^* \mid n_a(u) = n_b(u)\}$,

ahol $n_a(u)$ az u szóban levő a betűk számát, $n_b(u)$ pedig a b betűk számát jelöli.

Nyelvek megadása nyelvtannal

Értelmezzük a ***generatív nyelvtan*** vagy röviden ***nyelvtan*** (grammatika) fogalmát.

19.1. definíció. ***Generatív nyelvtan****nak nevezzük a* $G = (N, T, P, S)$ *rendezett négyest, ahol*

- N *a* ***változók*** *(vagy nemterminális jelek) ábécéje,*
- T *a* ***terminális jelek*** *ábécéje, ahol* $N \cap T = \emptyset$,
- $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ *véges halmaz, vagyis* P *az* (u, v) *alakú* ***helyettesítési szabályok*** *véges halmaza, ahol* $u, v \in (N \cup T)^*$, *és* u *tartalmaz legalább egy nemterminális jelet,*
- $S \in N$ *a nyelvtan* ***kezdőszimbóluma.***

Megjegyzés. Az (u, v) jelölés helyett gyakran használjuk az $u \to v$ jelölést, amely szemléletesebben utal a helyettesítésre (amint azt később látni fogjuk).

Az $u \to v$, azaz (u, v) szabályban u-t a szabály bal, v-t pedig a jobb oldalának nevezzük. Amennyiben a nyelvtanban több olyan szabály van, amelyeknek a bal oldala azonos, akkor ezeket egyszerűbben is írhatjuk:

$$u \to v_1,\ u \to v_2, \ldots, u \to v_r \quad \text{helyett} \quad u \to v_1 \mid v_2 \mid \ldots \mid v_r\ .$$

Értelmezzük az $(N \cup T)^*$ halmazban a ***közvetlen levezetés*** relációt:

$$u \Longrightarrow v, \qquad \text{ha} \quad u = p_1 p p_2, \quad v = p_1 q p_2 \quad \text{és} \quad (p, q) \in P\ .$$

Tulajdonképpen u-ban a p részszó valamely előfordulását q-val helyettesítjük, és eredményül v-t kapjuk. A közvetlen levezetésre még szokás a $\vdash$ vagy a $\models$ jeleket is használni.

Ha hangsúlyozni szeretnénk a G nyelvtant, amelynek szabályait használjuk, akkor a $\Longrightarrow$ jelölés helyett a $\underset{G}{\Longrightarrow}$ jelölést használjuk. A $\Longrightarrow$ reláció reflexív és tranzitív lezártját $\overset{*}{\Longrightarrow}$, míg a tranzitív lezártját $\overset{+}{\Longrightarrow}$ jelöli. A $\overset{*}{\Longrightarrow}$ reláció neve ***levezetés***.

A reflexív és tranzitív lezárt definíciójából következik, hogy $u \overset{*}{\Longrightarrow} v$, ha léteznek a $w_0, w_1, \ldots, w_n \in (N \cup T)^*$, $n \geq 0$ szavak, és $u = w_0,\ w_0 \Longrightarrow w_1, w_1 \Longrightarrow w_2, \ldots, w_{n-1} \Longrightarrow w_n,\ w_n = v$. Ezt röviden így is írhatjuk: $u = w_0 \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \ldots \Longrightarrow w_{n-1} \Longrightarrow$

$w_n = v$. Ha $n = 0$, akkor $u = v$. Hasonlóképpen értelmezhető az $u \overset{+}{\Longrightarrow} v$ reláció, azzal a különbséggel, hogy itt $n \geq 1$, tehát elvégzünk legalább egy helyettesítést

19.2. definíció. *A $G = (N, T, P, S)$ nyelvtan által **generált nyelv**:*

$$L(G) = \{u \in T^* \mid S \overset{*}{\Longrightarrow} u\} .$$

Tehát $L(G)$ mindazokat a T ábécé betűiből képzett szavakat tartalmazza, amelyek az S kezdőszimbólumból levezethetők P szabályainak a segítségével.

19.1. példa. Legyen $G = (N, T, P, S)$, ahol
$N = \{S\}$,
$T = \{a, b\}$,
$P = \{S \to aSb,\ S \to ab\}$.
Könnyű belátni, hogy ekkor $L(G) = \{a^n b^n \mid n \geq 1\}$, mivel

$$S \underset{G}{\Longrightarrow} aSb \underset{G}{\Longrightarrow} a^2Sb^2 \underset{G}{\Longrightarrow} \cdots \underset{G}{\Longrightarrow} a^{n-1}Sb^{n-1} \underset{G}{\Longrightarrow} a^n b^n ,$$

ahol az utolsó előtti helyettesítésig mindig az első helyettesítési szabályt ($S \to aSb$) használtuk, míg az utolsónál az $S \to ab$ szabályt. Ezt a levezetést röviden így is írhatjuk: $S \overset{*}{\underset{G}{\Longrightarrow}} a^n b^n$. Tehát $a^n b^n$ tetszőleges n-re levezethető S-ből, és más szót nem lehet levezetni S-ből.

19.3. definíció. *Azt mondjuk, hogy a G_1 és G_2 nyelvtanok **ekvivalensek,** és ezt $G_1 \cong G_2$-vel jelöljük, ha $L(G_1) = L(G_2)$.*

19.2. példa. A következő két nyelvtan ekvivalens, mivel mindegyik az $\{a^n b^n c^n \mid n \geq 1\}$ nyelvet generálja.
$G_1 = (N_1, T, P_1, S_1)$, ahol
$N_1 = \{S_1, X, Y\}$, $T = \{a, b, c\}$,
$P_1 = \{S_1 \to abc,\ S_1 \to aXbc,\ Xb \to bX,\ Xc \to Ybcc,\ bY \to Yb,\ aY \to aaX,\ aY \to aa\}$.
$G_2 = (N_2, T, P_2, S_2)$, ahol
$N_2 = \{S_2, A, B, C\}$,
$P_2 = \{S_2 \to aS_2BC,\ S_2 \to aBC,\ CB \to BC,\ aB \to ab,\ bB \to bb,\ bC \to bc,\ cC \to cc\}$.
Először teljes indukcióval megmutatjuk, hogy $n \geq 2$-re $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1}Yb^n c^n$. Ha $n = 2$, akkor

$$S_1 \underset{G_1}{\Longrightarrow} aXbc \underset{G_1}{\Longrightarrow} abXc \underset{G_1}{\Longrightarrow} abYbcc \underset{G_1}{\Longrightarrow} aYb^2c^2 .$$

Feltételezzük, hogy $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-2}Yb^{n-1}c^{n-1}$. Alkalmazzuk az $aY \to aaX$ szabályt, majd $(n-1)$-szer az $Xb \to bX$ szabályt, egyszer az $Xc \to Ybcc$ szabályt, utána pedig szintén $(n-1)$-szer a $bY \to Yb$ szabályt. Tehát

$$S_1 \underset{G_1}{\Longrightarrow} a^{n-2}Yb^{n-1}c^{n-1} \underset{G_1}{\Longrightarrow} a^{n-1}Xb^{n-1}c^{n-1} \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1}b^{n-1}Xc^{n-1} \underset{G_1}{\Longrightarrow} a^{n-1}b^{n-1}Ybc^n \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1}Yb^n c^n .$$

Ha most alkalmazzuk az $aY \to aa$ szabályt, azt kapjuk, hogy $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^n b^n c^n$, $n \geq 2$-re, de $S_1 \underset{G_1}{\Longrightarrow} abc$ az $S_1 \to abc$ szabály alapján, tehát $a^n b^n c^n \in L(G_1)$ tetszőleges $n \geq 1$-re. Be kell még bizonyítanunk, hogy a szabályok alkalmazásával más szót nem lehet generálni, csak $a^n b^n c^n$ alakút. Ezt könnyű belátni, hisz eredményes levezetés (amely csak terminális betűket tartalmazó szóban végződik) csak a fenti módon lehetséges.

Hasonlóképpen $n \geq 2$-re

$$S_2 \underset{G_2}{\Longrightarrow} aS_2BC \overset{*}{\underset{G_2}{\Longrightarrow}} a^{n-1}S_2(BC)^{n-1} \underset{G_2}{\Longrightarrow} a^n(BC)^n \overset{*}{\underset{G_2}{\Longrightarrow}} a^nB^nC^n$$

$$\underset{G_2}{\Longrightarrow} a^nbB^{n-1}C^n \overset{*}{\underset{G_2}{\Longrightarrow}} a^nb^nC^n \underset{G_2}{\Longrightarrow} a^nb^ncC^{n-1} \overset{*}{\underset{G_2}{\Longrightarrow}} a^nb^nc^n \ .$$

Itt sorrendben a következő szabályokat alkalmaztuk: $S_2 \to aS_2BC$ ($n-1$-szer), $S_2 \to aBC$, $CB \to BC$ ($n-1$-szer), $aB \to ab$, $bB \to bb$ ($n-1$-szer), $bC \to bc$, $cC \to cc$ ($n-1$-szer). Ugyanakkor $S_2 \underset{G_2}{\Longrightarrow} aBC \underset{G_2}{\Longrightarrow} abC \underset{G_2}{\Longrightarrow} abc$, tehát $S_2 \overset{*}{\underset{G_2}{\Longrightarrow}} a^nb^nc^n$, $n \geq 1$. Itt is könnyű belátni, hogy más szavakat nem lehet generálni a G_2-ben.

A következő két nyelvtan:

$G_3 = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to \varepsilon\}, S)$ és

$G_4 = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to ab\}, S)$

nem ekvivalens, mert $L(G_3) \setminus \{\varepsilon\} = L(G_4)$.

19.4. tétel. *Létezik olyan formális nyelv, amelyet nem lehet nyelvtannal megadni.*

Bizonyítás. A bizonyításhoz a nyelvtanokat a $\{0, 1\}$ ábécé feletti szavakként kódoljuk. Egy adott $G = (N, T, P, S)$ nyelvtan esetében legyen $N = \{S_1, S_2, \ldots, S_n\}$, $T = \{a_1, a_2, \ldots, a_m\}$ és $S = S_1$. A kódolás a következő:

S_i kódja $10\underbrace{11\ldots11}_{i\text{-szer}}01$, $\qquad$ a_i kódja $100\underbrace{11\ldots11}_{i\text{-szer}}001$.

A szimbólumok kódját a kódolásban 000 választja el, a nyíl kódja 0000, a szabályokat pedig 00000 jelsorozattal választjuk el.

Nyilván elég csak a szabályokat kódolni. Példaként vegyük a következő nyelvtant:

$G = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to ab\}, S)$.

S kódja 10101, a kódja 1001001, b kódja 10011001. A nyelvtan kódja:

$$\underbrace{10101}\,0000\,\underbrace{1001001}\,000\,\underbrace{10101}\,000\,\underbrace{10011001}\,00000\,\underbrace{10101}\,0000\,\underbrace{1001001}\,000$$

$$\underbrace{10011001}\ .$$

Ebből a kódolásból következik, hogy a T terminális ábécével rendelkező nyelvtanok felsorolhatók[1] a következőképpen: $G_1, G_2, \ldots, G_k, \ldots$, és így ezen nyelvtanok halmazának számossága megszámlálhatóan végtelen.

Tekintsük most a T ábécé feletti összes nyelvek halmazát: $\mathcal{L}_T = \{L \mid L \subseteq T^*\}$, azaz $\mathcal{L}_T = \mathcal{P}(T^*)$. A T^* halmaz megszámlálhatóan végtelen, hisz szavai sorrendbe írhatók. Legyen ez a sorrend $s_0, s_1, s_2, \ldots$, ahol legyen $s_0 = \varepsilon$. Ekkor minden $L \in \mathcal{L}_T$ nyelvnek megfeleltethetünk egy $b_0, b_1, b_2, \ldots$ végtelen bináris sorozatot a következőképpen:

$$b_i = \begin{cases} 1, & \text{ha } s_i \in L \\ 0, & \text{ha } s_i \notin L \end{cases} \qquad i = 0, 1, 2, \ldots \ .$$

[1] Tegyük fel, hogy az $\{0, 1\}$ ábécén adott egy $<$ lineáris rendezés, mondjuk $0 < 1$. Ezek után a nyelvtanokat kódoló szavak felsorolhatók úgy, hogy a szavakat előbb a hosszúságuk szerint rendezzük, majd azon belül ábécésorrendbe, az ábécé betűi közötti rendezés alapján. De lehet lexikografikus sorrend is, amely azt jelenti, hogy $u < v$ (u előbb van, mint v), ha u valódi kezdőszelete v-nek, vagy létezik az $u = xay$ és $v = xby'$ felbontás, ahol x, y, y' részszavak, a és b betűk, és $a < b$.

Könnyű belátni, hogy az így keletkezett bináris sorozatok halmaza nem megszámlálhatóan végtelen, hisz minden sorozat tekinthető úgy, mint egy 1-nél kisebb pozitív valós szám kettes számrendszerbeli ábrázolása (a tizedesvesszőt az első számjegy elé képzeljük). Fordítva is igaz, hogy minden 1-nél kisebb pozitív szám kettes számrendszerbeli ábrázolása, ha a tizedesvessző utáni részt vesszük, megfelel egy ilyen sorozatnak. Így a végtelen bináris sorozatok halmazának számossága megegyezik a $[0, 1]$ intervallum kontinuum számosságával. Következésképpen az $\mathcal{L}_T$ halmaz is kontinuum számosságú. Rendeljük most hozzá minden T terminális ábécével rendelkező nyelvtanhoz az általa generált T feletti nyelvet. Mivel a nyelvtanok számossága megszámlálhatóan végtelen, lesz olyan $\mathcal{L}_T$-beli nyelv, amelyhez nem tartozik nyelvtan, más szóval, amelyet nem lehet nyelvtannal generálni. ■

19.1.3. Chomsky-féle nyelvosztályok

Ha a helyettesítési szabályokra bizonyos megkötéseket teszünk, a következő négy nyelvtantípust különböztethetjük meg.

19.5. definíció. *A $G = (N, T, P, S)$ nyelvtanra a következő négy típust definiáljuk.*

*A G nyelvtan 0-típusú (**általános** vagy **mondatszerkezetű**), ha semmilyen megkötést nem teszünk a helyettesítési szabályaira.*

*A G nyelvtan 1-típusú **(környezetfüggő),** ha minden szabálya $\alpha A\gamma \to \alpha\beta\gamma$ alakú, ahol $A \in N$, $\alpha, \gamma \in (N \cup T)^*$, $\beta \in (N \cup T)^+$. Ezenkívül megengedhető az $S \to \varepsilon$ szabály is, ha S nem szerepel egyetlen szabály jobb oldalán sem.*

*A G nyelvtan 2-típusú **(környezetfüggetlen),** ha minden szabálya $A \to \beta$ alakú, ahol $A \in N$, $\beta \in (N \cup T)^+$. Ezenkívül megengedhető az $S \to \varepsilon$ szabály is, ha S nem szerepel egyetlen szabály jobb oldalán sem.*

*A G nyelvtan 3-típusú **(reguláris),** ha szabályai $A \to aB$ vagy $A \to a$ alakúak, ahol $a \in T$ és $A, B \in N$. Ezenkívül megengedhető az $S \to \varepsilon$ szabály is, ha S nem szerepel egyetlen szabály jobb oldalán sem.*

Ha G i-típusú nyelvtan, akkor az $L(G)$ nyelvet szintén i-típusúnak (általánosnak, környezetfüggőnek, környezetfüggetlennek, regulárisnak) nevezzük.

Ez a felosztás Noam Chomsky nevéhez fűződik.

Egy L nyelv i-típusú ($i = 0, 1, 2, 3$), ha létezik egy i-típusú G nyelvtan, amelyik az L nyelvet generálja, vagyis $L = L(G)$.

Jelölje $\mathcal{L}_i$ ($i = 0, 1, 2, 3$) az i-típusú nyelvek osztályát. Ekkor bizonyítható, hogy

$$\mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3 \ .$$

A nyelvtantípusok fenti értelmezése alapján a tartalmazás ($\supseteq$) nyilvánvaló, de a szigorú tartalmazás ($\supset$) bizonyításra szorul.

19.3. példa. Adunk egy-egy példát a környezetfüggő, környezetfüggetlen és reguláris nyelvtanokra.

Környezetfüggő nyelvtan. $G_1 = (N_1, T_1, P_1, S_1)$, ahol $N_1 = \{S_1, A, B, C\}$, $T_1 = \{a, 0, 1\}$.

P_1 elemei:

$$\begin{aligned} S_1 &\to ACA, \\ AC &\to AACA \mid ABa \mid AaB, \\ B &\to AB \mid A, \\ A &\to 0 \mid 1. \end{aligned}$$

Az $L(G_1)$ nyelv olyan uav szavakból áll, ahol $u, v \in \{0,1\}^*$ és $|u| \neq |v|$.

Környezetfüggetlen nyelvtan. $G_2 = (N_2, T_2, P_2, S)$, ahol $N_2 = \{S, A, B\}$, $T_2 = \{+, *, (,), a\}$.

P_2 elemei:

$S \rightarrow S + A \mid A,$
$A \rightarrow A * B \mid B,$
$B \rightarrow (S) \mid a.$

Az $L(G_2)$ olyan algebrai kifejezésekből áll, amelyek az a betűből a $+$ és $*$ műveleti jelekkel és a zárójelekkel „szabályosan” felépíthetők.

Reguláris nyelvtan. $G_3 = (N_3, T_3, P_3, S_3)$, ahol $N_3 = \{S_3, A, B\}$, $T_3 = \{a, b\}$.

P_3 elemei:

$S_3 \rightarrow aA$
$A \rightarrow aB \mid a$
$B \rightarrow aB \mid bB \mid a \mid b.$

Az $L(G_3)$ nyelv olyan, a és b betűkből képezhető szavakból áll, amelyek legalább két a-val kezdődnek.

Könnyű bebizonyítani, hogy minden véges nyelv reguláris. A nyelvtan szabályait úgy adjuk meg, hogy generálják az összes szót. Például, ha $u = a_1 a_2 \ldots a_n$ eleme a nyelvnek, akkor bevezetjük a következő szabályokat: $S \rightarrow a_1 A_1$, $A_1 \rightarrow a_2 A_2$, $\ldots A_{n-2} \rightarrow a_{n-1} A_{n-1}$, $A_{n-1} \rightarrow a_n$, ahol S a nyelvtan kezdőszimbóluma, $A_1, \ldots, A_{n-1}$ pedig különböző nemterminálisok. Ezt a véges nyelv minden szavára megtesszük, oly módon, hogy különböző szavakhoz, az S kivételével, különböző nemterminálisokat használunk. Ha az üres szó is eleme a nyelvnek, akkor azt természetesen az $S \rightarrow \varepsilon$ szabállyal generáljuk.

Az üres halmaz szintén reguláris nyelv, hisz például a $G = (\{S\}, \{a\}, \{S \rightarrow aS\}, S)$ reguláris nyelvtan az üres halmazt generálja.

Átnevezések kiküszöbölése

Egy $A \rightarrow B$ alakú szabályt, ahol $A, B \in N$, ***átnevezésnek*** nevezzük. Az átnevezéseket ki lehet küszöbölni a G nyelvtanból úgy, hogy az újonnan kapott nyelvtan ugyanolyan típusú legyen, mint G, és ugyanazt a nyelvet ismerje fel, mint G.

Legyen $G = (N, T, P, S)$ átnevezéseket tartalmazó nyelvtan. Definiáljuk a vele ekvivalens, de átnevezéseket nem tartalmazó $G' = (N, T, P', S)$ nyelvtant. A következő algoritmus megkonstruálja az új nyelvtan szabályait.

ÁTNEVEZÉS-KIZÁRÁS(G,G')

1 valahányszor $A \rightarrow B$ és $B \rightarrow C$ átnevezések P-ben vannak, mindannyiszor vegyük fel P-be az $A \rightarrow C$ átnevezést is, mindaddig, amíg P bővíthető,
2 valahányszor $A \rightarrow B$ átnevezés és a $B \rightarrow \alpha$ ($\alpha \notin N$) szabály P-ben van, mindannyiszor vegyük fel P-be az $A \rightarrow \alpha$ szabályt is,
3 legyen P' azon P-beli szabályok halmaza, amelyek nem átnevezések.

Könnyen belátható, hogy G és G' ekvivalensek. Továbbá, ha G $i \in \{0,1,2,3\}$ típusú, akkor G' is i típusú lesz.

19.4. példa. Alkalmazzuk az előbbi algoritmust a $G = (\{S, A, B, C\}, \{a, b\}, P, S)$ nyelvtanra, ahol a P elemei:

$S \to A$, $A \to B$, $B \to C$, $C \to B$, $D \to C$,
$S \to B$, $A \to D$, $C \to Aa$,
$A \to aB$,
$A \to b$.

Az algoritmus első lépése alapján a következő új átnevezések kerülnek be a szabályok közé:

$S \to D$ ($S \to A$ és $A \to D$ miatt),
$S \to C$ ($S \to B$ és $B \to C$ miatt),
$A \to C$ ($A \to B$ és $B \to C$ miatt),
$B \to B$ ($B \to C$ és $C \to B$ miatt),
$C \to C$ ($C \to B$ és $B \to C$ miatt),
$D \to B$ ($D \to C$ és $C \to B$ miatt).

Az algoritmus második lépése alkalmazásakor csak azok az átnevezések jöhetnek számításba, amelyeknek a jobb oldala vagy A vagy C, mivel csak az $A \to aB$, $A \to b$ és $C \to Aa$ szabályok alkalmazhatók (a többi szabály mind átnevezés). A következő új szabályok jelennek meg:

$S \to aB$ ($S \to A$ és $A \to aB$ miatt),
$S \to b$ ($S \to A$ és $A \to b$ miatt),
$S \to Aa$ ($S \to C$ és $C \to Aa$ miatt),
$A \to Aa$ ($A \to C$ és $C \to Aa$ miatt),
$B \to Aa$ ($B \to C$ és $C \to Aa$ miatt).

Az új $G' = (\{S, A, B, C\}, \{a, b\}, P', S)$ nyelvtan szabályai:

$S \to b$, $A \to b$, $B \to Aa$, $C \to Aa$,
$S \to aB$, $A \to aB$,
$S \to Aa$ $A \to Aa$,

Normálalakú nyelvtanok

Ha egy nyelvtan szabályainak bal oldalán terminális betűk nem szerepelnek, akkor a nyelvtant *normálalakúnak* mondjuk.

Szükségünk lesz a következő fogalmakra. Adott Σ_1 és Σ_2 ábécékre *homomorfizmusnak* nevezzük a $h : \Sigma_1^* \to \Sigma_2^*$ függvényt, ha $h(u_1u_2) = h(u_1)h(u_2)$, $\forall u_1, u_2 \in \Sigma_1^*$. Könnyű belátni, hogy tetszőleges $u = a_1a_2 \dots a_n \in \Sigma_1^*$ esetében a $h(u)$ értéket egyértelműen meghatározza h-nak a Σ_1-re való megszorítása, ugyanis $h(u) = h(a_1)h(a_2) \dots h(a_n)$.

Ha a h függvény még bijektív is, akkor *izomorfizmusról* beszélünk.

19.6. tétel. *Tetszőleges nyelvtanhoz megkonstruálhatunk egy vele ekvivalens, azonos típusú nyelvtant, amely normálalakú.*

Bizonyítás.

A 2- és 3-típusú nyelvtanok szabályai a bal oldalon csak egy-egy nemterminális szimbólumot tartalmaznak, tehát eleve normálalakúak.

A bizonyítást csupán a 0- és 1-típusú nyelvtanokra kell elvégezni. Legyen az eredeti nyelvtan $G = (N, T, P, S)$ és definiáljuk a $G' = (N', T, P', S)$ normálalakú nyelvtant a következőképpen.

Legyenek $a_1, a_2, \dots, a_k$ azok a terminális szimbólumok, amelyek szerepelnek a szabályok bal oldalán. Ekkor vezessük be az $A_1, A_2, \dots, A_k$ új nemterminálisokat. Használjuk a következő jelöléseket: $T_1 = \{a_1, a_2, \dots, a_k\}$, $T_2 = T \setminus T_1$, $N_1 = \{A_1, A_2, \dots, A_k\}$ és $N' = N \cup N_1$.

Használjuk a $h : N \cup T \longrightarrow N' \cup T_2$ izomorfizmust, ahol:

$$h(a_i) = A_i, \quad \text{ha } a_i \in T_1,$$

$h(X) = X, \quad$ ha $X \in N \cup T_2$

Most értelmezzük a P' szabályhalmazt:

$$P' = \left\{h(\alpha) \to h(\beta) \;\middle|\; (\alpha \to \beta) \in P\right\} \cup \left\{A_i \longrightarrow a_i \;\middle|\; i = 1, 2, \ldots, k\right\}$$

Ebben az esetben $\alpha \overset{*}{\underset{G}{\Longrightarrow}} \beta$ akkor és csakis akkor, ha $h(\alpha) \overset{*}{\underset{G'}{\Longrightarrow}} h(\beta)$.

Innen pedig azonnal következik a tételünk, hisz $S \overset{*}{\underset{G}{\Longrightarrow}} u \;\Leftrightarrow\; S = h(S) \overset{*}{\underset{G'}{\Longrightarrow}} h(u) = u$. ■

19.5. példa. Adott a $G = (\{S, D, E\}, \{a, b, c, d, e\}, P, S)$, ahol P szabályai:

$S \to aebc \mid aDbc$
$Db \to bD$
$Dc \to Ebccd$
$bE \to Eb$
$aE \to aaD \mid aae$

Bal oldali szabályokban az a, b, c terminálisok szerepelnek, ezért felvesszük a nemterminálisok közé az A, B, C új nemterminálisokat, és P'-be az $A \to a$, $B \to b$ és $C \to c$ szabályokat.

Az a, b, c terminálisokat minden szabályban helyettesítjük rendre az A, B, C nemterminálisokkal. Ekkor P' szabályai a következők:

$S \to AeBC \mid ADBC$
$DB \to BD$
$DC \to EBCCd$
$BE \to EB$
$AE \to AAD \mid AAe$
$A \to a$
$B \to b$
$C \to c$

Vizsgáljuk meg milyen szavakat generál ez a nyelvtan! $S \Longrightarrow AeBC \overset{*}{\Longrightarrow} aebc$ miatt $aebc \in L(G')$. $S \Longrightarrow ADBC \Longrightarrow ABDC \Longrightarrow ABEBCCd \Longrightarrow AEBBCCd \Longrightarrow AAeBBCCd \overset{*}{\Longrightarrow} aaebbccd$, tehát $aaebbccd \in L(G')$.

Feltételezzük, hogy $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1}$, $n \geq 2$. Ezt matematikai indukcióval bizonyíthatjuk.Már láttuk, hogy feltevésünk igaz ha $n = 2$. Folytassuk az előbbi levezetést: $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1} \Longrightarrow A^{n-2}AADB^nC(Cd)^{n-1} \overset{*}{\Longrightarrow} A^nB^nDC(Cd)^{n-1} \Longrightarrow A^nB^nEBCCd(Cd)^{n-1} \overset{*}{\Longrightarrow} A^nEB^{n+1}CCd(Cd)^{n-1} = A^nEB^{n+1}C(Cd)^n$, amit éppen bizonyítani kellett.

De $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1} \Longrightarrow A^{n-2}AAeB^nC(Cd)^{n-1} \overset{*}{\Longrightarrow} a^neb^nc(cd)^{n-1}$. Tehát $a^neb^nc(cd)^{n-1} \in L(G')$, $n \geq 1$. Ezeket a szavakat G-ben is hasonlóan le lehet vezetni.

19.1.4. Kiterjesztett nyelvtanok

Ebben a részben az 1-típusú, 2-típusú és a 3-típusú kiterjesztett nyelvtanokat mutatjuk be.

1-típusú kiterjesztett nyelvtan. Minden szabály $\alpha \to \beta$ alakú, ahol $|\alpha| \leq |\beta|$, kivéve esetleg az $S \to \varepsilon$ szabályt.

2-típusú kiterjesztett nyelvtan. Minden szabály $A \to \beta$ alakú, ahol $A \in N, \beta \in (N \cup T)^*$.

3-típusú kiterjesztett nyelvtan. Minden szabály $A \to uB$ vagy $A \to u$ alakú, ahol $A, B \in N, u \in T^*$.

19.7. tétel. *Tetszőleges kiterjesztett nyelvtanhoz megadható egy vele ekvivalens, ugyanolyan típusú nyelvtan.*

Bizonyítás. Jelöljük G_{ki}-vel a kiterjesztett nyelvtant és G-vel azt a nyelvtant, amelyet minden típusra külön értelmezünk, és amelyről meg fogjuk mutatni, hogy ekvivalens G_{ki}-vel.

1-típus. A G nyelvtan szabályait úgy kapjuk, hogy a G_{ki} nyelvtan $\alpha \to \beta$ szabályait, ahol $|\alpha| \le |\beta|$, átírjuk a G esetében megengedett $\gamma_1\delta\gamma_2 \to \gamma_1\gamma\gamma_2$ alakú szabályokra a következőképpen.

Legyen $X_1X_2\ldots X_m \to Y_1Y_2\ldots Y_n$ $(m \le n)$ a G_{ki} nyelvtan egy szabálya, amely nem megfelelő alakú. Vegyük fel G szabályhalmazába a következő szabályokat, ahol $A_1, A_2, \ldots, A_m$ új változók:

$$
\begin{array}{lcl}
X_1X_2\ldots X_m & \to & A_1X_2X_3\ldots X_m \\
A_1X_2\ldots X_m & \to & A_1A_2X_3\ldots X_m \\
 & \ldots & \\
A_1A_2\ldots A_{m-1}X_m & \to & A_1A_2\ldots A_{m-1}A_m \\
A_1A_2\ldots A_{m-1}A_m & \to & Y_1A_2\ldots A_{m-1}A_m \\
Y_1A_2\ldots A_{m-1}A_m & \to & Y_1Y_2\ldots A_{m-1}A_m \\
 & \ldots & \\
Y_1Y_2\ldots Y_{m-2}A_{m-1}A_m & \to & Y_1Y_2\ldots Y_{m-2}Y_{m-1}A_m \\
Y_1Y_2\ldots Y_{m-1}A_m & \to & Y_1Y_2\ldots Y_{m-1}Y_mY_{m+1}\ldots Y_n.
\end{array}
$$

Továbbá, G_{ki} minden megengedett, vagyis $\gamma_1\delta\gamma_2 \to \gamma_1\gamma\gamma_2$ alakú szabályát vegyük át változtatás nélkül G szabályhalmazába.

Ezek után a $L(G_{ki}) \subseteq L(G)$ tartalmazás abból következik, hogy a G_{ki} minden szabályának az alkalmazását egy, a belőle képzett G-beli szabályok alkalmazásával tudjuk szimulálni. Továbbá, mivel a G szabályai csak a felírt sorrendben alkalmazhatók, nem kapunk újabb szavakat, ezért $L(G) \subseteq L(G_{ki})$ is teljesül.

2-típus. Legyen $G_{ki} = (N, T, P, S)$. Ki kell küszöbölnünk az $A \to \varepsilon$ alakú szabályokat úgy, hogy csak $S \to \varepsilon$ maradhat, ha S nem szerepel szabály jobb oldalán. Ehhez felépítjük a következő halmazokat:

$U_0 = \{A \in N \mid (A \to \varepsilon) \in P\}$

$U_i = U_{i-1} \cup \{A \in N \mid (A \to w) \in P,\ w \in U_{i-1}^+\}$.

Mivel minden $i \ge 1$ esetében $U_{i-1} \subseteq U_i$ és $U_i \subseteq N$, továbbá N véges halmaz, léteznie kell egy olyan k-nak, amelyre $U_{k-1} = U_k$, és ezt a halmazt nevezzük el U-nak. Könnyű belátni, hogy egy A nemterminális akkor és csakis akkor eleme U-nak, ha $A \overset{*}{\Longrightarrow} \varepsilon$. (Mellesleg, $\varepsilon \in L(G_{ki})$, akkor és csakis akkor ha $S \in U$.)

G szabályait a következőképpen kapjuk G_{ki} szabályaiból. G_{ki} minden olyan $A \to \alpha$ szabálya esetében melyre $\alpha \ne \varepsilon$, vegyük fel a G szabályai közé ezt a szabályt és mellette azokat is, amelyeket úgy képezünk, hogy α-ból elhagyunk egy vagy több U-beli változót, de csak akkor, ha ezáltal a jobb oldal nem lesz ε. Nem nehéz belátni, hogy az így kapott G nyelvtan ugyanazt a nyelvet generálja, mint G_{ki}, kivéve az ε szót, amelyet nem tud generálni. Ezért, ha $\varepsilon \notin L(G_{ki})$, akkor a bizonyítást befejeztük. Ha viszont $\varepsilon \in L(G_{ki})$, akkor két esetet különböztetünk meg. Ha az S kezdőszimbólum nem szerepel egyetlen szabály jobb oldalán sem, akkor az $S \to \varepsilon$ szabály bevezetésével a G nyelvtan már az üres szót is generálni fogja. Ha viszont S szerepel valamelyik szabály jobb oldalán, akkor egy új kezdőszimbólum (S') bevezetésével ε is generálható lesz, ha bevesszük a G szabályai közé az $S' \to S$ és $S' \to \varepsilon$ szabályokat.

3-típus. Először alkalmazzuk G_{ki}-re a 2-típus esetében használt eljárást az $A \to \varepsilon$ alakú szabályok kiküszöbölésére. A kapott nyelvtanból kiküszöböljük az átnevezéseket az ÁTNEVEZÉS-KIZÁRÁS algoritmus segítségével (899. oldal).

Az így kapott nyelvtan minden $A \to a_1 a_2 \dots a_n B$ szabály esetén, ahol $B \in N \cup \{\varepsilon\}$, vegyük fel G szabályai közé a következő szabályokat:

$$\begin{aligned} A &\to a_1 A_1, \\ A_1 &\to a_2 A_2, \\ &\dots \\ A_{n-1} &\to a_n B, \end{aligned}$$

ahol $A_1, A_2, \dots, A_{n-1}$ új változók. Könnyen igazolható, hogy az így megkonstruált G nyelvtan ekvivalens G_{ki}-vel. ■

19.6. példa. Adva van a következő 1-típusú kiterjesztett nyelvtan: $G_{ki} = (N, T, P, S)$, ahol $N = \{S, B, C\}$, $T = \{a, b, c\}$ és P a következő szabályokból áll:

$$\begin{aligned} S &\to aSBC \mid aBC & CB &\to BC \\ aB &\to ab & bB &\to bb \\ bC &\to bc & cC &\to cc\,. \end{aligned}$$

Az egyetlen szabály, amely nem környezetfüggő, az a $CB \to BC$. Ehelyett bevezetjük a következőket, a bizonyításban adott módszer alapján:

$$\begin{aligned} CB &\to AB \\ AB &\to AD \\ AD &\to BD \\ BD &\to BC \end{aligned}$$

Így az új nyelvtan, amely most már környezetfüggő: $G = (\{S, A, B, C, D\}, \{a, b, c\}, P', S)$, ahol P' elemei:

$$\begin{aligned} S &\to aSBC \mid aBC & & \\ CB &\to AB & aB &\to ab \\ AB &\to AD & bB &\to bb \\ AD &\to BD & bC &\to bc \\ BD &\to BC & cC &\to cc. \end{aligned}$$

Igazolni lehet, hogy $L(G_{ki}) = L(G) = \{a^n b^n c^n \mid n \geq 1\}$.

19.7. példa. Legyen $G_{ki} = (\{S, B, C\}, \{a, b, c\}, P, S)$ 2-típusú kiterjesztett nyelvtan, ahol P elemei:

$$\begin{aligned} S &\to aSc \mid B \\ B &\to bB \mid C \\ C &\to Cc \mid \varepsilon. \end{aligned}$$

Ekkor $U_0 = \{C\}$, $U_1 = \{B, C\}$, $U_3 = \{S, B, C\} = U$. Az új nyelvtan szabályai:

$$\begin{aligned} S &\to aSc \mid ac \mid B \\ B &\to bB \mid b \mid C \\ C &\to Cc \mid c. \end{aligned}$$

Mivel az eredeti nyelvtan generálja az üres szót is, és S szerepel szabály jobb oldalán, új kezdőszimbólumot kell bevezetnünk és még két szabályt: $S' \to S, S' \to \varepsilon$. Tehát az eredetivel ekvivalens környezetfüggetlen nyelvtan:

$G = (\{S', S, B, C\}, \{a, b, c\}, P', S')$ és a szabályok:

$$\begin{aligned} S' &\to S \mid \varepsilon \\ S &\to aSc \mid ac \mid B \\ B &\to bB \mid b \mid C \\ C &\to Cc \mid c. \end{aligned}$$

Mindkét nyelvtan az $\{a^m b^n c^p \mid p \geq m \geq 0, n \geq 0\}$ nyelvet generálja.

19.8. példa. Legyen $G_{ki} = (\{S, A, B\}, \{a, b\}, P, S)$ a vizsgálandó 3-típusú kiterjesztett nyelvtan, ahol P:

$S \rightarrow abA$
$A \rightarrow bB$
$B \rightarrow S \mid \varepsilon.$

Először küszöböljük ki a $B \rightarrow \varepsilon$ szabályt. Mivel $U_0 = U = \{B\}$, a szabályok a következők lesznek:

$S \rightarrow abA$
$A \rightarrow bB \mid b$
$B \rightarrow S.$

Ez utóbbi szabályt (amely átnevezés) ki lehet küszöbölni, bevezetve helyette a $B \rightarrow abA$ szabályt. Hátra van még az $S \rightarrow abA$ és $B \rightarrow abA$ szabályok jobb oldalának a feldarabolása. Mivel mindkét szabály jobb oldala ugyanaz, elég egy új változót bevezetnünk, és az $S \rightarrow abA$ helyett az $S \rightarrow aC$ és $C \rightarrow bA$ szabályokat használni. A $B \rightarrow abA$ helyett ekkor elég a $B \rightarrow aC$ szabályt venni. Az új nyelvtan: $G = (\{S, A, B, C\}, \{a, b\}, P', S)$, ahol P':

$S \rightarrow aC$
$A \rightarrow bB \mid b$
$B \rightarrow aC$
$C \rightarrow bA.$

Könnyű bizonyítani, hogy $L(G_{ki}) = L(G) = \{(abb)^n \mid n \geq 1\}$.

19.1.5. A Chomsky-féle nyelvosztályok zártsági tulajdonságai

Bebizonyítjuk a következő tételt, amely szerint a Chomsky-nyelvosztályok mindegyike zárt a reguláris műveletekre nézve, azaz két i-típusú nyelv egyesítése és szorzata is i-típusú, i-típusú nyelv iteráltja is i-típusú ($i = 0, 1, 2, 3$).

19.8. tétel. *Az $\mathcal{L}_i$ ($i = 0, 1, 2, 3$) nyelvek osztálya zárt a reguláris műveletekre nézve.*

Bizonyítás. Kiterjesztett nyelvtanok segítségével végezzük a bizonyítást. Legyenek $G_1 = (N_1, T_1, P_1, S_1)$ és $G_2 = (N_2, T_2, P_2, S_2)$ i-típusú kiterjesztett nyelvtanok. Feltételezzük, hogy $N_1 \cap N_2 = \emptyset$.

Egyesítés. Legyen $G_\cup = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}, S)$.

Könnyű igazolni, hogy $L(G_\cup) = L(G_1) \cup L(G_2)$. Ha $i = 0, 2, 3$, akkor abból, hogy G_1 és G_2 i-típusú, következik, hogy $G_\cup$ is az lesz. Ha $i = 1$, akkor, ha valamelyik nyelvtan generálja az üres szót, akkor a $G_\cup$ szabályaiból kivesszük a megfelelő (esetleg mindkettő) $S_k \rightarrow \varepsilon$ ($k = 1, 2$) szabályt és helyettesítjük $S \rightarrow \varepsilon$ szabállyal.

Szorzat. Legyen $G_\times = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}, S)$.

Könnyű igazolni, hogy $L(G_\times) = L(G_1)L(G_2)$. Az $i = 0, 2$ típusoknál $G_\times$ is ugyanolyan típusú lesz. Az $i = 1$ típusnál, ha van P_1-ben $S_1 \rightarrow \varepsilon$ szabály, de P_2-ben nincs $S_2 \rightarrow \varepsilon$ szabály, akkor a $S_1 \rightarrow \varepsilon$ szabályt helyettesítjük az $S \rightarrow S_2$ szabállyal. Hasonlóképpen járunk el a szimmetrikus esetnél. Ha van P_1-ben $S_1 \rightarrow \varepsilon$ szabály és P_2-ben $S_2 \rightarrow \varepsilon$ szabály, akkor ezeket helyettesítjük az $S \rightarrow \varepsilon$ szabállyal.

Másképp kell megadni a szabályokat a reguláris nyelvtanok ($i = 3$) esetében, mert $S \rightarrow S_1 S_2$ nem reguláris szabály. Helyette a következő nyelvtant használjuk:

$G_\times = (N_1 \cup N_2, T_1 \cup T_2, P_1' \cup P_2, S_1)$, ahol P_1' annyiban különbözik P_1-től, hogy az $A \rightarrow u, u \in T^*$ szabályok helyett $A \rightarrow uS_2$ kerül be P_1'-be .

Iteráció. Legyen $G_* = (N_1 \cup \{S\}, T_1, P, S)$.

2-típusú nyelvtanoknál legyen $P = P_1 \cup \{S \to S_1 S,\ S \to \varepsilon\}$. Ekkor G_* is 2-típusú lesz.

A 3-típusnál, a szorzathoz hasonlóan átalakítjuk a szabályokat, azaz $P = P_1' \cup \{S \to S_1,\ S \to \varepsilon\}$, ahol P_1' abban különbözik P_1-től, hogy minden $A \to u$ ($u \in T^*$) alakú szabály helyett $A \to uS$ alakút veszünk, a többit változatlanul hagyjuk. Ekkor G_* is 3-típusú lesz.

$i = 0, 1$-re nem jók a 2-típusnál megadott szabályok, mert az $S \to S_1 S$ alkalmazása során megtörténhet, hogy a következő levezetésekhez jutunk: $S \overset{*}{\Longrightarrow} S_1 S_1$, $S_1 \overset{*}{\Longrightarrow} \alpha_1\beta_1$, $S_1 \overset{*}{\Longrightarrow} \alpha_2\beta_2$, ahol $\beta_1\alpha_2$ egy helyettesítési szabály bal oldala. Ekkor az $S \overset{*}{\Longrightarrow} \alpha_1\beta_1\alpha_2\beta_2$ levezetésben helyettesítve $\beta_1\alpha_2$-t a neki megfelelő szabály jobb oldalával, olyan szót is generálhatunk, amelyik nincsen benne az iterált nyelvben. Hogy ezt elkerüljük, először feltételezzük, hogy a nyelvtan normálalakú, azaz szabályok bal oldalán nincsenek terminális jelek (*lásd* 900. oldal), majd bevezetünk egy új S' nemterminálist, tehát a nemterminálisok halmaza most $N_1 \cup \{S, S'\}$, a szabályok pedig a következők lesznek:

$$P = P_1 \cup \{S \to \varepsilon,\ S \to S_1 S'\} \cup \{aS' \to aS \mid a \in T_1\}\ .$$

Így most már elkerülhetjük, hogy esetleg olyan szabályt is alkalmazzunk a levezetésben, amelynek bal oldala átnyúlik a szavak határán az iteráció miatt. Most már az előbbi levezetéseket csak úgy lehet alkalmazni, hogy az $S \Longrightarrow S_1 S'$ helyettesítéssel kezdjük, majd eljutunk az $S \overset{*}{\Longrightarrow} \alpha_1\beta_1 S'$ levezetéshez. Ezt csak akkor tudjuk S' helyettesítésével folytatni, ha β_1 utolsó betűje terminális, és miután alkalmaztuk valamelyik $aS' \to aS$ helyettesítési szabályt.

Könnyen igazolható, hogy mindegyik típus esetében $L(G_*) = L(G_1)^*$. ■

Gyakorlatok

19.1-1. Adjunk meg egy nyelvtant, amely az $L = \{uu^{-1} \mid u \in \{a, b\}^*\}$ nyelvet generálja, és határozzuk meg a típusát.

19.1-2. Adott a $G = (N, T, P, S)$ kiterjesztett környezetfüggetlen nyelvtan, ahol

$N = \{S, A, C, D\}$, $T = \{a, b, c, d, e\}$,

$P = \{S \to abCADe,\ C \to cC,\ C \to \varepsilon,\ D \to dD,\ D \to \varepsilon,\ A \to \varepsilon,\ A \to dDcCA\}$.

Adjunk meg egy vele ekvivalens környezetfüggetlen nyelvtant.

19.1-3. Mutassuk meg, hogy Σ^* és Σ^+ reguláris nyelvek, tetszőleges Σ ábécére.

19.1-4. Adjunk meg egy nyelvtant az $L = \{u \in \{0, 1\}^* \mid n_0(u) = n_1(u)\}$ nyelv generálására, ahol $n_0(u)$ az u szóban szereplő 0-k, $n_1(u)$ pedig az 1-ek számát jelenti.

19.1-5. Adjunk meg egy nyelvtant, amely a természetes számokat generálja.

19.1-6. Adjunk meg egy-egy nyelvtant a következő nyelvek generálására.

$L_1 = \{a^n b^m c^p \mid n \geq 1, m \geq 1, p \geq 1\}$,

$L_2 = \{a^{2n} \mid n \geq 1\}$,

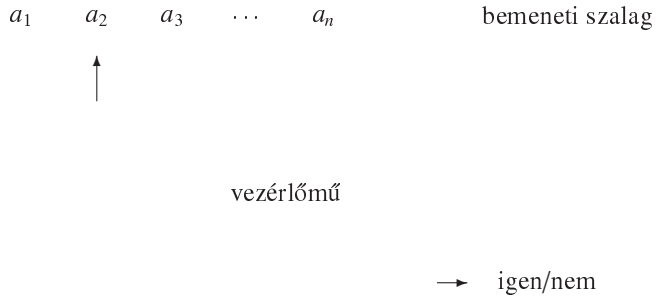
$L_3 = \{a^n b^m \mid n \geq 0, m \geq 0\}$,

$L_4 = \{a^n b^m \mid n \geq m \geq 1\}$.

19.1-7. Adott a $G = (N, T, P, S)$ kiterjesztett nyelvtan, ahol $N = \{S, A, B, C\}$, $T = \{a\}$, P pedig a következő helyettesítési szabályokat tartalmazza:

$$S \to BAB,\ BA \to BC,\ CA \to AAC,\ CB \to AAB,\ A \to a,\ B \to \varepsilon\ .$$

Milyen típusú ez a nyelvtan? Adjunk meg egy vele ekvivalens, ugyanolyan típusú nem kiterjesztett nyelvtant. Milyen nyelvet generál a G nyelvtan?



19.1. ábra. Véges automata.

19.2. Véges automaták és reguláris nyelvek

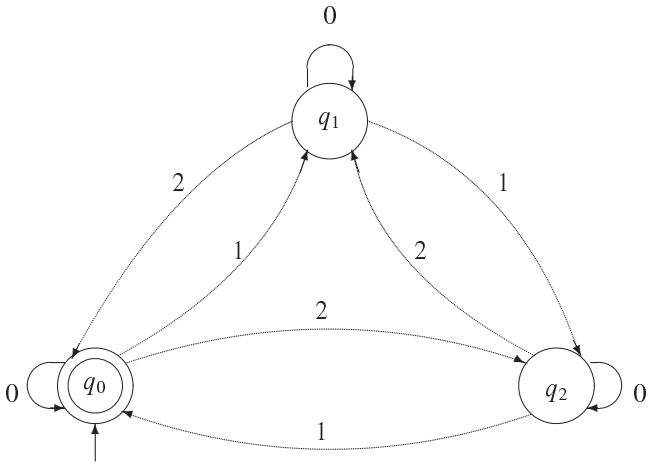
A véges automaták olyan számítási modellek, amelyek rendelkeznek egy bemeneti szalaggal és több állapottal (19.1. ábra). Az állapotok között vannak kezdőállapotnak, illetve végállapotnak nevezett állapotok. A véges automata egy szót kap bemenetként, amely a bemeneti szalagra van írva, és a bemeneti szó első betűjét olvassa. Az automata kezdőállapotból indul, a szalagról sorra olvassa a betűket, miközben állapotot válthat. Érzékeli, ha végigolvasta a szót, és amennyiben az utolsó állapot végállapot, akkor azt mondjuk, hogy felismerte az adott szót. Egy ilyen automata által felismert szavak halmazát az automata által felismert nyelvnek nevezzük.

19.9. definíció. ***Nemdeterminisztikus véges automatának*** *nevezzük az* $\mathrm{A} = (Q, \Sigma, E, I, F)$ *rendezett ötöst, ahol*

- Q *egy véges, nem üres halmaz, az* ***állapotok*** *halmaza,*
- Σ *a* ***bemeneti ábécé,***
- E *az* ***átmenetek*** *(vagy élek) halmaza, ahol* $E \subseteq Q \times \Sigma \times Q$,
- $I \subseteq Q$ *a* ***kezdőállapotok*** *halmaza,*
- $F \subseteq Q$ *a* ***végállapotok*** *halmaza.*

A nemdeterminisztikus véges automata tulajdonképpen egy olyan irányított, címkézett gráf, amelynek csúcsai az állapotok, és egy p csúcsából akkor vezet egy a betűvel megcímkézett él a q csúcsba, ha $(p, a, q) \in E$. Az állapotokat jelentő csúcsok között bizonyosak kezdő- és bizonyosak végállapotok. A kezdőállapotokat egy-egy befutó nyíl jelzi, míg a végállapotokat két-két koncentrikus kör. Ha két csúcs között több, ugyanolyan irányú él van, akkor ezeket helyettesítjük egyetlen éllel, amelyre a betűket vesszővel elválasztva írjuk. Ezt a gráfot átmenetgráfnak fogjuk hívni.

19.9. példa. Legyen $\mathrm{A} = (Q, \Sigma, E, I, F)$, ahol $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1, 2\}$,
$E = \{(q_0, 0, q_0), (q_0, 1, q_1), (q_0, 2, q_2),$
$(q_1, 0, q_1), (q_1, 1, q_2), (q_1, 2, q_0),$
$(q_2, 0, q_2), (q_2, 1, q_0), (q_2, 2, q_1)\}$
$I = \{q_0\}$, $F = \{q_0\}$.
Az automata a 19.2. ábrán látható.



19.2. ábra. A 19.9. példában szereplő véges automata.

Egy (p, a, q) élnek p a kezdőpontja, q a végpontja, a pedig a címkéje. Értelmezzük a gráfoknál használatos ***séta*** fogalmát. A

$$(q_0, a_1, q_1), (q_1, a_2, q_2), \ldots, (q_{n-2}, a_{n-1}, q_{n-1}), (q_{n-1}, a_n, q_n)$$

élsorozat a nemdeterminisztikus véges automata egy sétája, amelynek címkéje az $a_1 a_2 \ldots a_n$ szó. Ha $n = 0$, akkor $q_0 = q_n$ és $a_1 a_2 \ldots a_n = \varepsilon$. Az ilyen sétát ***üres sétának*** nevezzük. A séta jelölése

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n \ ,$$

vagy ha $w = a_1 a_2 \ldots a_n$, akkor röviden: $q_0 \xrightarrow{w} q_n$. Itt q_0 a séta kezdőpontja, q_n pedig a végpontja. A sétában szereplő állapotok nem feltétlenül különbözők. Egy séta ***produktív***, ha kezdőpontja kezdőállapot, végpontja pedig végállapot. Azt mondjuk, hogy a nemdeterminisztikus véges automata ***felismer*** egy szót, ha az a szó egy produktív séta címkéje. Az ε üres szót a nemdeterminisztikus véges automata akkor ismeri fel, ha van produktív üres séta, amely egyenértékű azzal, hogy van olyan kezdőállapot, amely egyben végállapot is.

Egy nemdeterminisztikus véges automata által felismert szavak halmazát a nemdeterminisztikus véges automata által felismert nyelvnek mondjuk. Az A ***nemdeterminisztikus véges automata által felismert nyelv***

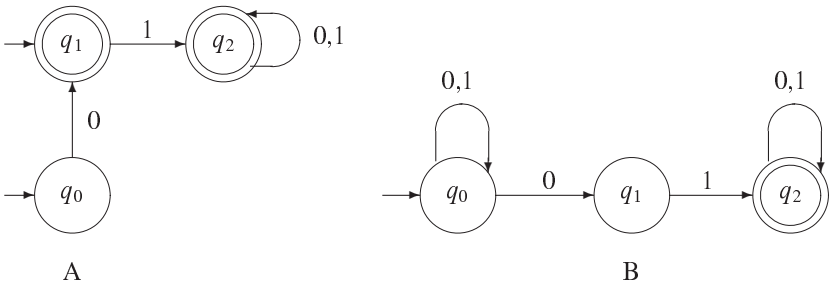
$$L(\mathrm{A}) = \left\{ w \in \Sigma^* \mid \exists p \in I,\ \exists q \in F,\ \exists p \xrightarrow{w} q \right\} .$$

Az A_1 és A_2 véges automaták ***ekvivalensek,*** ha $L(\mathrm{A}_1) = L(\mathrm{A}_2)$.

Sokszor hasznos definiálni a következő átmenetfüggvényt:

$$\delta : Q \times \Sigma \to P(Q), \quad \delta(p, a) = \{ q \in Q \mid (p, a, q) \in E \} .$$

Ez a függvény egy p állapotnak és egy a betűnek megfelelteti azt az állapothalmazt, amelynek állapotaiba átmehet a véges automata, ha a p állapotban van és az olvasófej az a



19.3. ábra. Nemdeterminisztikus véges automaták.

δ	0	1
q_0	$\{q_1\}$	$\emptyset$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$

A

δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$

B

19.4. ábra. A 19.3. ábrán látható két véges automata átmenettáblázata.

betűre mutat. Jelöljük $|H|$-val a H halmaz elemeinek a számát.[2] Egy nemdeterminisztikus véges automatáról azt mondjuk, hogy ***determinisztikus,*** ha

$$|I| = 1 \text{ és } |\delta(q,a)| \leq 1,\ \forall q \in Q,\ \forall a \in \Sigma\ .$$

A 19.2. ábrán egy determinisztikus véges automata látható.

A $|\delta(q,a)| \leq 1$ feltételt helyettesíthetjük a következővel:

$$(p,a,q) \in E,\ (p,a,r) \in E \implies q = r\ , \forall p,q,r \in Q, \forall a \in \Sigma\ .$$

Ha a determinisztikus véges automata olyan, hogy $|\delta(q,a)| = 1$ minden $q \in Q$ állapotra és minden $a \in \Sigma$ betűre, akkor azt mondjuk, hogy ***teljes, determinisztikus véges automata***.

Minden determinisztikus véges automata teljessé tehető egy új állapot bevezetésével, amelyet csapdaállapotnak szokás nevezni. Legyen $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ egy determinisztikus véges automata. A vele ekvivalens teljes, determinisztikus véges automata pedig $\mathrm{A}' = (Q \cup \{s\}, \Sigma, E', \{q_0\}, F)$, ahol s egy új állapot és $E' = E \cup \{(p,a,s) \mid \delta(p,a) = \emptyset, p \in Q, a \in \Sigma\} \cup \{(s,a,s) \mid a \in \Sigma\}$. Mivel ez az új állapot nem produktív, könnyű belátni, hogy $L(\mathrm{A}) = L(\mathrm{A}')$.

A következőkben, amennyiben csak véges automatát írunk, ezalatt mindig nemdeterminisztikus véges automatát értünk. Ha az automata determinisztikus, akkor ezt mindig hangsúlyozzuk.

Az átmenetfüggvény segítségével könnyen elkészíthetjük a véges automata átmenettáblázatát. A táblázat sorai Q elemeivel, oszlopai Σ elemeivel vannak indexelve. A $q \in Q$

[2] Nem értelemzavaró az, hogy ugyanazt a jelölést használjuk a halmaz számosságára, mint a szó hosszára, hiszen a szót mindig kisbetűvel jelöljük, a halmazt pedig nagybetűvel. Kivétel csak a $\delta(q,a)$ jelölés, amely nem téveszthető össze szóval.

sor és az $a \in \Sigma$ oszlop kereszteződésénél található elem $\delta(q, a)$. A 19.2. ábra esetében az átmenettáblázat a következő :

δ	0	1	2
q_0	$\{q_0\}$	$\{q_1\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\{q_2\}$	$\{q_0\}$
q_2	$\{q_2\}$	$\{q_0\}$	$\{q_1\}$

A 19.3. ábrán látható két véges automata egyike sem determinisztikus, az első (az A véges automata) azért mert két kezdőállapota van, a második (a B véges automata) pedig azért, mert a q_0 állapotból 0-val a q_0 és q_1 állapotokba is el lehet jutni. A két véges automata átmenettáblázata a 19.4. ábrán látható. L(A) azon szavak halmaza $\Sigma = \{0, 1\}$ felett, amelyek nem kezdődnek két 0-val (természetesen az ε is eleme a nyelvnek), L(B) pedig azon szavaké, amelyekben van 01 részszó.

Elérhetetlen állapotok kizárása

Legyen A $= (Q, \Sigma, E, I, F)$ egy véges automata. Egy állapotot ***elérhetőnek*** nevezünk, ha egy kezdőállapotból valamely bemeneti szó hatására eljuthatunk ebbe az állapotba. A következő algoritmus meghatározza egy véges automatában az elérhető állapotokat az $U_0, U_1, U_2, \ldots$ halmazok felépítésével, ahol U_0 a kezdőállapotok halmaza, és tetszőleges $i \geq 1$ természetes számra U_i azon állapotok halmaza, amelyekbe el lehet jutni valamely kezdőállapotból egy legfeljebb i hosszúságú bemeneti szó hatására.

ELÉRHETŐ-ÁLLAPOTOK(A, U)

```
1  U_0 ← I
2  i ← 0
3  repeat
4        i ← i + 1
5        U_i ← U_{i-1}
6        for minden q ∈ U_{i-1}
7            do for minden a ∈ Σ
8                   do U_i ← U_i ∪ δ(q, a)
9  until U_i = U_{i-1}
10 U ← U_i
11 return U
```

A $Q \setminus U$ halmaz elemei nem elérhető állapotok, ezért kizárhatók a véges automatából anélkül, hogy az általa felismert nyelvet megváltoztatnánk.

Ha $|Q| = n$ és $|\Sigma| = m$, akkor a fenti algoritmus lépésszáma legrosszabb esetben $O(n^2m)$, mivel a két egymásba ágyazott ciklus lépésszáma legfeljebb nm, a **repeat** ciklusé pedig n.

Az így megkonstruált U halmaznak megvan az a tulajdonsága, hogy L(A) $\neq \emptyset$ akkor és csakis akkor, ha $U \cap F \neq \emptyset$. Ezáltal a fenti algoritmus kiegészíthető a $U \cap F \neq \emptyset$ feltétellel, hogy eldöntse, hogy a felismert L(A) nyelv üres-e vagy sem.

Nemproduktív állapotok kizárása

Legyen A $= (Q, \Sigma, E, I, F)$ egy véges automata. Egy állapotot ***produktívnak*** nevezünk, ha abból az állapotból valamely bemeneti szó hatására eljuthatunk egy végállapotba.

A produktív állapotok meghatározására szolgáló következő algoritmus használja a δ^{-1} függvényt, amelynek definíciója a következő:

$$\delta^{-1} : Q \times \Sigma \to \mathcal{P}(Q), \quad \delta^{-1}(p,a) = \{q \mid (q,a,p) \in E\} .$$

Ez a függvény egy p állapotra és egy a betűre megadja azt az állapothalmazt, amelynek elemeiből az a betű hatására el lehet jutni a p állapotba.

PRODUKTÍV-ÁLLAPOTOK(A, V)

```
1  V_0 ← F
2  i ← 0
3  repeat
4          i ← i + 1
5          V_i ← V_{i-1}
6          for minden p ∈ V_{i-1}
7              do for minden a ∈ Σ
8                      do V_i ← V_i ∪ δ^{-1}(p, a)
9  until V_i = V_{i-1}
10 V ← V_i
11 return V
```

A $Q \setminus V$ halmaz elemei nem produktív állapotok, ezért kizárhatók a véges automatából, anélkül, hogy az általa felismert nyelvet befolyásolnánk.

Ha n az állapotok száma és m a betűk száma, akkor a lépésszám ebben az esetben is $O(n^2m)$, akárcsak az ELÉRHETŐ-ÁLLAPOTOK algoritmus esetében.

Az így megkonstruált V halmaznak is megvan az a tulajdonsága, hogy $L(\mathrm{A}) \neq \emptyset$ akkor és csakis akkor, ha $V \cap I \neq \emptyset$. Ezért, kis módosítással, ez az algoritmus is használható annak eldöntésére, hogy $L(\mathrm{A})$ üres-e.

19.2.1. Nemdeterminisztikus véges automata átalakítása determinisztikus véges automatává

A következőkben megmutatjuk, hogy tetszőleges nemdeterminisztikus véges automata átalakítható olyan determinisztikus véges automatává, amelyik ekvivalens az eredetivel.

19.10. tétel. *Tetszőleges nemdeterminisztikus véges automatához mindig megkonstruálható egy vele ekvivalens determinisztikus véges automata.*

Bizonyítás. Legyen $\mathrm{A} = (Q, \Sigma, E, I, F)$ egy nemdeterminisztikus véges automata. Értelmezzük az $\overline{\mathrm{A}} = (\overline{Q}, \Sigma, \overline{E}, \overline{I}, \overline{F})$ determinisztikus véges automatát, ahol

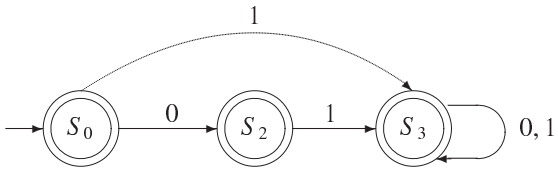
$\overline{Q} = \mathcal{P}(Q) \setminus \emptyset$,

$\overline{E}$ élei azon (S, a, R) alakú hármasokból állnak, amelyekre $R, S \in \overline{Q}$, egyik sem üres, $a \in \Sigma$ és $R = \bigcup_{p \in S} \delta(p, a)$,

$\overline{I} = \{I\}$,

$\overline{F} = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$.

Be kell bizonyítanunk, hogy $L(\mathrm{A}) = L(\overline{\mathrm{A}})$.



19.5. ábra. A 19.3. ábra A véges automatájával ekvivalens determinisztikus véges automata.

a) Bebizonyítjuk, hogy $L(\mathrm{A}) \subseteq L(\overline{\mathrm{A}})$. Legyen $w = a_1 a_2 \dots a_k \in L(\mathrm{A})$. Ekkor létezik a

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} q_k, \quad q_0 \in I,\ q_k \in F$$

séta. Képezzük a következő halmazokat, felhasználva az $\overline{\mathrm{A}}$ véges automata $\overline{\delta}$ átmenetfüggvényét: $S_0 = \{q_0\}$, $\overline{\delta}(S_0, a_1) = S_1, \dots \overline{\delta}(S_{k-1}, a_k) = S_k$. Ekkor $q_1 \in S_1, \dots, q_k \in S_k$, és mivel $q_k \in F$, következik, hogy $S_k \cap F \neq \emptyset$, tehát $S_k \in \overline{F}$. Így létezik az

$$S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} S_{k-1} \xrightarrow{a_k} S_k, \quad S_0 \subseteq I,\ S_k \in \overline{F}$$

séta. Vannak olyan $S'_0, \dots, S'_k$ halmazok, amelyekre $S'_0 = I$, továbbá minden $i = 0, 1, \dots, k$-ra $S_i \subseteq S'_i$ és

$$S'_0 \xrightarrow{a_1} S'_1 \xrightarrow{a_2} S'_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} S'_{k-1} \xrightarrow{a_k} S'_k$$

is produktív séta. Ezért $w \in L(\overline{\mathrm{A}})$. Tehát $L(\mathrm{A}) \subseteq L(\overline{\mathrm{A}})$.

b) Bebizonyítjuk, hogy $L(\overline{\mathrm{A}}) \subseteq L(\mathrm{A})$. Legyen $w = a_1 a_2 \dots a_k \in L(\overline{\mathrm{A}})$. Ekkor létezik a

$$\overline{q_0} \xrightarrow{a_1} \overline{q_1} \xrightarrow{a_2} \overline{q_2} \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} \overline{q}_{k-1} \xrightarrow{a_k} \overline{q}_k, \quad \overline{q}_0 \in \overline{I},\ \overline{q}_k \in \overline{F}$$

séta. Az $\overline{F}$ definíciója alapján $\overline{q}_k \cap F \neq \emptyset$, azaz létezik $q_k \in \overline{q}_k \cap F$, tehát $q_k \in F$ és $\overline{q}_k$ definíciója alapján létezik q_{k-1} úgy, hogy $(q_{k-1}, a_k, q_k) \in E$. Hasonlóképpen, léteznek a $q_{k-2}, \dots, q_1, q_0$ állapotok úgy, hogy $(q_{k-2}, a_k, q_{k-1}) \in E, \dots, (q_0, a_1, q_1) \in E$, ahol $q_0 \in \overline{q}_0 = I$, ezért létezik a

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} q_k, \quad q_0 \in I,\ q_k \in F$$

séta, tehát $L(\overline{\mathrm{A}}) \subseteq L(\mathrm{A})$. ■

A determinisztikus véges automata megkonstruálásában segítségünkre lehet ennek $\overline{\delta}$ átmenetfüggvénye:

$$\overline{\delta}(\overline{q}, a) = \left\{ \bigcup_{q \in \overline{q}} \delta(q, a) \right\}, \quad \forall \overline{q} \in \overline{Q}, \forall a \in \Sigma.$$

Mivel az üres halmazt kizártuk az állapotok közül, a fenti képletben $\{\emptyset\}$ helyett $\emptyset$-t írunk.

19.10. példa. Alkalmazzuk a 19.10. tételt a 19.3. ábra A véges automatájára, amely nemdeterminisztikus. Vezessük be a determinisztikus véges automata állapotaira a következő jelöléseket:

$S_0 := \{q_0, q_1\}$, $S_1 := \{q_0\}$, $S_2 := \{q_1\}$, $S_3 := \{q_2\}$,
$S_4 := \{q_0, q_2\}$, $S_5 := \{q_1, q_2\}$, $S_6 := \{q_0, q_1, q_2\}$,

amelyek közül S_0 a kezdőállapot. Ekkor alkalmazva az átmenetfüggvényt a következő átmenettáblázatot kapjuk:

$\overline{\delta}$	0	1
S_0	$\{S_2\}$	$\{S_3\}$
S_1	$\{S_2\}$	$\emptyset$
S_2	$\emptyset$	$\{S_3\}$
S_3	$\{S_3\}$	$\{S_3\}$
S_4	$\{S_5\}$	$\{S_3\}$
S_5	$\{S_3\}$	$\{S_3\}$
S_6	$\{S_5\}$	$\{S_3\}$

Ebben az automatában sok elérhetetlen állapot van. Az ELÉRHETŐ-ÁLLAPOTOK algoritmus szerint a véges automata elérhető állapotai a következők szerint határozhatók meg.

$U_0 = \{S_0\}$, $U_1 = \{S_0, S_2, S_3\}$, $U_2 = \{S_0, S_2, S_3\} = U_1 = U$.

Az S_0 kezdőállapot és egyben végállapot is. Az S_2 és S_3 mindegyike végállapot. Az S_1, S_4, S_5, S_6 elérhetetlen állapotok, ezért kizárjuk őket a véges automatából. Az így kapott véges automata átmenettáblázata a következő:

$\overline{\delta}$	0	1
S_0	$\{S_2\}$	$\{S_3\}$
S_2	$\emptyset$	$\{S_3\}$
S_3	$\{S_3\}$	$\{S_3\}$

Az ennek megfelelő determinisztikus véges automata átmenetgráfja a 19.5. ábrán látható.

A 19.10. tétel által nyújtott algoritmus egyszerűsíthető. Nem kell a nemdeterminisztikus véges automata állapothalmazának minden részhalmazát figyelembe venni. Az $\overline{\mathrm{A}}$ véges automata állapotait fokozatosan kapjuk meg úgy, hogy elindulunk a $\overline{q}_0 = I$ állapottal, meghatározzuk a $\overline{\delta}(\overline{q}_0, a)$ állapotokat, minden $a \in \Sigma$ elemre. Az újonnan kapott állapotokra szintén meghatározzuk az átmenetek alapján a belőlük elérhető állapotokat. Ezt addig folytatjuk, amíg már nem kapunk új állapotokat.

Az előző példánkban legyen $\overline{q}_0 := \{q_0, q_1\}$ a kezdőállapot, és innen

$\overline{\delta}(\overline{q}_0, 0) = \{\overline{q}_1\}$, ahol $\overline{q}_1 := \{q_1\}$, $\overline{\delta}(\overline{q}_0, 1) = \{\overline{q}_2\}$, ahol $\overline{q}_2 := \{q_2\}$,
$\overline{\delta}(\overline{q}_1, 0) = \emptyset$, $\overline{\delta}(\overline{q}_1, 1) = \{\overline{q}_2\}$,
$\overline{\delta}(\overline{q}_2, 0) = \{\overline{q}_2\}$, $\overline{\delta}(\overline{q}_2, 1) = \{\overline{q}_2\}$.

Az átmenettáblázat a következő:

$\overline{\delta}$	0	1
$\overline{q}_0$	$\{\overline{q}_1\}$	$\{\overline{q}_2\}$
$\overline{q}_1$	$\emptyset$	$\{\overline{q}_2\}$
$\overline{q}_2$	$\{\overline{q}_2\}$	$\{\overline{q}_2\}$

amely lényegében ugyanaz (ha eltekintünk a jelölésektől), mint az előbb kapott véges automata átmenettáblázata.

A következő algoritmus egy $\mathrm{A} = (Q, \Sigma, E, I, F)$ nemdeterminisztikus véges automatához megkonstruálja a vele ekvivalens $\overline{\mathrm{A}} = (\overline{Q}, \Sigma, \overline{E}, \overline{I}, \overline{F})$ determinisztikus véges automata M átmenettáblázatát, de nem tartalmazza annak megállapítását, hogy egy állapot

végállapot-e vagy sem. Ez utóbbi azonban könnyűszerrel beépíthető. Az algoritmusban a BENNEVAN($\overline{q}$, $\overline{Q}$) értéke igaz, ha a $\overline{q}$ állapot már szerepel a $\overline{Q}$ halmazban, és hamis ellenkező esetben. Legyen $a_1, a_2, \ldots, a_m$ a Σ betűinek egy felsorolása.

NEMDET-DET(A, $\overline{\text{A}}$)

```
1  q̄_0 ← I
2  Q̄ ← {q̄_0}
3  i ← 0                                          ▷ i a sorokat számolja.
4  k ← 0                                          ▷ k az állapotokat számolja.
5  repeat
6        for j = 1, 2, ..., m                     ▷ j az oszlopokat számolja.
7            do q̄ ← ⋃_{p∈q̄_i} δ(p, a_j)
8               if q̄ ≠ ∅
9                  then if BENNEVAN(q̄, Q̄)
10                           then M[i, j] ← {q̄}
11                           else k ← k + 1
12                                q̄_k ← q̄
13                                M[i, j] ← {q̄_k}
14                                Q̄ ← Q̄ ∪ {q̄_k}
15                 else M[i, j] ← ∅
16       i ← i + 1
17 until i = k + 1
18 return az Ā automata M átmenettáblázata
```

Mivel a **repeat** ciklust annyiszor végezzük el, ahány állapota van az új véges automatának, legrosszabb esetben ez exponenciális érték is lehet, hisz ha a nemdeterminisztikus véges automata állapotainak száma n, akkor az eredményautomatának akár $2^n - 1$ állapota is lehet. (Egy n elemű halmaz részhalmazainak a száma, beleértve az üres halmazt is, 2^n.)

A 19.10. tétel szerint tetszőleges nemdeterminisztikus véges automatához mindig hozzárendelhető egy vele ekvivalens determinisztikus véges automata. Ez fordítva is igaz, mivel az értelmezés szerint minden determinisztikus véges automata egyben nemdeterminisztikus is. Ezért a nemdeterminisztikus véges automaták ugyanazt a nyelvosztályt ismerik fel, mint a determinisztikus véges automaták.

19.2.2. Determinisztikus véges automaták ekvivalenciájának vizsgálata

Ebben a részben csak teljes, determinisztikus véges automatákkal dolgozunk. Ezen automaták esetében a $\delta(q, a)$ halmaz mindig egyetlen elemet tartalmaz. Néha egyszerűbb, bizonyos képletekben, a $\delta(q, a)$ halmaz helyett annak az elemét használni, ezért értelmezzük az egyelemű $A = \{a\}$ halmazra az $elem(A)$ függvényt, amely visszaadja az A halmaz egyetlen elemét, azaz $elem(A) = a$. Determinisztikus véges automaták ekvivalenciáját vizsgáljuk az azonos címkéjű, kezdőállapottal kezdődő séták segítségével a két véges automatában. Ha egyik séta végállapottal végződik, a másik pedig nem, akkor a két automata nyilvánvalóan nem lehet ekvivalens.

Adott két determinisztikus véges automata ugyanazon ábécé felett: $A = (Q, \Sigma, E, \{q_0\}, F)$ és $A' = (Q', \Sigma, E', \{q_0'\}, F')$, amelyek ekvivalenciáját vizsgáljuk. Készítünk egy táblázatot, amely (q, q') alakú állapotpárokat fog tartalmazni, ahol $q \in Q$ és $q' \in Q'$. A táblázat második oszlopától kezdődően a Σ ábécé minden betűjének megfeleltetünk egy oszlopot. Ha a táblázat i-edik sorának első eleme (q, q'), akkor az i-edik sor és az a betűhöz tartozó oszlop találkozásánál levő elem $\Big(elem(\delta(q,a)), elem(\delta'(q',a))\Big)$ lesz.

	$\ldots \quad a \quad \ldots$
$\ldots$	$\ldots$
(q, q')	$\Big(elem(\delta(q,a)), elem(\delta'(q',a))\Big)$
$\ldots$	$\ldots$

A táblázat első sorának első oszlopába a (q_0, q_0') állapotpár kerül, majd kitöltjük az első sort a fent leírtak alapján. Ha az első sor valamelyik oszlopában megjelenik egy olyan pár, amelyre egyik állapot végállapot, a másik meg nem, akkor az algoritmust befejezzük: ***a két véges automata nem ekvivalens***. Amennyiben nincs ilyen pár, minden új párt beírunk az első oszlopba, és folytatjuk az algoritmust a következő olyan sorral, amelyik még nincs kitöltve. Ha már nem jelenik meg új állapotpár, és minden párra igaz, hogy mindkét eleme végállapot vagy egyik sem az, akkor az algoritmus szintén befejeződik, és ***a két véges automata ekvivalens***.

AUTOMATA-EKVIVALENCIA(A, A′)

```
1  írjuk be a táblázat első sorának első oszlopába a (q_0, q_0') állapotpárt
2  i ← 0
3  repeat
4        i ← i + 1
5        legyen (q, q') a táblázat i-edik sorának első oszlopában levő állapotpár
6        for minden a ∈ Σ betűre
7            do írjuk be a táblázat i-edik sora a jelzésű oszlopába a
                 (elem(δ(q,a)), elem(δ'(q',a))) állapotpárt
8               if (elem(δ(q,a)), elem(δ'(q',a))) egyik állapota végállapot, a másik nem
9                  then return NEM
10                 else  írjuk be a (elem(δ(q,a)), elem(δ'(q',a))) párt az első oszlop
                         következő üres sorába, ha még nem szerepel az első oszlopban
11 until (i + 1)-edik sor első eleme üres
12 return IGEN
```

Ha $|Q| = n$, $|Q'| = n'$ és $|\Sigma| = m$, akkor figyelembe véve, hogy a **repeat** ciklust legrosszabb esetben nn'-szer kell végrehajtani, a **for** ciklust pedig m-szer, kiszámíthatjuk, hogy a maximális lépésszám legrosszabb esetben $O(nn'm)$, vagy ha $n = n'$, akkor $O(n^2m)$.

Algoritmusunkat két véges automata ekvivalenciájának vizsgálatára csak a teljes, determinisztikus véges automatákra írtuk le. Ha két tetszőleges nemdeterminisztikus véges automatáról szeretnénk eldönteni, hogy ekvivalensek-e, akkor előbb mindkettőt átalakítjuk determinisztikus véges automatává, majd alkalmazzuk a fentebb leírt algoritmust annak megállapítására, hogy ekvivalensek-e.

19.11. példa. Vizsgáljuk meg, hogy a 19.6. ábrán látható két véges automata ekvivalens-e. Elkészítjük az állapotpárok következő táblázatát.

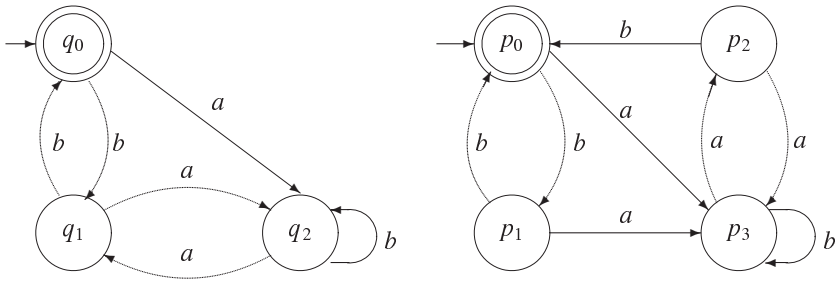
	a	b
(q_0, p_0)	(q_2, p_3)	(q_1, p_1)
(q_2, p_3)	(q_1, p_2)	(q_2, p_3)
(q_1, p_1)	(q_2, p_3)	(q_0, p_0)
(q_1, p_2)	(q_2, p_3)	(q_0, p_0)

A két véges automata ekvivalens, mivel minden lehetséges állapotpárt figyelembe vettünk, és minden pár mindkét eleme végállapot vagy egyik sem az.

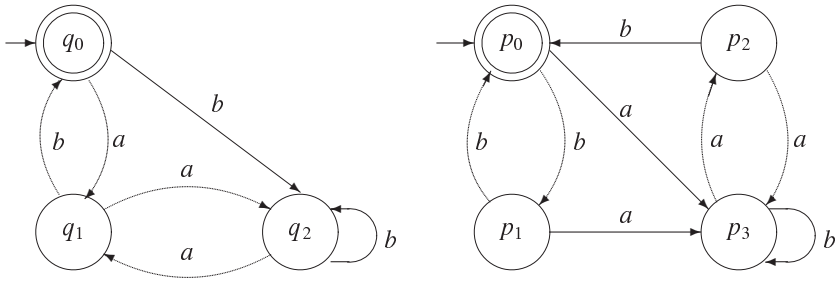
19.12. példa. A 19.7. ábrán látható két véges automata állapotpárjainak táblázata:

	a	b
(q_0, p_0)	(q_1, p_3)	(q_2, p_1)
(q_1, p_3)	(q_2, p_2)	(q_0, p_3)
(q_2, p_1)		
(q_2, p_2)		

A két véges automata nem ekvivalens, mivel a második sor utolsó oszlopában a (q_0, p_3) állapotpár első eleme végállapot, a második pedig nem az.



19.6. ábra. Ekvivalens véges automaták (19.11. példa).



19.7. ábra. Nem ekvivalens véges automaták (19.12. példa).

19.2.3. Véges automaták és reguláris nyelvtanok ekvivalenciája

Láttuk, hogy a nemdeterminisztikus véges automaták ugyanazt a nyelvosztályt ismerik fel, mint a determinisztikus véges automaták. A következő két tétel azt mutatja, hogy ez a nyelvosztály nem más, mint a reguláris nyelvek osztálya.

19.11. tétel. *Ha L egy tetszőleges determinisztikus véges automata által felismert nyelv, akkor megkonstruálható olyan reguláris nyelvtan, amelyik az L nyelvet generálja.*

Bizonyítás. Legyen $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ az L nyelvet felismerő determinisztikus véges automata, azaz $L = L(\mathrm{A})$. Értelmezzük a $G = (Q, \Sigma, P, q_0)$ reguláris nyelvtant a következő szabályokkal:

• Ha $(p, a, q) \in E$ valamilyen $p, q \in Q$ és $a \in \Sigma$-ra, akkor vegyük be P-be a $p \to aq$ szabályt.

• Ha $(p, a, q) \in E$ és $q \in F$, akkor a fenti szabály mellé még vegyük be P-be a $p \to a$ szabályt is.

Bebizonyítjuk, hogy $L(G) = L(\mathrm{A}) \setminus \{\varepsilon\}$.

Legyen $u = a_1 a_2 \dots a_n \in L(\mathrm{A})$ és $u \neq \varepsilon$. Ekkor, mivel az A véges automata felismeri az u szót, létezik a

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n, \quad q_n \in F$$

séta. Ekkor P-ben léteznek a következő szabályok:

$$q_0 \to a_1 q_1, \quad q_1 \to a_2 q_2, \quad \dots, \quad q_{n-2} \to a_{n-1} q_{n-1}, \quad q_{n-1} \to a_n$$

(az utóbbi szabály jobb oldalán nem szerepel q_n, mivel $q_n \in F$), tehát létezik a

$$q_0 \Longrightarrow a_1 q_1 \Longrightarrow a_1 a_2 q_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} q_{n-1} \Longrightarrow a_1 a_2 \dots a_n$$

levezetés. Ezért $u \in L(G)$.

Fordítva, legyen $u = a_1 a_2 \dots a_n \in L(G)$, és $u \neq \varepsilon$. Ekkor létezik a

$$q_0 \Longrightarrow a_1 q_1 \Longrightarrow a_1 a_2 q_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} q_{n-1} \Longrightarrow a_1 a_2 \dots a_n$$

levezetés, amelyben a

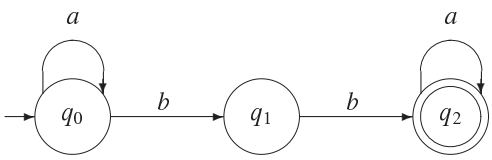
$$q_0 \to a_1 q_1, \quad q_1 \to a_2 q_2, \quad \dots, \quad q_{n-2} \to a_{n-1} q_{n-1}, \quad q_{n-1} \to a_n$$

szabályokat használtuk, amelyek értelmezés szerint azt jelentik, hogy az A véges automatában létezik a következő séta:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n,$$

és mivel q_n végállapot, következik, hogy $u \in L(\mathrm{A}) \setminus \{\varepsilon\}$.

Ha a véges automata ε-t is felismeri, akkor a fenti nyelvtan annyiban módosul, hogy bevezetünk egy új q_0' kezdőszimbólumot q_0 helyett, bevesszük a szabályok közé a $q_0' \to \varepsilon$ szabályt, majd minden $q_0 \to \alpha$ szabály mellé bevesszük a $q_0' \to \alpha$ szabályt is. ■



19.8. ábra. A 19.13.. példa véges automatája.

19.13. példa. Adott az $\mathrm{A} = (\{q_0, q_1, q_2\}, \{a, b\}, E, \{q_0\}, \{q_2\})$ véges automata, ahol $E = \{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_2), (q_2, a, q_2)\}$. A véges automata átmenettáblázata a következő:

δ	a	b
q_0	$\{q_0\}$	$\{q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\emptyset$

Az A átmenetgráfja a 19.8. ábrán látható. A 19.11. tétel alapján a $G = (\{q_0, q_1, q_2\}, \{a, b\}, P, q_0)$ reguláris nyelvtan P szabályai a következők:

$$q_0 \to aq_0 \mid bq_1, \qquad q_1 \to bq_2 \mid b, \qquad q_2 \to aq_2 \mid a.$$

Igazolható, hogy $L(\mathrm{A}) = \{a^m bba^n \mid m \geq 0, n \geq 0\}$.

A 19.11. tétel bizonyításában megadott módszert könnyen átírhatjuk algoritmussá. Az $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ determinisztikus automatából kapott $G = (Q, \Sigma, P, q_0)$ reguláris nyelvtan szabályait a következő algoritmussal határozzuk meg.

AUTOMATÁBÓL-REGULÁRIS-NYELVTAN(A, G)

```
1  P ← ∅
2  for minden p ∈ Q
3      do for minden a ∈ Σ
4          do for minden q ∈ Q
5              do if (p, a, q) ∈ E
6                  then P ← P ∪ {p → aq}
7                       if q ∈ F
8                          then P ← P ∪ {p → a}
9  if q0 ∈ F
10   then P ← P ∪ {q0 → ε}
```

Könnyű belátni, hogy az algoritmus futási ideje $\Theta(n^2 m)$, ha az állapotok száma n és a betűk száma m. A 2–4. sorokban lévő három ciklus helyett lehet csupán egyet venni, ha az E elemeit vizsgáljuk, ekkor a futási idő legrosszabb esetben $\Theta(p)$, ahol p az átmenetek száma. Ez szintén $O(n^2 m)$, mivel lehetséges, hogy minden átmenet jelen van. Ekkor az algoritmus a következőképpen írható le:

AUTOMATÁBÓL-REGULÁRIS-NYELVTAN'(A, G)

```
1  P ← ∅
2  for minden (p, a, q) ∈ E
3      do P ← P ∪ {p → aq}
4         if q ∈ F
5           then P ← P ∪ {p → a}
6  if q_0 ∈ F
7      then P ← P ∪ {q_0 → ε}
```

19.12. tétel. *Ha $L = L(G)$ reguláris nyelv, akkor megkonstruálható olyan nemdeterminisztikus véges automata, amely az L nyelvet felismeri.*

Bizonyítás. Legyen $G = (N, T, P, S)$ az L nyelvet generáló reguláris nyelvtan. Definiáljuk az $\mathrm{A} = (Q, T, E, \{S\}, F)$ nemdeterminisztikus véges automatát a következőképpen.

- $Q = N \cup \{Z\}$, ahol $Z \notin N \cup T$ (vagyis egy új szimbólum),
- Minden $A \to aB$ szabályra értelmezzük az (A, a, B) átmenetet E-ben.
- Minden $A \to a$ szabályra értelmezzük az (A, a, Z) átmenetet E-ben.
- $F = \begin{cases} \{Z\} & \text{ha } G\text{-ben nem szerepel az } S \to \varepsilon \text{ szabály,} \\ \{Z, S\} & \text{ha } G\text{-ben szerepel az } S \to \varepsilon \text{ szabály.} \end{cases}$

Bebizonyítjuk, hogy $L(G) = L(A)$.

Legyen $u = a_1 a_2 \dots a_n \in L(G)$, $u \neq \varepsilon$. Ekkor létezik u-nak egy G-beli levezetése:

$$S \Longrightarrow a_1 A_1 \Longrightarrow a_1 a_2 A_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} A_{n-1} \Longrightarrow a_1 a_2 \dots a_n.$$

Ez a levezetés a következő szabályok alapján történt:

$$S \to a_1 A_1, \quad A_1 \to a_2 A_2, \quad \dots, \quad A_{n-2} \to a_{n-1} A_{n-1}, \quad A_{n-1} \to a_n.$$

Ekkor az A véges automata átmeneteinek értelmezése alapján létezik az

$$S \overset{a_1}{\longrightarrow} A_1 \overset{a_2}{\longrightarrow} A_2 \overset{a_3}{\longrightarrow} \cdots \overset{a_{n-1}}{\longrightarrow} A_{n-1} \overset{a_n}{\longrightarrow} Z, \quad Z \in F$$

séta. Ez azt jelenti, hogy $u \in L(\mathrm{A})$. Ha $\varepsilon \in L(G)$, akkor van $S \to \varepsilon$ szabály, de ekkor a kezdőállapot végállapot is, tehát $\varepsilon \in L(A)$. Ezért $L(G) \subseteq L(A)$.

Legyen most $u = a_1 a_2 \dots a_n \in L(\mathrm{A})$. Ez azt jelenti, hogy létezik az

$$S \overset{a_1}{\longrightarrow} A_1 \overset{a_2}{\longrightarrow} A_2 \overset{a_3}{\longrightarrow} \cdots \overset{a_{n-1}}{\longrightarrow} A_{n-1} \overset{a_n}{\longrightarrow} Z, \quad Z \in F$$

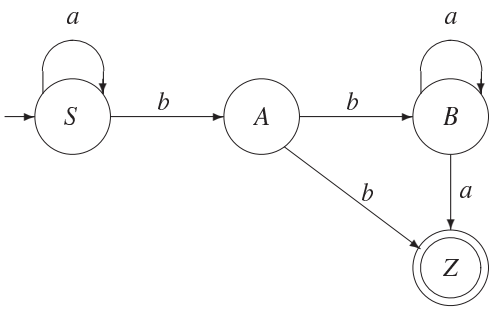
séta. Ha u az üres szó, akkor Z helyett S van, amely szintén végállapot. Más esetben csak Z szerepelhet utolsóként. Tehát G-ben szerepelnek a következő szabályok:

$$S \to a_1 A_1, \quad A_1 \to a_2 A_2, \quad \dots, \quad A_{n-2} \to a_{n-1} A_{n-1}, \quad A_{n-1} \to a_n\ ,$$

és így létezik az

$$S \Longrightarrow a_1 A_1 \Longrightarrow a_1 a_2 A_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} A_{n-1} \Longrightarrow a_1 a_2 \dots a_n$$

levezetés, tehát $u \in L(G)$, és ekkor $L(A) \subseteq L(G)$. ■



19.9. ábra. A 19.14. példa G nyelvtanához rendelt véges automata.

19.14. példa. Adott a $G = (\{S, A, B\}, \{a, b\}, \{S \to aS, S \to bA, A \to bB, A \to b, B \to aB, B \to a\}, S)$ reguláris nyelvtan. A hozzá rendelt véges automata $A = (\{S, A, B, Z\}, \{a, b\}, E, S, \{Z\})$, ahol $E = \{(S, a, S), (S, b, A), (A, b, B), (A, b, Z), (B, a, B), (B, a, Z)\}$. Ennek átmenettáblázata a következő:

δ	a	b
S	$\{S\}$	$\{A\}$
A	$\emptyset$	$\{B, Z\}$
B	$\{B, Z\}$	$\emptyset$
E	$\emptyset$	$\emptyset$

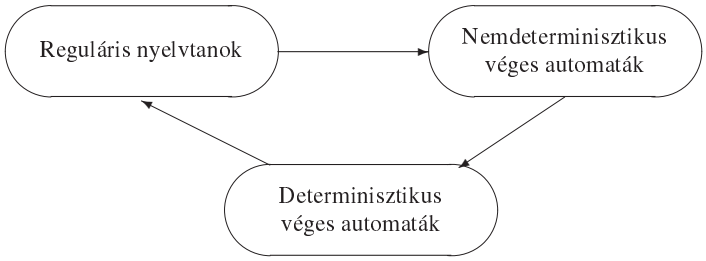
Az átmenetgráf 19.9. ábrán látható. Ez a véges automata egyszerűsíthető. A B és Z állapotok összevonhatók egyetlen végállapottá.

Az előbbi tétel alapján írunk egy algoritmust, amely hozzárendeli a $G = (N, T, P, S)$ reguláris nyelvtanhoz az $\mathrm{A} = (Q, T, E, \{S\}, F)$ véges automatát.

REGULÁRIS-NYELVTANBÓL-AUTOMATA(G, A)

```
1  E ← ∅
2  Q ← N ∪ {Z}
3  for minden A ∈ N
4      do for minden a ∈ T
5             do if (A → a) ∈ P
6                    then E ← E ∪ {(A, a, Z)}
7                for minden B ∈ N
8                    do if (A → aB) ∈ P
9                           then E ← E ∪ {(A, a, B)}
10 if (S → ε) ∉ P
11    then F ← {Z}
12    else F ← {Z, S}
```

Akárcsak az AUTOMATÁBÓL-REGULÁRIS-NYELVTAN algoritmus esetében, a futási idő ebben az esetben is $\Theta(n^2 m)$, ha a nemterminálisok száma n és a terminálisoké m. Lehetne a 3., 4. és 7. sorokban lévő ciklusokat helyettesíteni eggyel, amelyik a helyettesítési szabályokon megy végig. Ez sok esetben javítja az algoritmus hatékonyságát, amely ekkor $\Theta(p)$ lesz, ha p a szabályok száma. Az algoritmus a következő:



19.10. ábra. Kapcsolat véges automaták és reguláris nyelvtanok között. Tetszőleges reguláris nyelvtanhoz megadható egy olyan nemdeterminisztikus véges automata, amely felismeri az adott reguláris nyelvtan által generált nyelvet. Minden nemdeterminisztikus véges automata átalakítható determinisztikussá. Minden determinisztikus véges automatához megadható egy olyan reguláris nyelvtan, amely az adott automata által felismert nyelvet generálja.

REGULÁRIS-NYELVTANBÓL-AUTOMATA'(G, A)

```
1  E ← ∅
2  Q ← N ∪ {Z}
3  for minden (A → u) ∈ P
4      do if u = a
5           then E ← E ∪ {(A, a, Z)}
6         if u = aB
7           then E ← E ∪ {(A, a, B)}
8  if (S → ε) ∉ P
9    then F ← {Z}
10   else  F ← {Z, S}
```

A 19.10., 19.11. és 19.12. tételek segítségével bebizonyítottuk, hogy a reguláris nyelvek osztálya egybeesik mind a determinisztikus véges automaták, mind a nemdeterminisztikus véges automaták által felismert nyelvek osztályával. A három tétel eredményét a 19.10. ábra szemlélteti és következő tétel foglalja össze.

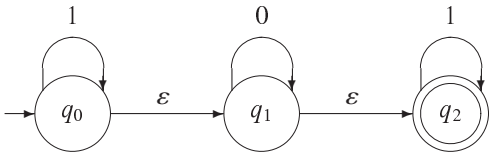
19.13. tétel. *A következő három nyelvosztály megegyezik:*

- *a reguláris nyelvek osztálya,*
- *a determinisztikus véges automatákkal felismerhető nyelvek osztálya,*
- *a nemdeterminisztikus véges automatákkal felismerhető nyelvek osztálya.*

Műveletek reguláris nyelvekkel

A 19.8. tétel alapján tudjuk, hogy a reguláris nyelvek $\mathcal{L}_3$ halmaza zárt a reguláris műveletekre, azaz ha L_1, L_2 reguláris, akkor regulárisak a következő nyelvek is: $L_1 \cup L_2$, $L_1 L_2$, L_1^*. Ezenkívül a reguláris nyelvekre igazak a következő állítások is.

Egy reguláris nyelvnek a komplementuma is reguláris. Ez könnyen igazolható véges automaták segítségével. Legyen ugyanis L egy reguláris nyelv és $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ egy, az L nyelvet felismerő teljes, determinisztikus véges automata. Könnyen belátható, hogy az $\overline{\mathrm{A}} = (Q, \Sigma, E, \{q_0\}, Q \setminus F)$ automata az $\overline{L}$ nyelvet ismeri fel. Így $\overline{L}$ is reguláris.



19.11. ábra. ε-lépéses véges automata.

Két reguláris nyelvnek a metszete is reguláris. Mivel $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, a metszet is reguláris.

Két reguláris nyelvnek a különbsége is reguláris. Mivel $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$, a különbség is reguláris.

19.2.4. ε-lépéses véges automaták és műveletek véges automatákkal

Az ε-lépéses véges automata annyiban különbözik a nemdeterminisztikus véges automatától, hogy megengedjük azt, hogy üres lépést is végezzen, azaz átmenjen egyik állapotból a másikba anélkül, hogy valamilyen bemeneti jelet olvasna. Az ε-lépéses $\mathrm{A} = (Q, \Sigma, E, I, F)$ véges automata átmeneteinek halmazára teljesül, hogy $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$.

Az ε-lépéses véges automata átmenetfüggvénye a következő:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q), \quad \delta(p, a) = \{q \in Q \mid (p, a, q) \in E\} .$$

A 19.11. ábrán látható ε-lépéses véges automata az uvw alakú szavakat ismeri fel, ahol $u \in \{1\}^*$, $v \in \{0\}^*$ és $w \in \{1\}^*$.

19.14. tétel. *Tetszőleges ε-lépéses véges automatához mindig megkonstruálható egy vele ekvivalens nemdeterminisztikus véges automata, amely ε-lépés nélküli.*

Az $\mathrm{A} = (Q, \Sigma, E, I, F)$ ε-lépéses véges automatával ekvivalens nemdeterminisztikus véges automata $\overline{\mathrm{A}} = (Q, \Sigma, \overline{E}, I, \overline{F})$ lesz. Algoritmusunk az $\overline{F}$ és az $\overline{E}$ halmazokat határozza meg.

Egy q állapotra jelöljük $\Lambda(q)$-val azon állapotok halmazát, amelyekbe el lehet jutni q-ból csupa ε-lépéssel (beleértve magát q-t is). Terjesszük ki ezt definíciót halmazokra is, azaz legyen

$$\Lambda(S) = \bigcup_{q \in S} \Lambda(q), \quad \forall S \subseteq Q .$$

Nyilvánvaló, hogy minden $q \in Q$-ra és $S \subseteq Q$-ra mind $\Lambda(q)$, mind $\Lambda(S)$ kiszámíthatók. A következőkben feltesszük, hogy ezek adottak.

A következő algoritmus az átmenetek meghatározását a $\overline{\delta}$ átmenetfüggvény segítségével végzi, amelyet az 5. sorában értelmezünk.

Ha $|Q| = n$ és $|\Sigma| = m$,, akkor a pszeudokód 2–6. soraiból látszik, hogy az algoritmus futási ideje legrosszabb esetben $O(n^2 m)$.

EPSZILON-MENTESÍTÉS$(\mathrm{A}, \overline{\mathrm{A}})$

1 $\overline{F} \leftarrow F \cup \{q \in I \mid \Lambda(q) \cap F \neq \emptyset\}$
2 **for** minden $q \in Q$
3 **do for** minden $a \in \Sigma$
4 **do** $\Delta \leftarrow \bigcup_{p \in \Lambda(q)} \delta(p, a)$
5 $\overline{\delta}(q, a) \leftarrow \Delta \cup \left(\bigcup_{p \in \Delta} \Lambda(p) \right)$
6 $\overline{E} \leftarrow \{(p, a, q), \mid p, q \in Q,\ a \in \Sigma, q \in \overline{\delta}(p, a)\}$

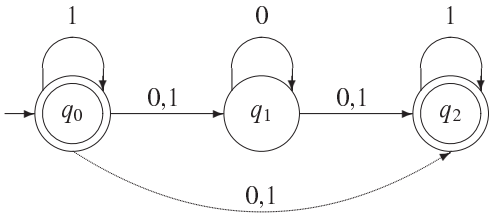
19.15. példa. Tekintsük a 19.11. ábrán lévő automatát, amelynek átmenettáblázata a következő:

δ	0	1	ε
q_0	$\emptyset$	$\{q_0\}$	$\{q_1\}$
q_1	$\{q_1\}$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\{q_2\}$	$\emptyset$

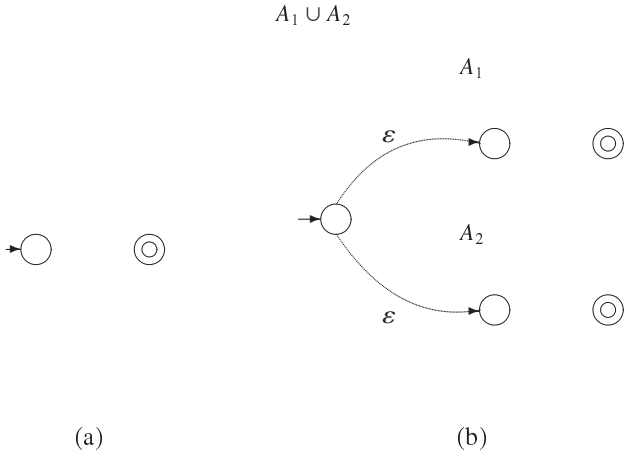
Alkalmazzuk az EPSZILON-MENTESÍTÉS algoritmust.
$\Lambda(q_0) = \{q_0, q_1, q_2\}$, $\Lambda(q_1) = \{q_1, q_2\}$, $\Lambda(q_2) = \{q_2\}$
$\Lambda(I) = \Lambda(q_0)$, és ennek metszete az F-fel nem üres, ezért $\overline{F} = F \cup \{q_0\} = \{q_0, q_2\}$.
$(q_0, 0)$:
$\Delta = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0) = \{q_1\}$, $\{q_1\} \cup \Lambda(q_1) = \{q_1, q_2\}$
$\overline{\delta}(q_0, 0) = \{q_1, q_2\}$.
$(q_0, 1)$:
$\Delta = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1) = \{q_0, q_2\}$, $\{q_0, q_2\} \cup (\Lambda(q_0) \cup \Lambda(q_2)) = \{q_0, q_1, q_2\}$
$\overline{\delta}(q_0, 1) = \{q_0, q_1, q_2\}$
$(q_1, 0)$:
$\Delta = \delta(q_1, 0) \cup \delta(q_2, 0) = \{q_1\}$, $\{q_1\} \cup \Lambda(q_1) = \{q_1, q_2\}$
$\overline{\delta}(q_1, 0) = \{q_1, q_2\}$
$(q_1, 1)$:
$\Delta = \delta(q_1, 1) \cup \delta(q_2, 1) = \{q_2\}$, $\{q_2\} \cup \Lambda(q_2) = \{q_2\}$
$\overline{\delta}(q_1, 1) = \{q_2\}$
$(q_1, 1) : \Delta = \delta(q_2, 0) = \emptyset$
$\overline{\delta}(q_2, 0) = \emptyset$
$(q_2, 1)$:
$\Delta = \delta(q_2, 1) = \{q_2\}$, $\{q_2\} \cup \Lambda(q_2) = \{q_2\}$
$\overline{\delta}(q_2, 1) = \{q_2\}$.
Tehát a $\overline{\mathrm{A}}$ véges automata átmenettáblázata a következő:

$\overline{\delta}$	0	1
q_0	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q_1	$\{q_1, q_2\}$	$\{q_2\}$
q_2	$\emptyset$	$\{q_2\}$

az átmenetdiagram pedig a 19.12. ábrán látható.



19.12. ábra. A 19.11. ábrán lévő ε-lépéses véges automatával ekvivalens ε-lépésmentes véges automata.



19.13. ábra. (a) Véges automata ábrázolása. Egy bemenő nyíl jelzi a kezdőállapotokat, míg két koncentrikus kör a végállapotokat. **(b)** Két véges automata egyesítése.

A következőkben értelmezzük a véges automatákon a reguláris műveleteket: egyesítés, szorzat, iteráció. Eredményül ε-lépéses automatát kapunk.

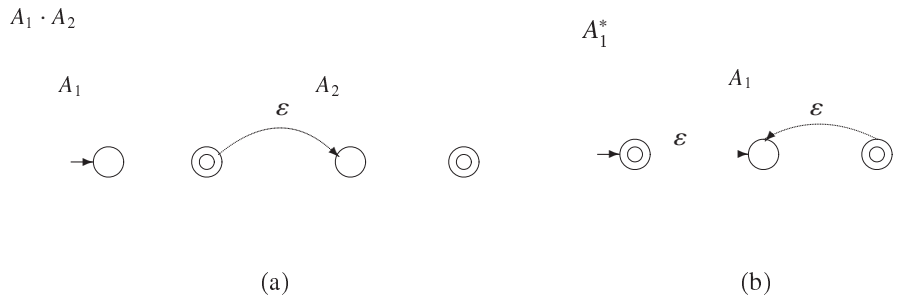
A műveleteket szemléletesen ábrákkal is megadjuk, egy véges automatát a 19.13(a) ábrán látható módon ábrázolunk. Egy nyíllal ellátott körrel jelöljük a kezdőállapotokat, és két koncentrikus körből álló jellel a végállapotokat.

Legyenek $\mathrm{A}_1 = (Q_1, \Sigma_1, E_1, I_1, F_1)$ és $\mathrm{A}_2 = (Q_2, \Sigma_2, E_2, I_2, F_2)$ véges automaták. A művelet eredménye az A-val jelölt $\mathrm{A} = (Q, \Sigma, E, I, F)$ ε-lépéses automata. Feltételezzük, hogy minden esetben $Q_1 \cap Q_2 = \emptyset$. Ha ez nem teljesül, akkor valamelyik állapothalmaz elemeit átnevezzük.

Egyesítés. $\mathrm{A} = \mathrm{A}_1 \cup \mathrm{A}_2$, ahol

$$
\begin{aligned}
&Q = Q_1 \cup Q_2 \cup \{q_0\},\\
&\Sigma = \Sigma_1 \cup \Sigma_2,\\
&I = \{q_0\},\\
&F = F_1 \cup F_2,\\
&E = E_1 \cup E_2 \cup \bigcup_{q \in I_1 \cup I_2} \{(q_0, \varepsilon, q)\}.
\end{aligned}
$$

Az eredményautomata a 19.13(b) ábrán látható. Ugyanazt az eredményt kapjuk, ha



19.14. ábra. (a) Két véges automata szorzata. **(b)** Véges automata iteráltja.

kezdőállapot-halmaznak vesszük a $I_1 \cup I_2$ halmazt egy újabb kezdőállapot helyett. Ekkor egyáltalán nem lesznek ε-lépések. Az egyesítés definíciója alapján belátható, hogy $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$.

Szorzat. $\mathrm{A} = \mathrm{A}_1 \cdot \mathrm{A}_2$, ahol

$$\begin{aligned} Q &= Q_1 \cup Q_2, \\ \Sigma &= \Sigma_1 \cup \Sigma_2, \\ F &= F_2, \\ I &= I_1, \\ E &= E_1 \cup E_2 \cup \bigcup_{\substack{p \in F_1 \\ q \in I_2}} \{(p, \varepsilon, q)\} \end{aligned}$$

Az eredményautomata a 19.14(a) ábrán látható. Itt is beláthatjuk, hogy $L(A_1 \cdot A_2) = L(A_1)L(A_2)$.

Iteráció. $\mathrm{A} = \mathrm{A}_1{}^*$, ahol

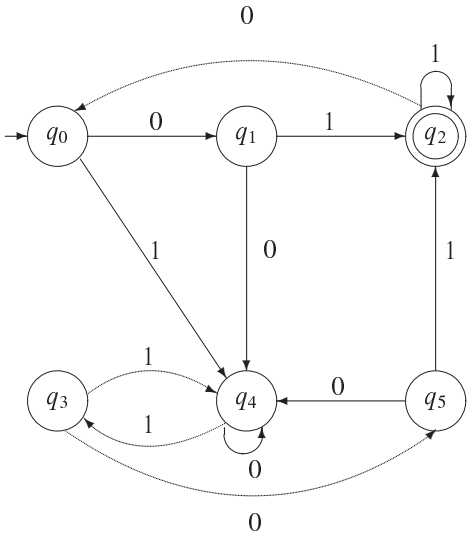
$$\begin{aligned} Q &= Q_1 \cup \{q_0\}, \\ \Sigma &= \Sigma_1, \\ F &= F_1 \cup \{q_0\}, \\ I &= \{q_0\} \\ E &= E_1 \cup \bigcup_{p \in I_1} \{(q_0, \varepsilon, p)\} \cup \bigcup_{\substack{q \in F_1 \\ p \in I_1}} \{(q, \varepsilon, p)\} \,. \end{aligned}$$

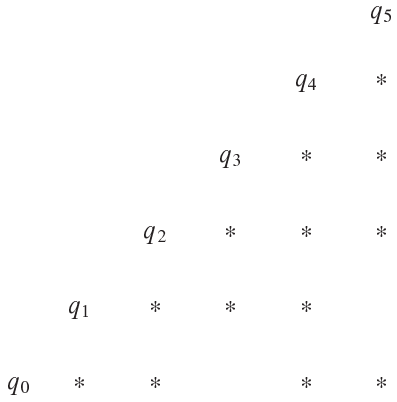
A véges automata iteráltja a 19.14(b) ábrán látható. Erre a műveletre az teljesül, hogy $L(A_1^*) = (L(A_1))^*$.

Az előbbiekben definiált három művelet segítségével újabb bizonyítását adtuk annak, hogy a reguláris nyelvek zártak a reguláris műveletekre nézve.

19.2.5. Determinisztikus véges automaták minimalizálása

Egy $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ teljes, determinisztikus véges automatát ***minimálisnak*** nevezünk, ha bármely vele ekvivalens $\mathrm{A}' = (Q', \Sigma, E', \{q_0'\}, F')$ teljes, determinisztikus véges automata esetében teljesül, hogy $|Q| \leq |Q'|$. A következőkben megadunk egy algoritmust, amely tetszőleges teljes, determinisztikus véges automatához megkonstruál egy vele ekvivalens minimális és teljes, determinisztikus véges automatát.





19.15. ábra. Véges automata minimalizálása.

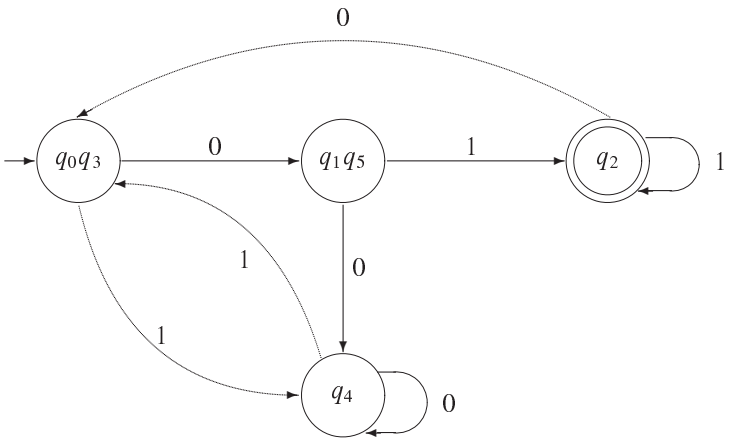
Az $A = (Q, \Sigma, E, \{q_0\}, F)$ determinisztikus véges automata p és q állapotát ***ekvivalensnek*** mondjuk, ha tetszőleges u szóra, mindkettőből végállapotba jutunk vagy egyikből sem jutunk végállapotba, azaz

$$p \equiv q \text{ ha minden } u \in \Sigma^* \text{ szóra } \begin{cases} p \xrightarrow{u} r,\ r \in F \text{ és } q \xrightarrow{u} s,\ s \in F \quad \text{vagy} \\ p \xrightarrow{u} r,\ r \notin F \text{ és } q \xrightarrow{u} s,\ s \notin F\ . \end{cases}$$

Ha két állapot nem ekvivalens, akkor azt mondjuk, hogy megkülönböztethetők. Az alábbi algoritmusban csillaggal jelöljük a megkülönböztethető állapotokat, az egymással ekvivalenseket pedig összevonjuk. Az algoritmus során bizonyos (nem rendezett) állapotpárokhoz állapotpárokból álló listát rendelünk a későbbi megcsillagozás reményében, azaz ha az algoritmus során egy állapotpárt megcsillagoztunk, akkor megcsillagozzuk a hozzárendelt lista összes elemét is. Az alábbi algoritmust olyan determinisztikus véges automatára alkalmazzuk, amelyből már kizártuk az elérhetetlen állapotokat. Mivel a véges automata determinisztikus és teljes, a $\delta(p, a)$ pontosan egy elemet tartalmaz, itt is alkalmazzuk a 913. oldalon definiált *elem* függvényt, amely az egyelemű halmaz egyetlen elemét adja vissza.

AUTOMATA-MINIMALIZÁLÁSA(A)

1 jelöljük meg egy-egy csillaggal az összes olyan $\{p, q\}$ állapotpárt, amelyre $p \in F$ és $q \notin F$ vagy fordítva.
2 minden jelöletlen $\{p, q\}$ állapotpárhoz rendeljünk egy üres listát,



19.16. ábra. A 19.15 ábrán látható determinisztikus véges automata minimalizált változata.

3 minden jelöletlen $\{p,q\}$ állapotpárra és minden $a \in \Sigma$ betűre
vizsgáljuk meg az $\{elem(\delta(p,a)), elem(\delta(q,a))\}$ állapotpárokat,
ha az így kapott állapotpárok közül valamelyik meg van csillagozva,
akkor csillagozzuk meg a $\{p,q\}$ párt is, egyetemben a már előzőleg a $\{p,q\}$ párhoz rendelt lista elemeivel,
különben, ha a fenti állapotpárok közül egy sincs megcsillagozva, akkor írjuk be a $\{p,q\}$ párt a $\{elem(\delta(p,a)), elem(\delta(q,a))\}$ párokhoz rendelt lista mindegyikébe, feltéve, hogy $\delta(p,a) \neq \delta(q,a)$,
4 vonjuk össze a megjelöletlen (ekvivalens) állapotokat

Az algoritmus befejeztével, ha a táblázatban egy cella nem tartalmaz csillagot, akkor a neki megfelelő sor és oszlop indexe két ekvivalens állapot, tehát összevonható. Az összevonást mindaddig folytatjuk, ameddig csak lehetséges. Általánosan, az ekvivalenciareláció az állapotok halmazát ekvivalenciaosztályokra bontja. Minden ilyen osztály állapotai egyetlen állapottá vonhatók össze.

Megjegyzés. Algoritmusunk abban az esetben is alkalmazható, ha a determinisztikus véges automata nem teljes, azaz vannak olyan állapotok, amelyekből adott bemeneti jelre nincs átmenet. Ekkor $\{\emptyset, \{q\}\}$ pár is előfordulhat, és ha q végállapot, úgy tekintjük, mintha ez a pár meg lenne csillagozva.

19.16. példa. Tekintsük a 19.15. ábrán látható determinisztikus véges automatát. Az algoritmus alkalmazásához egy táblázatot használunk, amelyben a csillagozást végezzük. A $\{p,q\}$ állapotpár megjelölését (megcsillagozását) a p sor és q oszlop (vagy q sor és p oszlop) találkozásnál elhelyezett csillag jelzi.

Először megcsillagozzuk a $\{q_2,q_0\}$, $\{q_2,q_1\}$, $\{q_2,q_3\}$, $\{q_2,q_4\}$ és $\{q_2,q_5\}$ párokat (mivel q_2 az egyetlen végállapot). Ezután sorra vesszük a csillaggal meg nem jelölt állapotokat, és az algoritmus szerint megvizsgáljuk őket. Kezdjük a $\{q_0,q_1\}$ párral. Hozzárendeljük a következő állapotpárokat: $\{elem(\delta(q_0,0)), elem(\delta(q_1,0))\}$, $\{elem(\delta(q_0,1)), elem(\delta(q_1,1))\}$, azaz $\{q_1,q_4\}$, $\{q_4,q_2\}$. Mivel $\{q_4,q_2\}$

már meg van jelölve, megjelöljük $\{q_0, q_1\}$-t is.

A $\{q_0, q_3\}$ pár esetében a két új pár $\{q_1, q_5\}$ és $\{q_4, q_4\}$. A $\{q_1, q_5\}$ párhoz hozzárendeljük egy listában a $\{q_0, q_3\}$-t, azaz

$$\{q_1, q_5\} \longrightarrow \{q_0, q_3\} .$$

Most $\{q_1, q_5\}$-tel folytatva, a $\{q_4, q_4\}$ és $\{q_2, q_2\}$ párokat kapjuk, amelyekhez az algoritmus szerint semmit sem rendelünk. Folytatjuk a $\{q_0, q_4\}$ párral. A hozzárendelt párok $\{q_1, q_4\}$ és $\{q_4, q_3\}$. Egyik sincs megcsillagozva, ezért hozzájuk rendeljük egy-egy listában a $\{q_0, q_4\}$ párt, azaz

$$\{q_1, q_4\} \longrightarrow \{q_0, q_4\}, \qquad \{q_4, q_3\} \longrightarrow \{q_0, q_4\} .$$

Most a $\{q_1, q_4\}$ párral folytatva a $\{q_4, q_4\}$, $\{q_2, q_3\}$ párokat kapjuk, és mivel ez utóbbi meg van jelölve csillaggal, megjelöljük a $\{q_1, q_4\}$ párt és a listában hozzárendelt $\{q_0, q_4\}$ párt is. Így folytatva, eljutunk a 19.15. ábrán látható táblázathoz, azaz azt kapjuk, hogy $q_0 \equiv q_3$ és $q_1 \equiv q_5$. Ezeket összevonva, a 19.16. ábrán látható determinisztikus véges automatát kapjuk, amelyik ekvivalens az eredetivel.

19.2.6. Pumpáló lemma reguláris nyelvekre

A következő tétel, amelyet *pumpáló lemmának* nevezünk, jól használható arra, hogy egy nyelvről bebizonyítsuk, hogy nem reguláris. Ez a tétel egy szükséges feltételt ad meg arra, hogy egy nyelv reguláris legyen.

19.15. tétel (pumpáló lemma). *Bármely L reguláris nyelv esetében létezik olyan $n \geq 1$ természetes szám (amely csak L-től függ), hogy L bármely legalább n hosszúságú u szava felírható $u = xyz$ alakban úgy, hogy*

(1) $|xy| \leq n$,

(2) $|y| \geq 1$,

(3) $xy^iz \in L$ *minden* $i = 0, 1, 2, \ldots$ *értékre.*

Bizonyítás. Ha L reguláris nyelv, akkor létezik olyan determinisztikus véges automata, amely felismeri az L nyelvet (19.12. és 19.10. tételek alapján). Legyen ez a determinisztikus véges automata $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$, tehát $L = L(\mathrm{A})$. Legyen n az automata állapotainak száma, azaz $|Q| = n$. Legyen $u = a_1 a_2 \ldots a_m \in L$ és $m \geq n$. Ekkor, mivel a determinisztikus véges automata az u szót felismeri, léteznek a $q_0, q_1, \ldots, q_m$ állapotok és a

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{m-1}} q_{m-1} \xrightarrow{a_m} q_m, \ q_m \in F$$

séta. Mivel összesen csak n állapotunk van, és $m \geq n$, a skatulya-elv[3] alapján a $q_0, q_1, \ldots, q_m$ állapotok között van legalább két megegyező (19.17. ábra). Legyen $q_j = q_k$, ahol $j < k$ és k a legkisebb ilyen index. Ekkor $j < k \leq n$. Bontsuk fel az u szót a következőképpen:

$x = a_1 a_2 \ldots a_j$

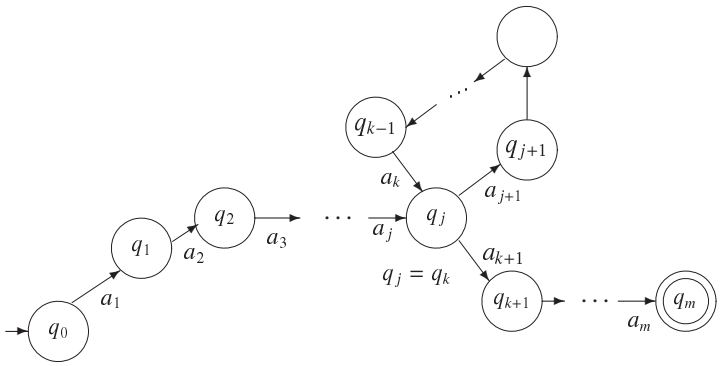
$y = a_{j+1} a_{j+2} \ldots a_k$

$z = a_{k+1} a_{k+2} \ldots a_m$.

Rögtön látszik, hogy $|xy| \leq n$ és $|y| \geq 1$. Bebizonyítjuk, hogy $xy^iz \in L$ tetszőleges i-re. Mivel $u = xyz \in L$, létezik a

$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_k \xrightarrow{z} q_m, \ q_m \in F$$

[3] *Skatulya-elv:* Ha k-nál több elemet k dobozba kell elhelyeznünk, akkor legalább egy dobozba egynél több elem kerül.



19.17. ábra. A pumpáló lemma bizonyításában használt determinisztikus véges automata rajza.

séta, és $q_j = q_k$ miatt felírható

$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_j \xrightarrow{z} q_m,\ q_m \in F\ ,$$

alakban is. Ebből következik, hogy a $q_j \xrightarrow{y} q_j$ séta elhagyható vagy többször is beilleszthető. Tehát léteznek a következő séták:

$$q_0 \xrightarrow{x} q_j \xrightarrow{z} q_m,\ q_m \in F\ ,$$

$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_j \xrightarrow{y} \ldots \xrightarrow{y} q_j \xrightarrow{z} q_m,\ q_m \in F\ .$$

Ebből következik, hogy $xy^iz \in L$ tetszőleges i-re, és ezzel bebizonyítottuk a lemmát. ■

19.17. példa. Bebizonyítjuk, hogy $L_1 = \{a^kb^k \mid k \geq 1\}$ nem reguláris. Tegyük fel, hogy L_1 reguláris, és legyen n a pumpáló lemma szerint az L_1-hez tartozó természetes szám. Mivel az $u = a^nb^n$ szó hossza $2n$, ezért ez a szó is felbontható a lemmában megadott módon. Bebizonyítjuk, hogy ez ellentmondáshoz vezet. Legyen ugyanis $u = xyz$ a felbontás. A lemma szerint ekkor $|xy| \leq n$, tehát x is és y is csak a-t tartalmazhatnak, és mivel $|y| \geq 1$, y legalább egy a-t tartalmaz. Ekkor xy^iz, $i \neq 1$-re, különböző számú a-t és b-t tartalmaz, tehát $xy^iz \notin L_1$ tetszőleges $i \neq 1$ értékre. Ez ellentmond a lemma állításának, tehát az a feltevésünk, hogy L_1 reguláris, hamis. Tehát $L_1 \notin \mathcal{L}_3$.

Mivel a $G_1 = (\{S\}, \{a, b\}, \{S \to ab, S \to aSb\}, S)$ környezetfüggetlen nyelvtan L_1-et generálja, így $L_1 \in \mathcal{L}_2$. E két állításból rögtön következik, hogy $\mathcal{L}_3 \subset \mathcal{L}_2$.

19.18. példa. Bebizonyítjuk, hogy $L_2 = \{u \in \{0, 1\}^* \mid n_0(u) = n_1(u)\}$ nem reguláris. ($n_0(u)$ az u-ban szereplő nullák, $n_1(u)$ pedig az 1-esek számát jelenti).

Az előbbi példához hasonlóan járunk el az $u = 0^n1^n$ szóval, ahol n most a pumpáló lemmában az L_2-höz tartozó természetes szám.

19.19. példa. Bebizonyítjuk, hogy $L_3 = \{uu \mid u \in \{a,b\}^*\}$ nem reguláris. Legyen $w = a^n ba^n b = xyz$, ahol n itt is a pumpáló lemma szerinti L_3-hoz tartozó természetes szám. Mivel $|xy| \leq n$, következik, hogy y csak a betűket tartalmazhat, és legalább egyet tartalmaz is. De ekkor a lemma szerint $xz \in L_3$, ami lehetetlen. Tehát L_3 nem reguláris.

A pumpáló lemmának több érdekes következménye van.

19.16. következmény. *Az L reguláris nyelv akkor és csakis akkor nem üres, ha létezik $u \in L$, $|u| < n$, ahol n a pumpáló lemmában az L-hez tartozó természetes szám.*

Bizonyítás. Az állítás egyik irányba nyilvánvaló: ha létezik n-nél rövidebb szó L-ben, akkor $L \neq \emptyset$. Fordítva, legyen $L \neq \emptyset$, és legyen u a legrövidebb szó L-ben. Megmutatjuk, hogy $|u| < n$. Ha ugyanis $|u| \geq n$, akkor alkalmazzuk a pumpáló lemmát, és azt kapjuk, hogy $u = xyz$, $|y| > 1$ és $xz \in L$. Ellentmondás, mivel $|xz| < |u|$, és u volt a legrövidebb L-beli szó. Tehát $|u| < n$. ■

19.17. következmény. *Létezik olyan algoritmus, amely eldönti, hogy egy reguláris nyelv üres-e.*

Bizonyítás. Tegyük fel, hogy $L = L(\mathrm{A})$, ahol $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ egy determinisztikus véges automata. A 19.16. következmény és a 19.15. tétel szerint L akkor és csakis akkor nem üres, ha tartalmaz n-nél rövidebb szót, ahol n az A automata állapotainak a száma. Következésképpen, elegendő azt eldönteni, hogy van-e olyan n-nél rövidebb szó, amelyet A elfogad. Mivel az n-nél rövidebb szavak száma véges, a kérdés algoritmikusan eldönthető. ■

Amikor a véges automaták elérhetetlen állapotainak meghatározására adtunk eljárást, akkor megjegyeztük, hogy az az eljárás használható annak eldöntésére is, hogy az automata által felismert nyelv üres-e. Mivel a véges automaták reguláris nyelveket ismernek fel, immár két eljárást ismerünk annak eldöntésére, hogy egy reguláris nyelv üres-e vagy sem. Sőt, van egy harmadik eljárásunk is, ha figyelembe vesszük, hogy a nem produktív állapotok kizárására szolgáló algoritmus is alkalmazható arra, hogy eldöntsük egy reguláris nyelvről, hogy üres-e vagy sem.

19.18. következmény. *Egy L reguláris nyelv akkor és csakis akkor végtelen, ha létezik $u \in L$ úgy, hogy $n \leq |u| < 2n$, ahol n a pumpáló lemmában az L-hez tartozó természetes szám.*

Bizonyítás. Ha L végtelen, akkor tartalmaz $2n$-nél hosszabb szót, és legyen u a legrövidebb, de $2n$-nél hosszabb L-beli szó. Mivel L reguláris, alkalmazható rá a pumpáló lemma, tehát $u = xyz$, ahol $|xy| \leq n$, tehát $|y| \leq n$ is igaz. A lemma szerint $u' = xz \in L$. Mivel $|u'| < |u|$, és a legrövidebb, de $2n$-nél hosszabb L-beli szó u, kapjuk, hogy $|u'| < 2n$. Másrészt $|y| \leq n$ miatt $|u'| \geq n$ is teljesül.

Fordítva, ha létezik $u \in L$ úgy, hogy $n \leq |u| < 2n$, akkor alkalmazva rá a pumpáló lemmát, következik, hogy $u = xyz$, $|y| \geq 1$ és $xy^i z \in L$ tetszőleges i-re, tehát L végtelen. ■

Feltehetjük a kérdést, hogy alkalmazhatjuk-e a pumpáló lemmát egy véges nyelvre , hisz a pumpálással végtelen sok szót kapunk? A válasz abban rejlik, hogy egy L, véges nyelvet felismerő bármely véges automata állapotainak száma nagyobb, mint L leghosszabb

$$
\begin{aligned}
x + y &\equiv y + x \\
(x + y) + z &\equiv x + (y + z) \\
(xy)z &\equiv x(yz) \\
(x + y)z &\equiv xz + yz \\
x(y + z) &\equiv xy + xz \\
(x + y)^* \equiv (x^* + y)^* &\equiv (x + y^*)^* \equiv (x^* + y^*)^* \\
(x + y)^* \equiv (xy^*)^* &\equiv (x^*y)^* \equiv (x^*y^*)^* \\
(x^*)^* &\equiv x^* \\
x^*x &\equiv xx^* \\
xx^* + \varepsilon &\equiv x^*
\end{aligned}
$$

19.1. táblázat. Reguláris kifejezések tulajdonságai.

szavának a hossza. Ezért L-ben egyetlen szó sincs, amelynek a hossza legalább n, ahol n a pumpáló lemmában az L nyelvhez tartozó természetes szám. Tehát egyetlen L-beli szó sem bontható fel xyz alakban, ahol $|xyz| \geq n$, $|xy| \leq n$, $|y| \geq 1$, és ezért nem kaphatunk végtelen sok további L-beli szót.

19.2.7. Reguláris kifejezések

A következőkben tetszőleges Σ ábécé esetén bevezetjük a Σ feletti reguláris kifejezés és az általa jelölt nyelv fogalmát. A reguláris kifejezés egy formula, míg az általa jelölt nyelv egy Σ feletti nyelv lesz. Például, ha $\Sigma = \{a, b\}$, akkor az a^*, b^*, $a^* + b^*$ kifejezések Σ feletti reguláris kifejezések lesznek, amelyek rendre az $\{a\}^*$, $\{b\}^*$, $\{a\}^* \cup \{b\}^*$ nyelveket jelölik. A pontos definíció a következő.

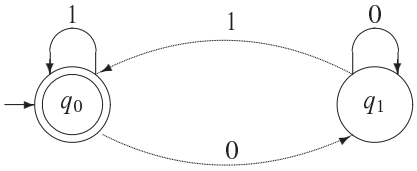
19.19. definíció. *Rekurzívan értelmezzük a Σ feletti reguláris kifejezés és az általa jelölt nyelv fogalmát.*

- *$\emptyset$ reguláris kifejezés és az üres nyelvet jelöli.*
- *ε reguláris kifejezés és az $\{\varepsilon\}$ nyelvet jelöli.*
- *Ha $a \in \Sigma$, a reguláris kifejezés és az $\{a\}$ nyelvet jelöli.*
- *Ha x, y reguláris kifejezések és az X, illetve Y nyelveket jelölik, akkor $(x + y)$, (xy), (x^*) is reguláris kifejezések és rendre az $X \cup Y$, XY és X^* nyelveket jelölik.*

Csak azok Σ feletti reguláris kifejezések, amelyeket a fenti szabályok véges sokszori alkalmazásával kapunk.

Egy reguláris kifejezésben bizonyos zárójeleket elhagyhatunk, amennyiben figyelembe véve a műveletek prioritási sorrendjét (iteráció, szorzat, egyesítés), nem változtatjuk meg az általa jelölt nyelvet. Például $((x^*)(x + y))$ helyett $x^*(x + y)$-t is írhatunk.

Két reguláris kifejezés ***ekvivalens,*** ha ugyanazt a nyelvet jelölik, azaz $x \equiv y$, ha $X = Y$, ahol X és Y rendre az x és y reguláris kifejezések által jelölt nyelvek. A 19.1. táblázatban néhány példát mutatunk ekvivalens reguláris kifejezésekre.



19.18. ábra. A 19.20. példában szereplő véges automata, amelyhez reguláris kifejezést rendelünk az 1. módszer alapján.

Megmutatjuk, hogy minden véges L nyelvhez megadható olyan x reguláris kifejezés, amely L-et jelöli. Ha $L = \emptyset$, akkor $x = \emptyset$. Ha $L = \{w_1, w_2, \ldots, w_n\}$, akkor $x = x_1 + x_2 + \ldots + x_n$, ahol minden $i = 1, 2, \ldots, n$ esetében x_i a $\{w_i\}$ nyelvet jelölő reguláris kifejezés. Ez utóbbit pedig a következőképpen adjuk meg. Ha $w_i = \varepsilon$, akkor $x_i = \varepsilon$. Különben, ha $w_i = a_1 a_2 \ldots a_m$, ahol $m \geq 1$ függ i-től, akkor az $x_i = a_1 a_2 \ldots a_m$, ahol elhagytuk a zárójeleket.

A következőkben bebizonyítjuk Kleene tételét, amely kapcsolatot teremt a reguláris nyelvek és a reguláris kifejezések között.

19.20. tétel (Kleene tétele). *Az $L \subseteq \Sigma^*$ nyelv pontosan akkor reguláris, ha van olyan Σ feletti reguláris kifejezés, amely éppen L-et jelöli.*

Bizonyítás. Először bebizonyítjuk, hogy ha x reguláris kifejezés, akkor az L nyelv, amelyet x jelöl, szintén reguláris. A bizonyítást a reguláris kifejezés felépítése szerinti indukcióval végezzük.

Ha $x = \emptyset$, $x = \varepsilon$, $x = a, \forall a \in \Sigma$, akkor $L = \emptyset$, $L = \{\varepsilon\}$, $L = \{a\}$. Mivel L mindhárom esetben véges, ezért reguláris.

Ha $x = (x_1 + x_2)$, akkor $L = L_1 \cup L_2$, ahol L_1 és L_2 rendre az x_1 és x_2 reguláris kifejezések által jelölt nyelvek. Az indukciós feltevésünk értelmében L_1 és L_2 reguláris nyelvek, így L is az, mivel a reguláris nyelvek osztálya zárt az egyesítésre. Az $x = (x_1 x_2)$ és $x = (x_1^*)$ esetek bizonyítása hasonlóan végezhető el.

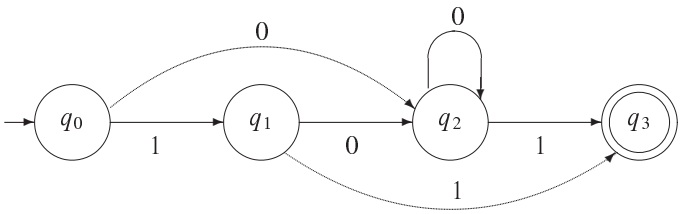
Fordítva, bebizonyítjuk, hogy ha L reguláris nyelv, akkor hozzárendelhető egy x reguláris kifejezés, amely éppen az L nyelvet jelöli. Ha L reguláris, akkor létezik egy $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ determinisztikus véges automata, amelyre $L = L(\mathrm{A})$. Legyenek A állapotai $q_0, q_1, \ldots, q_n$. Értelmezzük az R_{ij}^k nyelveket, minden $-1 \leq k \leq n$ és $0 \leq i, j \leq n$ értékekre. R_{ij}^k azon szavak halmaza, amelyek hatására az A véges automata a q_i állapotból a q_j állapotba kerül úgy, hogy közben nem használja a k-nál nagyobb indexű állapotokat. Az átmenetgráfot tekintve, egy szó akkor és csakis akkor tartozik R_{ij}^k-ba, ha a q_i állapotból indulva élek mentén eljuthatunk a q_j állapotba úgy, hogy az élek címkéit összeolvasva éppen a szót kapjuk, és közben nem érintjük a $q_{k+1}, \ldots q_n$ állapot egyikét sem. Az R_{ij}^k halmazokat formálisan is leírhatjuk:

$R_{ij}^{-1} = \{a \in \Sigma \mid (q_i, a, q_j) \in E\}$, ha $i \neq j$,

$R_{ii}^{-1} = \{a \in \Sigma \mid (q_i, a, q_i) \in E\} \cup \{\varepsilon\}$,

$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1} \left(R_{kk}^{k-1}\right)^* R_{kj}^{k-1}$ minden $i, j, k \in \{0, 1, \ldots, n\}$ értékre.

Indukcióval be lehet bizonyítani, hogy az R_{ij}^k halmazok leírhatók reguláris kifejezésekkel. Valóban, ha $k = -1$, akkor minden i-re és j-re az R_{ij}^k nyelv véges, és ezért megadható



19.19. ábra. A 19.21. példában szereplő véges automata, amelyhez reguláris kifejezést rendelünk az 1. módszer alapján. A számításokat a 19.2 táblázat tartalmazza.

olyan reguláris kifejezés, amely ezt a véges nyelvet jelöli. Továbbá, ha minden i-re és j-re az R_{ij}^{k-1} nyelv jelölhető reguláris kifejezéssel, akkor az R_{ij}^{k} nyelv is jelölhető reguláris kifejezéssel, amelyet az R_{ij}^{k-1}, R_{ik}^{k-1}, R_{kk}^{k-1} és R_{kj}^{k-1} nyelveket jelölő reguláris kifejezésekből építünk fel alkalmas módon, az R_{ij}^{k} fentebb definiált képlete alapján.

Végül, ha $F = \{q_{i_1}, q_{i_2}, \ldots, q_{i_p}\}$ az A véges automata végállapotainak halmaza, akkor $L = L(\mathrm{A}) = R_{0i_1}^{n} \cup R_{0i_2}^{n} \cup \ldots \cup R_{0i_p}^{n}$ is megadható reguláris kifejezéssel az $R_{0i_1}^{n}, R_{0i_2}^{n}, \ldots, R_{0i_p}^{n}$ nyelveket jelölő reguláris kifejezésekből a + művelet segítségével. ■

A következőkben eljárásokat adunk meg, amelyek segítségével tetszőleges reguláris kifejezéshez megadhatjuk a megfelelő véges automatát, és fordítva, tetszőleges véges automatához hozzárendelhetjük a megfelelő reguláris kifejezést.

Reguláris kifejezés hozzárendelése véges automatához

Három módszert mutatunk be, amelyek mindegyike tetszőleges véges automatához hozzárendeli a megfelelő reguláris kifejezést.

1. módszer. Felhasználjuk Kleene tételének az eredményét, azaz megkonstruáljuk az R_{ij}^{k} halmazokat, és felírjuk az $L = R_{0i_1}^{n} \cup R_{0i_2}^{n} \cup \ldots \cup R_{0i_p}^{n}$ nyelvet jelölő reguláris kifejezést, ahol $F = \{q_{i_1}, q_{i_2}, \ldots, q_{i_p}\}$ az automata végállapotainak halmaza.

19.20. példa. Tekintsük a 19.18. ábrán látható véges automatát.

$L(\mathrm{A}) = R_{00}^{1} = R_{00}^{0} \cup R_{01}^{0} \left(R_{11}^{0}\right)^{*} R_{10}^{0}$

R_{00}^{0} : $1^{*} + \varepsilon \equiv 1^{*}$

R_{01}^{0} : $1^{*}0$

R_{11}^{0} : $11^{*}0 + \varepsilon + 0 \equiv (11^{*} + \varepsilon)0 + \varepsilon \equiv 1^{*}0 + \varepsilon$

R_{10}^{0} : 11^{*}

Ekkor az $L(\mathrm{A})$-nak megfelelő reguláris kifejezés: $1^{*} + 1^{*}0(1^{*}0 + \varepsilon)^{*}11^{*} \equiv 1^{*} + 1^{*}0(1^{*}0)^{*}11^{*}$.

19.21. példa. Keressük meg a 19.19. ábrán lévő véges automatához rendelt reguláris kifejezést. A számításokat a 19.2. táblázat tartalmazza. Az R_{03}^{3}-nak megfelelő reguláris kifejezés: $11 + (0 + 10)0^{*}1$.

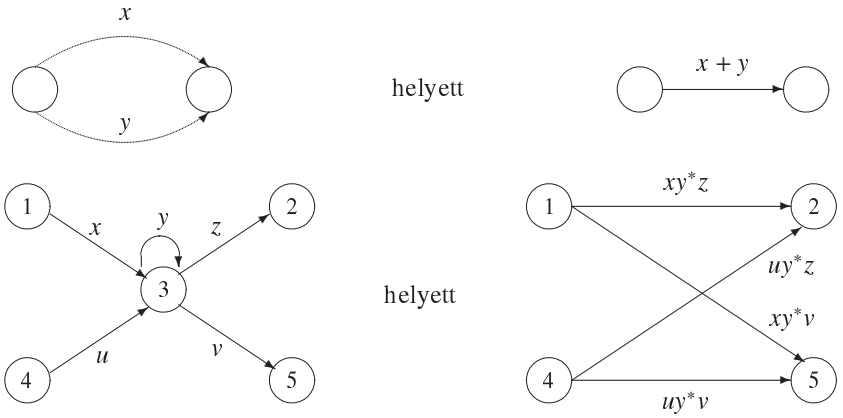
2. módszer. A véges automata fogalmát általánosítjuk úgy, hogy az automata gráfjának éleit nem betűkkel, hanem reguláris kifejezésekkel címkézzük meg. Egy ilyen automatában

	$k=-1$	$k=0$	$k=1$	$k=2$	$k=3$
R_{00}^k	ε	ε	ε	ε	
R_{01}^k	1	1	1	1	
R_{02}^k	0	0	$0+10$	$(0+10)0^*$	
R_{03}^k	$\emptyset$	$\emptyset$	11	$11+(0+10)0^*1$	$11+(0+10)0^*1$
R_{11}^k	ε	ε	ε	ε	
R_{12}^k	0	0	0	00^*	
R_{13}^k	1	1	1	$1+00^*1$	
R_{22}^k	$0+\varepsilon$	$0+\varepsilon$	$0+\varepsilon$	0^*	
R_{23}^k	1	1	1	0^*1	
R_{33}^k	ε	ε	ε	ε	

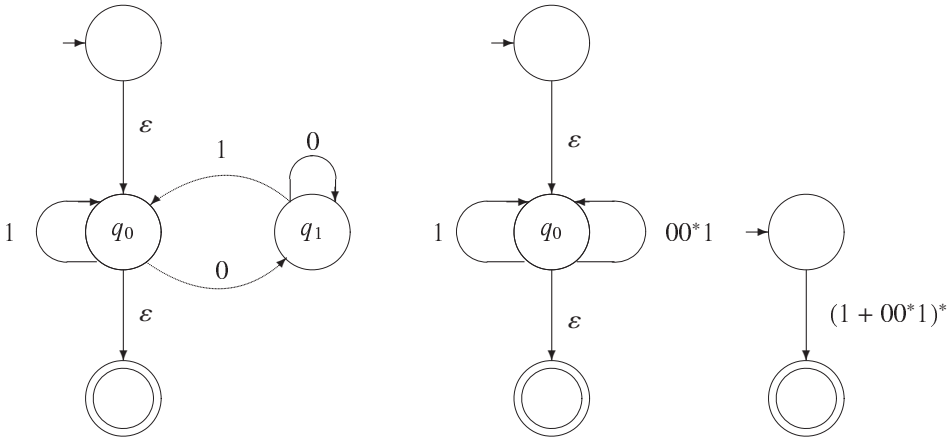
19.2. táblázat. A 19.19. ábra véges automatájához rendelt reguláris kifejezés meghatározása az R_{ij}^k halmazok segítségével.

minden séta meghatároz egy reguláris kifejezést, amely meghatároz egy reguláris nyelvet. Az általánosított véges automata által felismert nyelven a produktív séták által meghatározott reguláris nyelvek egyesítését értjük. Könnyen belátható, hogy az ilyen általánosított véges automatákkal is éppen a reguláris nyelvek ismerhetők fel.

Ugyanakkor az általánosított véges automaták előnye, hogy rajtuk ekvivalens átalakításokat végezhetünk, amelyek csökkentik az automata gráfja éleinek a számát, ugyanakkor nem változtatják meg az automata által felismert nyelvet. Végül elérhetjük, hogy az általánosított véges automata gráfjának egyetlen éle legyen, amelynek címkéje éppen az eredeti automata által felismert nyelvet jelölő reguláris kifejezés.



19.20. ábra. Lehetséges ekvivalens átalakítások véges automatához rendelhető reguláris kifejezés meghatározására.

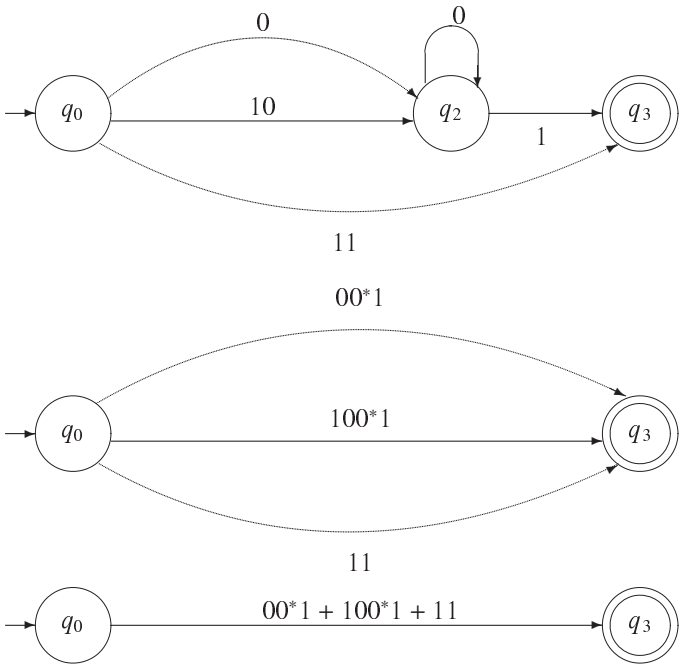


19.21. ábra. A 19.18. ábrán lévő véges automata átalakítása.

Az ekvivalens átalakítások a 19.20. ábrán láthatók. Amennyiben az ábrán látható 1, 2, 4, 5 pontok közül bármelyik kettő egybeesik, a végeredményben ezeket összevonjuk, így hurokél is megjelenik.

Először átalakítjuk a véges automatát megfelelő ε-átmenetek segítségével úgy, hogy egyetlen kezdő- és végállapota legyen. Ezután addig alkalmazzuk rá az ekvivalens átalakításokat, amíg gráfja egyetlen élt tartalmaz, amelynek címkéje az eredeti véges automata által felismert nyelvet jelölő reguláris kifejezés.

19.22. példa. A 19.18. ábra esetében a 19.21. ábrán látható lépesekkel jutunk el az eredményhez. Tehát az eredmény $(1 + 00^*1)^*$, amely, habár más alakú, de ugyanazt a nyelvet jelenti, mint az előbbi módszerrel kapott kifejezés (19.20. példa).



19.22. ábra. A 19.23. példa lépései.

19.23. példa. A 19.19. ábra esetében nem szükséges új kezdő- és végállapotot bevezetni. Az átalakítás lépései a 19.22. ábrán láthatók. A kapott reguláris kifejezés még így is írható: $(0+10)0^*1+11$, amely azonos az előbbi módszerrel kapott kifejezéssel.

3. módszer. Egy másik módszer a reguláris kifejezés felírására a formális egyenletek módszere. Minden állapothoz hozzárendelünk egy X változót (különböző állapotokhoz különbözőt) és egy egyenletet, amelynek bal oldalán X, jobb oldalán pedig Ya alakú kifejezések összege vagy ε állhatnak, ahol Y is egy állapothoz rendelt változó, a pedig egy bemeneti szimbólum. Ha az X változónak megfelelő állapotba nem vezet él, akkor az X baloldalú egyenlet jobb oldalán ε szerepel, különben az összes olyan Ya alakú tag összege, amelyekre teljesül, hogy a véges automata gráfjában az Y változónak megfelelő állapotból egy a-val címkézett él vezet az X változónak megfelelő állapotba. Amennyiben az X változónak megfelelő állapot kezdőállapot és egyben végállapot is, akkor az X baloldalú egyenlet jobb oldalán megjelenik egy ε tag is. Például a 19.19. ábra esetében legyenek ezek a változók X, Y, Z, U, amelyek a q_0, q_1, q_2, q_3 állapotoknak felelnek meg. A megfelelő egyenletek a következők:

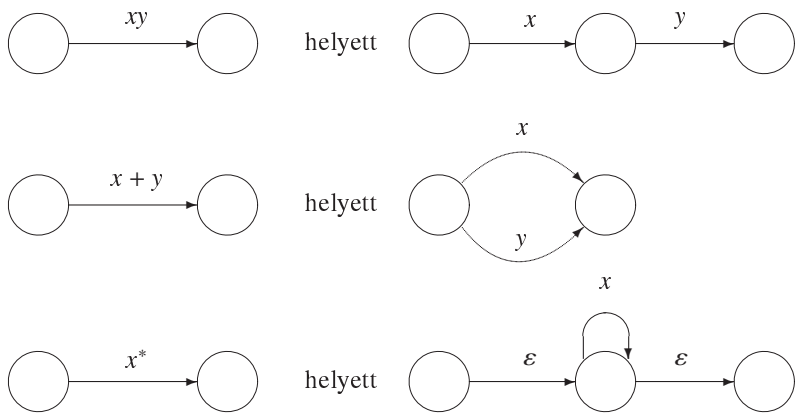
$X = \varepsilon$

$Y = X1$

$Z = X0 + Y0 + Z0$

$U = Y1 + Z1.$

Ha egy egyenlet $X = X\alpha + \beta$ alakú, ahol α, β tetszőleges szavak, amelyek nem tartal-



19.23. ábra. Lehetséges átalakítások reguláris kifejezéshez rendelt véges automata meghatározásához.

mazzák az X változót, akkor könnyű ellenőrizni, egyszerű behelyettesítéssel, hogy $X = \beta\alpha^*$ megoldása az egyenletnek.

Mivel az egyenletek lineárisak a változókban, minden egyenlet felírható $X = X\alpha + \beta$ vagy $X = X\alpha$ alakban, ahol α nem tartalmaz egyetlen változót sem. Ezt behelyettesítve a többi egyenletbe, eggyel csökkentjük azok számát. Így a rendszer, megfelelő helyettesítésekkel, megoldható minden változóra.

Az egyenletrendszert megoldva a végállapotoknak megfelelő változók adják a megoldást jelentő reguláris kifejezést úgy, hogy összeadjuk az ezen változóknak megfelelő reguláris kifejezéseket.

Példánkban, a fenti egyenletrendszer első egyenletének segítségével azt kapjuk, hogy $Y = 1$. Innen $Z = 0 + 10 + Z0$, azaz $Z = Z0 + (0 + 10)$, és ezt megoldva azt kapjuk, hogy $Z = (0 + 10)0^*$. Innen pedig U egyszerűen megkapható: $U = 11 + (0 + 10)0^*1$.

Ezt a módszert alkalmazva a 19.18. ábra esetében, a következő egyenletekhez jutunk:

$X = \varepsilon + X1 + Y1$

$Y = X0 + Y0$

Kiemelés után:

$X = \varepsilon + (X + Y)1$

$Y = (X + Y)0$.

A két egyenletet összeadva a következő egyenlethez jutunk:

$X + Y = \varepsilon + (X + Y)(0 + 1)$, ahonnan ($\varepsilon$-t β-nak, $(0 + 1)$-et α-nak tekintve) eredményül kapjuk a következőt:

$X + Y = (0 + 1)^*$.

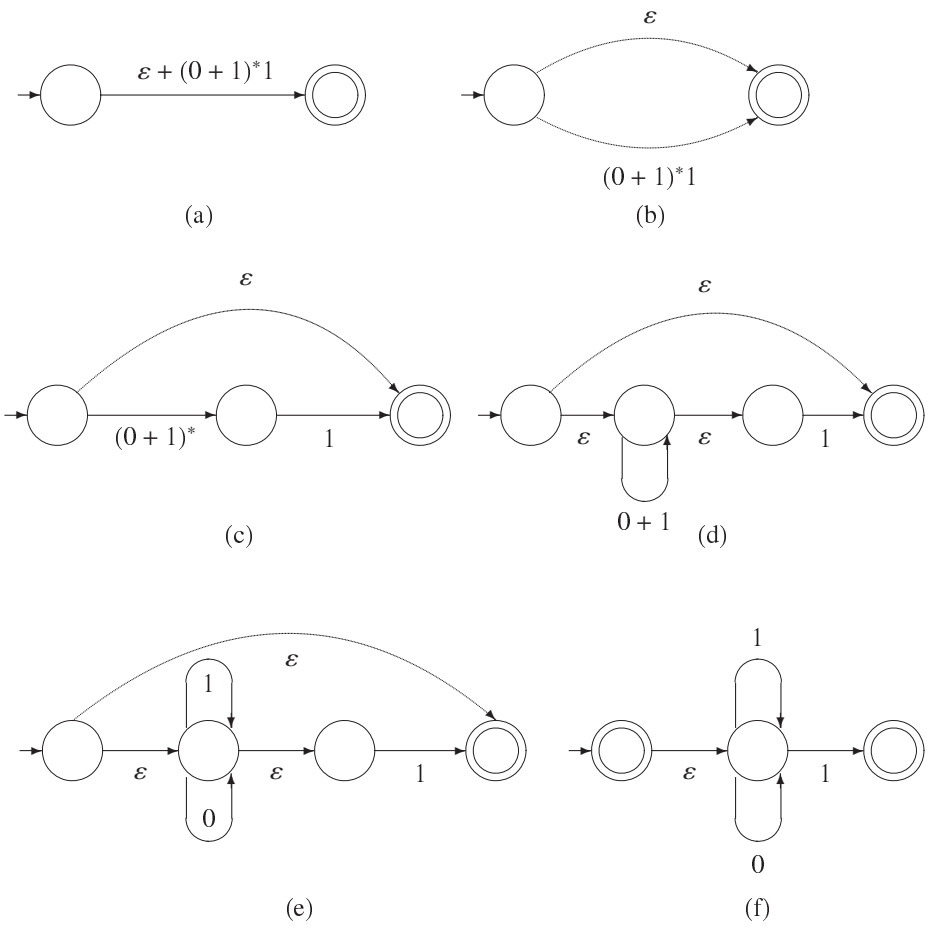
Innen – behelyettesítés után – megkapjuk X értékét:

$X = \varepsilon + (0 + 1)^*1$,

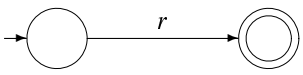
amely ekvivalens a másik módszerrel kapott kifejezéssel.

Véges automata hozzárendelése reguláris kifejezéshez

A r reguláris kifejezéshez hozzárendelünk egy általánosított véges automatát:

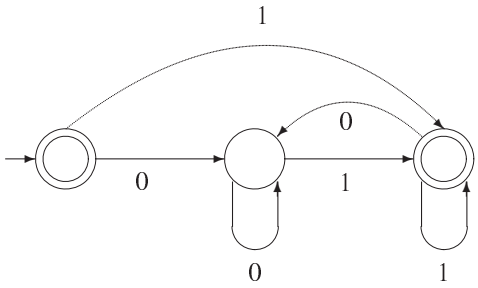


19.24. ábra. Véges automata hozzárendelése az $\varepsilon + (0 + 1)^*1$ reguláris kifejezéshez.

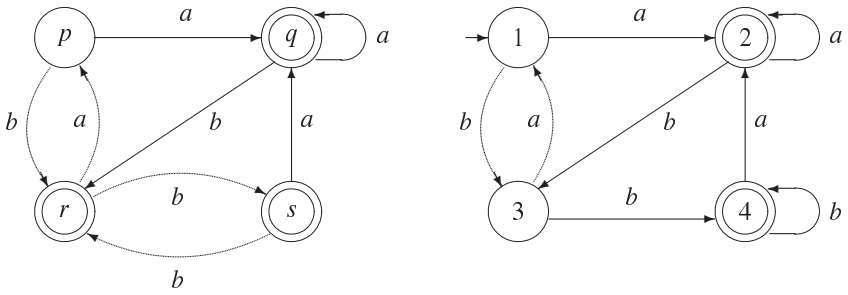


Azután lépésenként alkalmazzuk a 19.23. ábrán látható átalakításokat mindaddig, amíg a véges automata élei Σ elemeivel vagy ε-nal lesznek címkézve.

19.24. példa. Induljunk el az $\varepsilon + (0 + 1)^*1$ reguláris kifejezésből. Az átalakítás lépései a 19.24(a)-(e) ábrákon láthatók. Az utolsó véges automata (a 19.24(e) ábrán) egyszerűbb alakban is megadható, ez a 19.24(f) ábrán látható. Ha ebből kiküszöböljük a ε-lépést, átalakítjuk determinisztikussá, akkor a 19.25. ábrán látható véges automatát kapjuk eredményül, amelyről be lehet bizonyítani, hogy ekvivalens a 19.18. ábrán látható véges automatával.



19.25. ábra. Az $\varepsilon + (0 + 1)^*1$ kifejezéshez rendelt véges automata.



19.26. ábra. Minimalizálandó véges automaták a 19.2-5. gyakorlathoz.

Gyakorlatok

19.2-1. Adjunk meg egy determinisztikus véges automatát, amely a 9-cel osztható természetes számokat ismeri fel.

19.2-2. Adjunk meg egy-egy determinisztikus véges automatát, amely

a. a páros számú 0-t és páros számú 1-et tartalmazó szavakból álló nyelvet ismeri fel,

b. a páros számú 0-t és páratlan számú 1-et tartalmazó szavakból álló nyelvet ismeri fel,

c. a páratlan számú 0-t és páros számú 1-et tartalmazó szavakból álló nyelvet ismeri fel,

d. a páratlan számú 0-t és páratlan számú 1-et tartalmazó szavakból álló nyelvet ismeri fel.

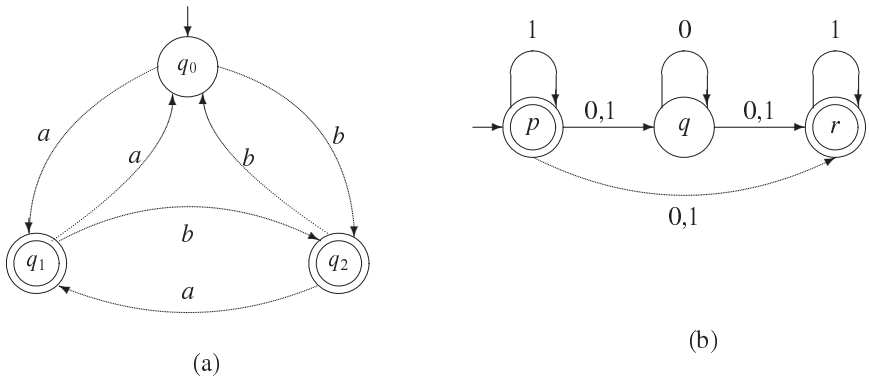
19.2-3. Adjunk meg egy-egy determinisztikus véges automatát a következő nyelvek felismerésére.

$L_1 = \{a^n b^m \mid n \geq 1, m \geq 0\}$, $L_2 = \{a^n b^m \mid n \geq 1, m \geq 1\}$,
$L_3 = \{a^n b^m \mid n \geq 0, m \geq 0\}$, $L_4 = \{a^n b^m \mid n \geq 0, m \geq 1\}$.

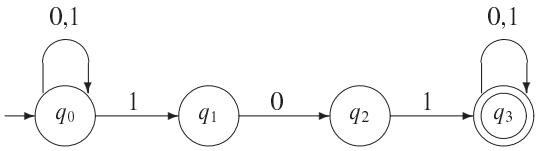
19.2-4. Adjunk meg egy nemdeterminisztikus véges automatát, amely a legalább két 0-át és tetszőleges számú 1-et tartalmazó szavakat ismeri fel. Adjunk meg egy vele ekvivalens determinisztikus véges automatát.

19.2-5. Minimalizáljuk a 19.26. ábrán lévő véges automatákat.

19.2-6. Mutassuk meg, hogy a 19.27.(a) ábrán látható véges automata minimális.



19.27. ábra. Véges automaták a 19.2-6. és 19.2-7. gyakorlatokhoz.



19.28. ábra. Véges automata a 19.2-9. gyakorlathoz.

19.2-7. Alakítsuk át a 19.27.(b) ábrán látható véges automatát determinisztikussá, majd minimalizáljuk.

19.2-8. Értelmezzük az A_1 véges automatát, amely felismeri a $0(10)^n$ alakú szavakat ($n \geq 0$), az A_2-t, amely pedig az $1(01)^n$ ($n \geq 0$) alakúakat. Adjuk meg az $A_1 \cup A_2$ egyesített véges automatát, majd küszöböljük ki az ε-lépéseket.

19.2-9. Rendeljünk a 19.28. ábrán látható véges automatához egy reguláris kifejezést.

19.2-10. Rendeljünk az $ab^*ba^* + b + ba^*a$ reguláris kifejezéshez egy véges automatát.

19.2-11. A pumpáló lemmát felhasználva bizonyítsuk be, hogy a következő nyelvek egyike sem reguláris:

$L_1 = \{a^n cb^n \mid n \geq 0\}$, $L_2 = \{a^n b^n a^n \mid n \geq 0\}$, $L_3 = \{a^p \mid p \text{ prím}\}$.

19.2-12. Bizonyítsuk be, hogy ha L reguláris nyelv, akkor az $\{u^{-1} \mid u \in L\}$ nyelv is reguláris.

19.2-13. Bizonyítsuk be, hogy ha $L \subseteq \Sigma^*$ reguláris nyelv, akkor regulárisak a következő nyelvek is:

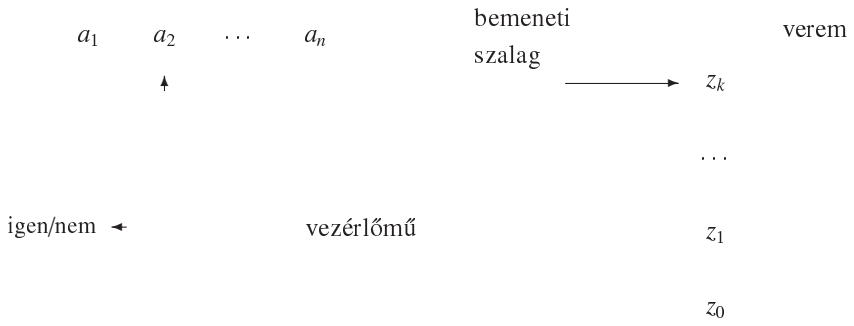
$\text{pre}(L) = \{w \in \Sigma^* \mid \exists u \in \Sigma^*, wu \in L\}$, $\text{suf}(L) = \{w \in \Sigma^* \mid \exists u \in \Sigma^*, uw \in L\}$.

19.2-14. Mutassuk meg, hogy az alábbi nyelvek mind regulárisak.

$L_1 = \{ab^n cd^m \mid n > 0, m > 0\}$,

$L_2 = \{(ab)^n \mid n \geq 0\}$,

$L_3 = \{a^{kn} \mid n \geq 0,\ k \text{ állandó}\}$.



19.29. ábra. Veremautomata.

19.3. Veremautomaták és környezetfüggetlen nyelvek

Ebben az alfejezetben veremautomatákkal és az általuk felismert nyelvek osztályával, a környezetfüggetlen nyelvekkel foglalkozunk.

Amint azt a 19.1. alfejezetben láttuk, egy környezetfüggetlen nyelvtan olyan $G = (N, T, P, S)$ nyelvtan, amelynek szabályai $A \to \beta$ alakúak, $A \in N$, $\beta \in (N \cup T)^+$. Megengedhető az $S \to \varepsilon$ szabály is, amennyiben S nem szerepel egyetlen szabály jobb oldalán sem. Az $L(G) = \{u \in T \mid S \overset{*}{\underset{G}{\Longrightarrow}} u\}$ nyelv a G nyelvtan által generált környezetfüggetlen nyelv.

19.3.1. Veremautomaták

Láttuk, hogy a véges automaták a reguláris nyelvek osztályát ismerik fel. A következőkben olyan automatákkal, az ún. ***veremautomatákkal*** ismerkedünk meg, amelyek környezetfüggetlen nyelveket ismernek fel. A veremautomaták lényegében abban különböznek a véges automatáktól, hogy egyrészt akkor is válthatnak állapotot, ha nem lépnek tovább a bemeneti szóban (üres jelet olvasnak), másrészt rendelkeznek egy veremmemóriával. A veremben az ún. veremábécé jeleit tárolhatja az automata (19.29. ábra).

A veremautomata egy szót kap bemenetként, a kezdőállapotból indul ki, és a veremmemóriájában (a továbbiakban csak veremben) egy speciális szimbólum, a verem kezdőszimbóluma áll. A működése során az aktuális állapot, a következő bemeneti szimbólum (amely üres szó is lehet), és a verem tetején levő szimbólum ismeretében állapotot vált, és a verem tetején levő szimbólum helyére egy szót ír be (amely szintén lehet üres is).

Kétféle felismerési mód lehetséges. Végállapottal való felismerésről beszélünk, ha azt követeljük meg, hogy a veremautomata a bemeneti szó elolvasása után végállapotba kerüljön. A másik felismerési mód, az üres veremmel való felismerés esetében azt követeljük meg, hogy a bemeneti szó elolvasásának pillanatában a verem üres legyen. Meg fogjuk mutatni, hogy a két felismerési mód ekvivalens egymással.

19.21. definíció. ***Nemdeterminisztikus veremautomatának*** *nevezzük a*

$$V = (Q, \Sigma, W, E, q_0, z_0, F)$$

rendezett hetest, ahol

- Q *az* ***állapotok*** *véges, nem üres halmaza,*
- Σ *a* ***bemeneti ábécé,***
- W *a* ***veremábécé,***
- $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times W \times W^* \times Q$ *az* ***átmenetek*** *vagy* ***élek*** *halmaza,*
- $q_0 \in Q$ *a* ***kezdőállapot,***
- $z_0 \in W$ *a* ***veremmemória kezdőjele,***
- $F \subseteq Q$ *a* ***végállapotok*** *halmaza.*

Egy (p, a, z, w, q) átmenet azt jelenti, hogy ha a V veremautomata a p állapotban van, a bemeneti szalagról az a jelet olvassa (amely most üres szó is lehet), és a verem tetején levő szimbólum z, akkor q állapotba megy át és verembe a z helyére a w szót írja (betűnként). A w szó beírása a verembe követi a természetes sorrendet, azaz w betűi balról jobbra, sorrendben kerülnek a verembe. A könnyebb olvashatóság kedvéért a (p, a, z, w, q) átmenet helyett a $(p, (a, z/w), q)$ jelölést használjuk, amely utal arra, hogy az a bemeneti jel elolvasása hatására a veremben kicseréljük z-t w-re.

Akárcsak a véges automaták esetében, most is értelmezhetünk egy átmenetfüggvényt a következőképpen:

$$\delta \ : \ Q \times (\Sigma \cup \{\varepsilon\}) \times W \to \mathcal{P}(W^* \times Q) \ ,$$

amely az aktuális állapothoz, bemeneti jelhez és verem tetején lévő elemhez hozzárendel (w, q) párokat, ahol $w \in W^*$ a verembe írt szó, $q \in Q$ pedig az új állapot.

Mivel a veremautomata nemdeterminisztikus, az átmenetfüggvény esetében

$\delta(q, a, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$ (ha az automata a bemeneti szalagról olvas egy jelet, majd az olvasófej továbblép), vagy

$\delta(q, \varepsilon, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$ (ha nem mozdul el az olvasófej a bemeneti szalagon).

A veremautomata ***determinisztikus,*** ha tetszőleges $q \in Q$ és $z \in W$ esetében

- $|\delta(q, a, z)| \leq 1, \forall a \in \Sigma \cup \{\varepsilon\}$
- Ha $\delta(q, \varepsilon, z) \neq \emptyset$, akkor $\delta(q, a, z) = \emptyset, \forall a \in \Sigma$.

A veremautomatához mindig megadható egy átmenettáblázat, akárcsak a véges automaták esetében. A táblázat sorai Q elemeivel vannak indexelve, oszlopai pedig a $\Sigma \cup \{\varepsilon\}$ és W elemeivel (minden $a \in \Sigma \cup \{\varepsilon\}$ és $z \in W$-nek megfelel egy oszlop). A $q \in Q$ állapotnak megfelelő sor és az $a \in \Sigma \cup \{\varepsilon\}$ és $z \in W$-nek megfelelő oszlop találkozásánál találhatók a $(w_1, p_1), \ldots, (w_k, p_k)$ párok, ha $\delta(q, a, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$.

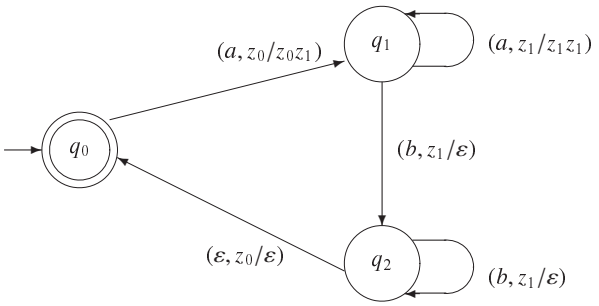
Ugyanakkor könnyen értelmezhetjük az átmenetgráfot is, amely csupán annyiban különbözik a véges automatáknál használt gráftól, hogy a $(p, (a, z/w), q)$ átmenetnek megfelelően a (p, q) élre $(a, z/w)$ kerül.

19.25. példa. $\mathrm{V}_1 = (\{q_0, q_1, q_2\}, \{a, b\}, \{z_0, z_1\}, E, q_0, z_0, \{q_0\})$. Az E halmaz elemei:

$(q_0, (a, z_0/z_0z_1), q_1)$
$(q_1, (a, z_1/z_1z_1), q_1)$ $\qquad$ $(q_1, (b, z_1/\varepsilon), q_2)$
$(q_2, (b, z_1/\varepsilon), q_2)$ $\qquad$ $(q_2, (\varepsilon, z_0/\varepsilon), q_0)$.

Az átmenetfüggvény:

$\delta(q_0, a, z_0) = \{(z_0z_1, q_1)\}$
$\delta(q_1, a, z_1) = \{(z_1z_1, q_1)\}$ $\qquad$ $\delta(q_1, b, z_1) = \{(\varepsilon, q_2)\}$
$\delta(q_2, b, z_1) = \{(\varepsilon, q_2)\}$ $\qquad$ $\delta(q_2, \varepsilon, z_0) = \{(\varepsilon, q_0)\}$.



19.30. ábra. Példa veremautomatára.

Az átmenettáblázat:

$\Sigma \cup \{\varepsilon\}$	a		b	ε
W	z_0	z_1	z_1	z_0
q_0	(z_0z_1, q_1)			
q_1		(z_1z_1, q_1)	(ε, q_2)	
q_2			(ε, q_2)	(ε, q_0)

Mivel az átmenetfüggvényben minden halmaz, amelyik nem üres, csak egy elemet tartalmaz (pl. $\delta(q_0, a, z_0) = \{(z_0z_1, q_1)\}$), a fenti átmenettáblázatban minden négyzetben csak egy elem szerepel, és nem használjuk a halmaz szokásos jelölését. Általában, ha egy halmaz egynél több elemet tartalmaz, akkor elemeit egymás alá írjuk. A veremautomata átmenetgráfja a 19.30. ábrán látható.

Az aktuális állapot, a bemeneti szó még el nem olvasott része és a verem tartalma együtt képezik a veremautomata egy ***konfigurációját***, vagyis minden $q \in Q$, $u \in \Sigma^*$, és $v \in W^*$ esetében (q, u, v) egy konfiguráció.

Ha $u = a_1a_2 \dots a_k$ és $v = x_1x_2 \dots x_m$, akkor a veremautomata kétféleképpen léphet (azaz konfigurációt válthat):

- $(q,\ a_1a_2 \dots a_k,\ x_1x_2 \dots x_{m-1}x_m) \Longrightarrow (p,\ a_2a_3 \dots a_k,\ x_1, x_2 \dots x_{m-1}w)$,
  ha $(q, (a_1, x_m/w), p) \in E$
- $(q,\ a_1a_2 \dots a_k,\ x_1x_2 \dots x_m) \Longrightarrow (p,\ a_1a_2 \dots a_k,\ x_1, x_2 \dots x_{m-1}w)$,
  ha $(q, (\varepsilon, x_m/w), p) \in E$.

A $\Longrightarrow$ reláció reflexív, tranzitív lezártját $\overset{*}{\Longrightarrow}$-gal jelöljük. A $\Longrightarrow$ jel helyett szokták még használni a $\vdash$ jelet is.

Az automata működése: elindulunk a $(q_0, a_1a_2 \dots a_n, z_0)$ kezdeti konfigurációból, majd meghatározzuk az összes lehetséges következő konfigurációt, majd ezekre a rákövetkezőket, és így tovább, ameddig lehet.

19.22. definíció. *Azt mondjuk, hogy a* V *veremautomata* ***végállapottal*** *felismer egy u szót, ha van* V*-beli konfigurációknak olyan sorozata, amelyre teljesülnek a következők:*

- *a sorozat első eleme* (q_0, u, z_0),
- *a sorozat minden eleméből van átmenet a sorozat következő elemébe, kivéve ha csak egy elemből áll,*
- *a sorozat utolsó eleme* (p, ε, w), *ahol* $p \in F$ *és* $w \in W^*$.

Tehát V akkor és csakis akkor ismeri fel egy u szót végállapottal, ha $(q_0, u, z_0) \overset{*}{\Longrightarrow} (p, \varepsilon, w)$, valamely $w \in W^*$-ra és $p \in F$-re. A V veremautomata által végállapotokkal felismert szavak halmazát a V által végállapotokkal felismert nyelvnek nevezzük, és $L(\mathrm{V})$-vel jelöljük.

19.23. definíció. *Azt mondjuk, hogy a* V *veremautomata* ***üres veremmel*** *felismer egy u szót, ha van* V*-beli konfigurációknak olyan sorozata, amelyre teljesülnek a következők:*

- *a sorozat első eleme* (q_0, u, z_0),
- *a sorozat minden eleméből van átmenet a sorozat következő elemébe,*
- *a sorozat utolsó eleme* $(p, \varepsilon, \varepsilon)$, *és p tetszőleges állapot.*

Tehát V akkor és csakis akkor ismer fel egy u szót üres veremmel, ha $(q_0, u, z_0) \overset{*}{\Longrightarrow} (p, \varepsilon, \varepsilon)$ valamely $p \in Q$-ra. A V veremautomata által üres veremmel felismert szavak halmazát a V által üres veremmel felismert nyelvnek nevezzük és $L_\varepsilon(\mathrm{V})$-vel jelöljük.

19.26. példa. Ha a 19.25. példa V_1 automatáját megvizsgáljuk, észrevehetjük, hogy az $\{a^n b^n \mid n \geq 0\}$ nyelvet ismeri fel végállapotokkal. Végezzük el a levezetést a következő szavakra: *aaabbb* és *abab*.

Az a^3b^3 szót felismeri az automata, mivel:

$(q_0, aaabbb, z_0) \Longrightarrow (q_1, aabbb, z_0z_1) \Longrightarrow (q_1, abbb, z_0z_1z_1) \Longrightarrow (q_1, bbb, z_0z_1z_1z_1)$

$\Longrightarrow (q_2, bb, z_0z_1z_1) \Longrightarrow (q_2, b, z_0z_1) \Longrightarrow (q_2, \varepsilon, z_0) \Longrightarrow (q_0, \varepsilon, \varepsilon)$, és mivel q_0 végállapot, a veremautomata felismeri a szót. Ugyanakkor, mivel a verem kiürült, üres veremmel is felismeri.

Mivel a kezdőállapot egyben végállapot is, az üres szót is felismeri végállapottal, ellenben üres veremmel nem.

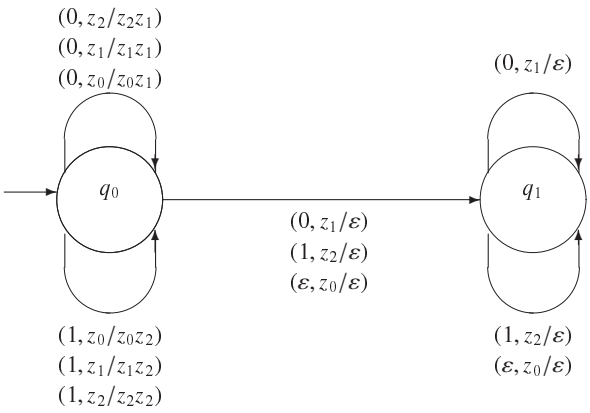
Hogy bebizonyítsuk, hogy az *abab* szót nem ismeri fel, szükségünk van az összes lehetőség megvizsgálására. Könnyű belátni, hogy ebben az esetben csak egyetlen lehetőség van:

$(q_0, abab, z_0) \Longrightarrow (q_1, bab, z_0z_1) \Longrightarrow (q_2, ab, z_0) \Longrightarrow (q_0, ab, \varepsilon)$, de innen nincs átmenet, tehát nem ismeri fel az *abab* szót.

19.27. példa. A $\mathrm{V}_2 = (\{q_0, q_1\}, \{0, 1\}, \{z_0, z_1, z_2\}, E, q_0, z_0, \emptyset)$ veremautomata átmenettáblázata:

$\Sigma \cup \{\varepsilon\}$	0			1			ε
W	z_0	z_1	z_2	z_0	z_1	z_2	z_0
q_0	(z_0z_1, q_0)	(z_1z_1, q_0) (ε, q_1)	(z_2z_1, q_0)	(z_0z_2, q_0)	(z_1z_2, q_0)	(z_2z_2, q_0) (ε, q_1)	(ε, q_1)
q_1		(ε, q_1)				(ε, q_1)	(ε, q_1)

Az átmenetgráf a 19.31. ábrán látható. A V_2 veremautomata az $\{uu^{-1} \mid u \in \{0, 1\}^*\}$ nyelvet ismeri fel.



19.31. ábra. A 19.27. példa átmenetgráfja.

Mivel V_2 nemdeterminisztikus, a (q_0, u, z_0) kezdőkonfigurációból elérhető összes konfigurációt egy ún. számítási fában tudjuk ábrázolni. Például a $(q_0, 1001, z_0)$ kezdőkonfigurációhoz tartozó számítási fa a 19.32. ábrán látható. A számítási fából megállapíthatjuk, hogy mivel $(q_1, \varepsilon, \varepsilon)$ a fa egyik levele, a V_2 verematomata üres veremmel felismeri az 1001 szót. A 19.33. ábrán látható számítási fa annak bizonyítéka, hogy a V_2 verematomata nem ismeri fel az 101 szót. A levelekben lévő konfigurációkat nem lehet folytatni, és egyik sem $(q, \varepsilon, \varepsilon)$ alakú.

19.24. tétel. *Egy L nyelv akkor és csakis akkor ismerhető fel valamely V_1 nemdeterminisztikus verematomatával üres veremmel, ha felismerhető valamely V_2 nemdeterminisztikus verematomatával végállapotokkal.*

Bizonyítás. *a)* Legyen $V_1 = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$ verematomata, amely üres veremmel ismeri fel az L nyelvet. Definiáljuk a $V_2 = (Q \cup \{p_0, p\}, \Sigma, W \cup \{x\}, E', p_0, x, \{p\})$ verematomatát, ahol $p, p_0 \notin Q$, , $x \notin W$ és

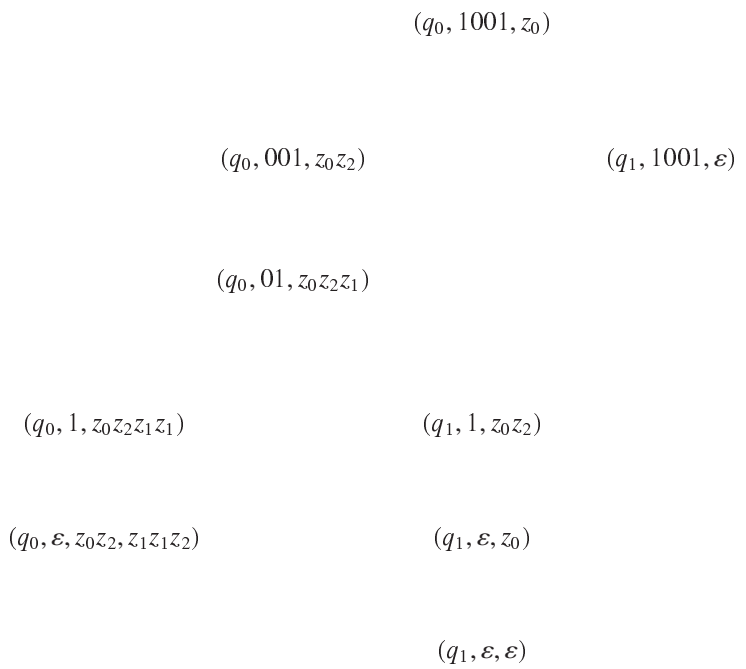
$$E' = E \cup \Big\{\big(p_0, (\varepsilon, x/xz_0), q_0\big)\Big\} \cup \Big\{\big(q, (\varepsilon, x/\varepsilon), p\big) \;\Big|\; q \in Q\Big\}$$

V_2 működése: V_2 először egy ε-lépéssel átmegy a V_1 kezdőállapotába, beírva a verembe x mellé z_0-t, V_1 kezdőszimbólumát. Ettől kezdve úgy működik, mint V_1. Ha V_1 egy adott szóra kiüríti a saját vermét, akkor V_2-nél még mindig marad egy x a veremben, amelyet V_2 ε-lépéssel töröl és végállapotba kerül. V_2 csak akkor kerülhet végállapotba, ha V_1 kiürítette a vermét.

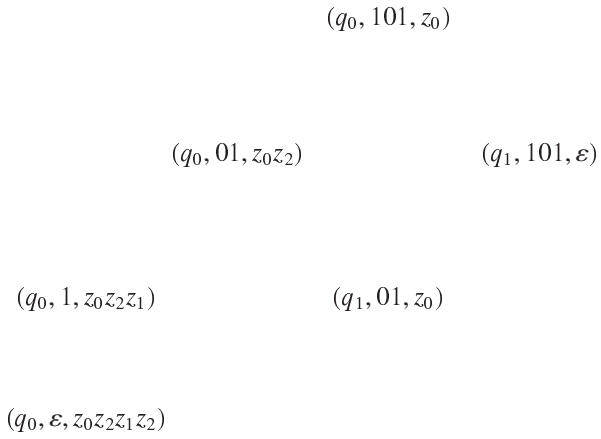
b) Legyen $V_2 = (Q, \Sigma, W, E, q_0, z_0, F)$ egy verematomata, amely az L nyelvet végállapotokkal ismeri fel. Definiáljuk a $V_1 = (Q \cup \{p_0, p\}, \Sigma, W \cup \{x\}, E', p_0, x, \emptyset)$ verematomatát, ahol $p_0, p \notin Q$, $x \notin W$ és

$$E' = E \cup \Big\{\big(p_0, (\varepsilon, x/xz_0), q_0\big)\Big\} \cup \Big\{\big(q, (\varepsilon, z/\varepsilon), p\big) \;\Big|\; q \in F, p \in Q, z \in W\Big\}$$
$$\cup \Big\{\big(p, (\varepsilon, z/\varepsilon), p\big) \;\Big|\; p \in Q, z \in W \cup \{x\}\Big\}$$

V_1 működése: V_1 először ε-lépéssel beírja a verembe x mellé z_0-t, V_2 vermének kezdőszimbólumát, ettől kezdve mint V_2 működik, azaz végállapotba jut minden felismert szóra.



19.32. ábra. Az 1001 szó felismerését bizonyító számítási fa (19.27. példa).



19.33. ábra. Számítási fa annak bizonyítására, hogy a 19.27. példa veremautomatája nem ismeri fel az 101 szót.

Innen V_1 ε-lépéssel kiüríti a vermet. V_1 csak akkor ürítheti ki a vermet, ha V_2 végállapotba kerül. ■

A következő két tétel azt bizonyítja, hogy a nemdeterminisztikus veremautomaták által felismert nyelvek halmaza éppen a környezetfüggetlen nyelvek halmaza.

19.25. tétel. *Ha G környezetfüggetlen nyelvtan, akkor létezik egy olyan* V *nemdeterminisztikus verematomata, amely üres veremmel felismeri az $L(G)$ nyelvet, azaz $L_\varepsilon(\mathrm{V}) = L(G)$.*

Csak a bizonyítás ötletét adjuk meg. Legyen $G = (N, T, P, S)$ egy környezetfüggetlen nyelvtan. Értelmezzük a $\mathrm{V} = (\{q\}, T, N \cup T, E, q, S, \emptyset)$ verematomatát, ahol $q \notin N \cup T$, az E átmenethalmaz értelmezése pedig:

- Ha létezik a G nyelvtan szabályai között $A \to \alpha$ szabály, akkor vegyük be E-be a $(q, (\varepsilon, A/\alpha^{-1}), q)$ átmenetet,
- Minden $a \in T$ jelre vegyük be E-be a $(q, (a, a/\varepsilon), q)$ átmenetet.

Ha van $S \to \alpha$ szabály G-ben, az automata egy ε-lépéssel beírja a verembe az α tükörképét. Ha a beolvasott betű egyezik a verem tetején lévővel, akkor törli azt a veremből. Ha a verem tetején az A nemterminális betű van, akkor beviszi a verembe valamelyik A-val kezdődő szabály jobb oldalának a tükörképét. Ha a szó beolvasása végén a verem kiürül, a verematomata felismerte a szót.

A következő algoritmus egy $G = (N, T, P, S)$ környezetfüggetlen nyelvtanhoz megkonstruálja azt a $\mathrm{V} = (\{q\}, T, N \cup T, E, q, S, \emptyset)$ verematomatát, amelyik üres veremmel felismeri a G által generált nyelvet.

KÖRNYEZETFÜGGETLEN-NYELVTANBÓL-VEREMAUTOMATA(G, V)

1 **for** minden $A \to \alpha$ szabályra
2 **do** vegyük be E-be a $(q, (\varepsilon, A/\alpha^{-1}), q)$ átmenetet
3 **for** minden $a \in T$ terminálisra
4 **do** vegyük be E-be a $(q, (a, a/\varepsilon), q)$ átmenetet

Ha a G nyelvtan szabályainak száma n, a terminális betűké pedig m, akkor az algoritmus lépésszáma $\Theta(n + m)$,

19.28. példa. Legyen $G = (\{S, A\}, \{a, b\}, \{S \to \varepsilon,\ S \to ab,\ S \to aAb,\ A \to aAb,\ A \to ab\}, S)$. Ekkor $\mathrm{V} = (\{q\}, \{a, b\}, \{a, b, A, S\}, E, q, S, \emptyset)$, a következő átmenettáblázattal.

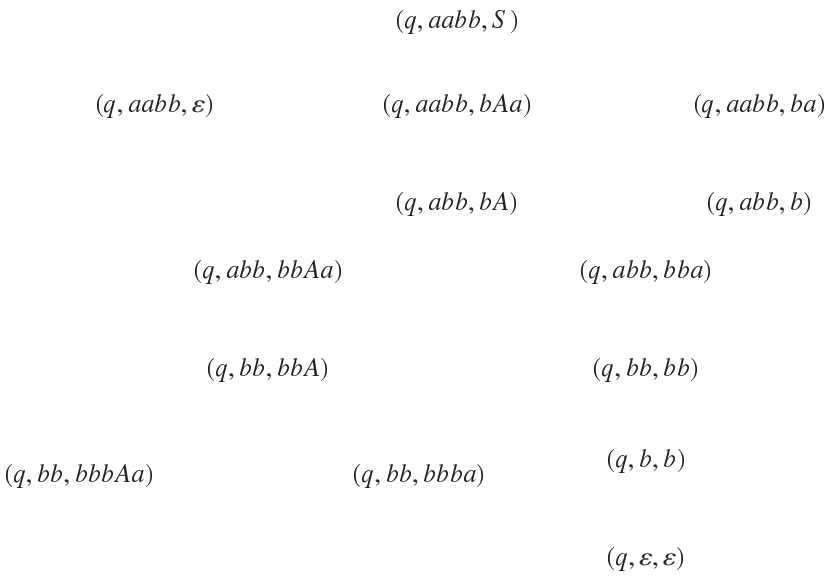
$\Sigma \cup \{\varepsilon\}$	a	b	ε	
W	a	b	S	A
q	(ε, q)	(ε, q)	(ε, q) (ba, q) (bAa, q)	(bAa, q) (ba, q)

Nézzük meg, hogyan ismeri fel a V verematomata az *aabb* szót, amelyet a G nyelvtanban a következőképpen lehet levezetni.

$$S \Longrightarrow aAb \Longrightarrow aabb,$$

ahol alkalmaztuk az $S \to aAb$ és $A \to ab$ szabályokat. A felismerés üres veremmel történik (19.34. ábra).

19.26. tétel. *Ha* V *nemdeterminisztikus verematomata, akkor létezik egy olyan G környezetfüggetlen nyelvtan, hogy* V *üres veremmel felismeri az $L(G)$ nyelvet, azaz $L_\varepsilon(\mathrm{V}) = L(G)$.*



19.34. ábra. Szó felismerése üres veremmel (19.28. példa).

Bizonyítás helyett megadjuk, hogyan kell definiálni a G nyelvtant. Legyen a nemdeterminisztikus veremautomata $\mathrm{V} = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$.

Ekkor $G = (N, T, P, S)$, ahol

$N = \{S\} \cup \{S_{p,z,q} \mid p, q \in Q, z \in W\}$ és $T = \Sigma$.

A P szabályait pedig a következőképpen kapjuk meg.

• Minden q állapotra vegyük be P-be az $S \to S_{q_0,z_0,q}$ szabályt.

• Ha $(q, (a, z/z_k \dots z_2 z_1), p) \in E$, ahol $q \in Q$, $z, z_1, z_2, \dots z_k \in W$ $(k \geq 1)$ és $a \in \Sigma \cup \{\varepsilon\}$, vegyük be P-be minden lehetséges $p_1, p_2, \dots, p_k$ állapotra az

$S_{q,z,p_k} \to a S_{p,z_1,p_1} S_{p_1,z_2,p_2} \dots S_{p_{k-1},z_k,p_k}$ szabályokat.

• Ha $(q, (a, z/\varepsilon), p) \in E$, ahol $p, q \in Q, z \in W$, és $a \in \Sigma \cup \{\varepsilon\}$, vegyük be P-be az

$S_{q,z,p} \to a$ szabályt.

Az így értelmezett nyelvtan kiterjesztett környezetfüggetlen nyelvtan, amelyhez, mint tudjuk, mindig hozzárendelhető egy vele ekvivalens környezetfüggetlen nyelvtan. A tétel bizonyítása azon alapszik, hogy minden konfigurációsorozatnak, amelynek segítségével a V veremautomata felismer egy szót, megfelel egy levezetés a G nyelvtanban. Azt, hogy ez levezetés éppen az adott szót generálja, az $S_{q,z,p_k} \to a S_{p,z_1,p_1} S_{p_1,z_2,p_2} \dots S_{p_{k-1},z_k,p_k}$ alakú szabályok biztosítják azáltal, hogy minden lehetséges $p_1, p_2, \dots, p_k$ állapotra értelmeztük őket. A 19.27. példa segítségével megmutatjuk, hogyan rendelhető hozzá egy levezetés az adott konfigurációsorozathoz. A példában értelmezett veremautomata a

$(q_0, 00, z_0) \Longrightarrow (q_0, 0, z_0 z_1) \Longrightarrow (q_1, \varepsilon, z_0) \Longrightarrow (q_1, \varepsilon, \varepsilon)$

konfigurációsorozattal ismeri fel a 00 szót, amely sorozat az

$(q_0, (0, z_0/z_0 z_1), q_0)$,

$(q_0, (0, z_1/\varepsilon), q_1)$,

$(q_1, (\varepsilon, z_1/\varepsilon), q_1)$.

átmeneteken alapszik.

A G nyelvtan értelmezése szerint ezeknek rendre megfelelnek a következő helyettesítési szabályok

(1) $S_{q_0,z_0,p_2} \longrightarrow 0S_{q_0,z_1,p_1}S_{p_1,z_0,p_2}$ minden $p_1, p_2 \in Q$ állapotra,
(2) $S_{q_0,z_1,q_1} \longrightarrow 0$,
(3) $S_{q_1,z_0,q_1} \longrightarrow \varepsilon$.

Továbbá, minden q állapotra definiáltuk az $S \longrightarrow S_{q_0,z_0,q}$ szabályokat is.

Az $S \longrightarrow S_{q_0,z_0,q}$ szabály alapján létezik az $S \Longrightarrow S_{q_0,z_0,q}$ levezetés ahol q tetszőlegesen választható. Válasszuk a fenti (1) szabályban p_2-t is q-nak. Ekkor létezik az

$$S \Longrightarrow S_{q_0,z_0,q} \Longrightarrow 0S_{q_0,z_1,p_1}S_{p_1,z_0,q}$$

levezetés is, ahol $p_1 \in Q$ tetszőlegesen megválasztható. Ha $p_1 = q_1$, akkor az

$$S \Longrightarrow S_{q_0,z_0,q} \Longrightarrow 0S_{q_0,z_1,q_1}S_{q_1,z_0,q} \Longrightarrow 00S_{q_1,z_0,q}$$

levezetést kapjuk. Most q-t választhatjuk q_1-nek, és ekkor

$$S \Longrightarrow S_{q_0,z_0,q_1} \Longrightarrow 0S_{q_0,z_1,q_1}S_{q_1,z_0,q_1} \Longrightarrow 00S_{q_1,z_0,q_1} \Longrightarrow 00,$$

amely azt bizonyítja, hogy a 00 szó levezethető a fent definiált nyelvtanban.

A következő algoritmus egy $\mathrm{V} = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$ veremautomatához megkonstruálja azt a $G = (N, T, P, S)$ környezetfüggetlen nyelvtant, amely a V veremautomata által üres veremmel felismert környezetfüggetlen nyelvet generálja.

VEREMAUTOMATÁBÓL-KÖRNYEZETFÜGGETLEN-NYELVTAN(V, G)

1 **for** minden $q \in Q$
2 **do** vegyük be P-be az $S \to S_{q_0,z_0,q}$ szabályt
3 **for** minden $(q, (a, z/z_k \ldots z_2 z_1), p) \in E$ ▷ $q \in Q$, $z, z_1, z_2, \ldots z_k \in W$ $(k \geq 1)$, $a \in \Sigma \cup \{\varepsilon\}$
4 **do for** minden $p_1, p_2, \ldots, p_k$ állapotra
5 **do** vegyük be P-be az $S_{q,z,p_k} \to aS_{p,z_1,p_1}S_{p_1,z_2,p_2} \ldots S_{p_{k-1},z_k,p_k}$ szabályokat
6 **for** minden $(q(a, z/\varepsilon), p) \in E$ ▷ $p, q \in Q, z \in W$, $a \in \Sigma \cup \{\varepsilon\}$
7 **do** vegyük be P-be az $S_{q,z,p} \to a$ szabályt

Ha az automata állapotainak száma n, átmeneteinek száma pedig m, akkor a fenti algoritmus legfeljebb $n + mn + m$ lépést hajt végre, tehát a lépésszáma legrosszabb esetben $O(nm)$.

Végül bizonyítás nélkül megemlítjük, hogy a determinisztikus veremautomatákkal felismerhető nyelvek osztálya valódi része a nemdeterminisztikus veremautomatákkal felismerhető nyelvek osztályának. Ebből a szempontból a veremautomaták a véges automatáktól eltérő módon viselkednek.

19.29. példa. Példaként, tekintsük az előbbi példában (19.28. példa) meghatározott V veremautomatát: $\mathrm{V} = (\{q\}, \{a, b\}, \{a, b, A, S\}, E, q, S, \emptyset)$. A G nyelvtan a következő:

$$G = (\{S, S_a, S_b, S_S, S_A, \}, \{a, b\}, P, S),$$

ahol minden $z \in \{a, b, S, A\}$ esetén S_z az $S_{q,z,q}$ jelölés rövidítése. Felírjuk az automata átmeneteit:

$$(q, (a, a/\varepsilon), q), \quad (q, (b, b/\varepsilon), q),$$
$$(q, (\varepsilon, S/\varepsilon), q), \quad (q, (\varepsilon, S/ba), q), \quad (q, (\varepsilon, S/bAa), q),$$
$$(q, (\varepsilon, A/ba), q), \quad (q, (\varepsilon, A/bAa), q).$$

Ezek alapján a következő szabályokat definiálhatjuk:

$S \to S_S$

$S_a \to a$
$S_b \to b$
$S_S \to \varepsilon \mid S_aS_b \mid S_aS_AS_b$
$S_A \to S_aS_AS_b \mid S_aS_b$.

Könnyen belátható, hogy S_S-t kiiktathatjuk, így a szabályok, miután az elsőt elhagyjuk, a következő lesznek:

$S \to \varepsilon \mid S_aS_b \mid S_aS_AS_b$,
$S_A \to S_aS_AS_b \mid S_aS_b$,
$S_a \to a,\ S_b \to b$,

ezek a szabályok pedig helyettesíthetők a következőkkel:

$S \to \varepsilon \mid ab \mid aAb$,
$A \to aAb \mid ab$.

19.3.2. Környezetfüggetlen nyelvek

Vegyünk egy $G = (N, T, P, S)$ környezetfüggetlen nyelvtant. G egy ***levezetési fájának*** egy olyan véges, rendezett, címkézett fát nevezünk, amelynek gyökere az S kezdőszimbólummal van címkézve, minden belső csúcs címkéje egy nemterminális szimbólum, és levelei terminális szimbólumokkal vannak címkézve. Teljesül továbbá az a feltétel, hogy ha egy belső csúcs címkéje az A nemterminális, és a csúcsnak k közvetlen leszármazottja van, akkor P-ben van egy olyan $A \to a_1a_2\ldots a_k$ szabály, hogy ezek a közvetlen leszármazottak rendre az a_1, a_2, $\ldots a_k$ szimbólumokkal vannak címkézve. A levezetési fa ***eredménye*** az a T feletti szó, amelyet úgy kapunk, hogy a fa leveleinek címkéjét balról jobbra haladva összeolvassuk. A levezetési fát még ***szintaxisfának*** is nevezzük.

Tekintsük a $G = (\{S, A\}, \{a, b\}, \{S \to aA,\ S \to a,\ S \to \varepsilon,\ A \to aA,\ A \to aAb,\ A \to ab,\ A \to b\}, S)$ környezetfüggetlen nyelvtant. Ez a nyelvtan az $L(G) = \{a^nb^m \mid n \geq m \geq 0\}$ nyelvet generálja. Az $a^4b^2 \in L(G)$ szó levezetése a következő:

$$S \Longrightarrow aA \Longrightarrow aaA \Longrightarrow aaaAb \Longrightarrow aaaabb.$$

Ezt a levezetést ábrázolhatjuk a 19.35. ábrán lévő levezetési fával, amelynek eredménye az *aaaabb* szó.

Minden levezetéshez hozzárendelhető egy levezetési fa. Fordítva, egy levezetési fához több levezetés is rendelhető. Például, a 19.35. ábrán látható fához, a fenti levezetésen kívül a

$$S \Longrightarrow aA \Longrightarrow aaAb \Longrightarrow aaaAb \Longrightarrow aaaabb.$$

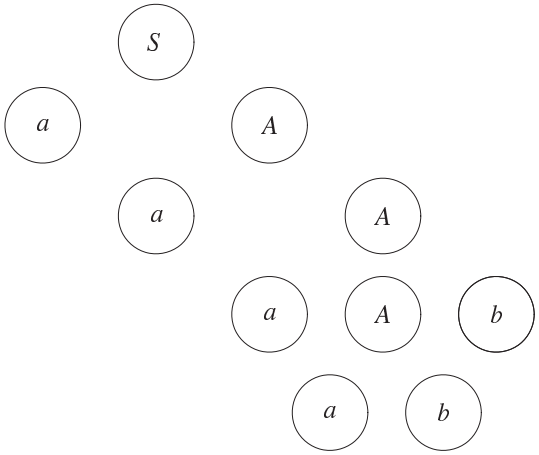
levezetés is hozzárendelhető.

19.27. definíció. *Az $\alpha_0 \Longrightarrow \alpha_1 \Longrightarrow \ldots \Longrightarrow \alpha_n$ levezetést **legbaloldalibb levezetésnek** nevezzük, ha minden $i = 1, 2, \ldots, n-1$ esetén van olyan $u_i \in T^*$, $\beta_i \in (N \cup T)^*$ szó és $(A_i \to \gamma_i) \in P$ szabály, hogy teljesülnek az*

$$\alpha_i = u_iA_i\beta_i \quad \text{és} \quad \alpha_{i+1} = u_i\gamma_i\beta_i$$

összefüggések.

Tekintsük a következő nyelvtant:
$G = (\{S, A\}, \{a, b, c\}, \{S \to bA,\ S \to bAS,\ S \to a,\ A \to cS,\ A \to a\}, S)$.



19.35. ábra. Az *aaaabb* szó levezetési fája.

Ebben a nyelvtanban a *bcbaa* szónak két, egymástól különböző legbaloldalibb levezetése van:

$S \Longrightarrow bA \Longrightarrow bcS \Longrightarrow bcbAS \Longrightarrow bcbaS \Longrightarrow bcbaa$,
$S \Longrightarrow bAS \Longrightarrow bcSS \Longrightarrow bcbAS \Longrightarrow bcbaS \Longrightarrow bcbaa$.

19.28. definíció. *Egy G környezetfüggetlen nyelvtan* ***nem egyértelmű,*** *ha $L(G)$-ben van olyan szó, amelynek egynél több legbaloldalibb levezetése van. Különben a G nyelvtan* ***egyértelmű****.*

Az előbbi G nyelvtan nem egyértelmű, mert a *bcbaa* szónak találtunk két legbaloldalibb levezetését. Egy nyelvet több nyelvtan is generálhat, és ezek között lehetnek egyértelműek és nem egyértelműek is. Egy környezetfüggetlen nyelv ***alapvetően nem egyértelmű,*** ha nem létezik egyetlen egyértelmű nyelvtan sem, amely generálja.

19.30. példa. Vizsgáljuk meg a következő két nyelvtant.

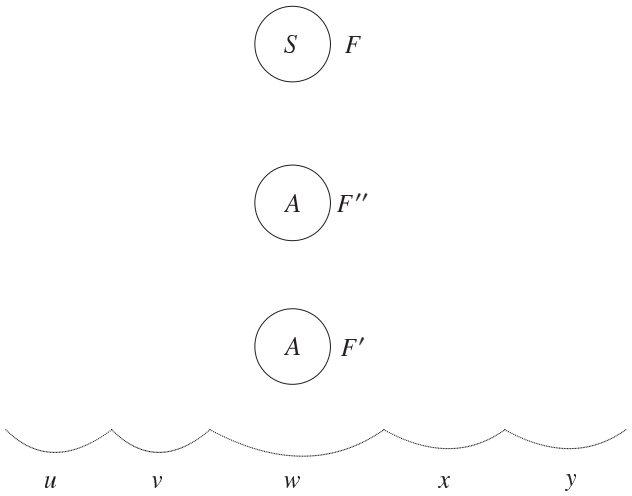
A $G_1 = (\{S\}, \{a, +, *\}, \{S \to S + S,\ S \to S * S,\ S \to a\}, S)$ nem egyértelmű, mert
$S \Longrightarrow S + S \Longrightarrow a + S \Longrightarrow a + S * S \Longrightarrow a + a * S \Longrightarrow a + a * S + S \Longrightarrow a + a * a + S$
$\Longrightarrow a + a * a + a$ és
$S \Longrightarrow S * S \Longrightarrow S + S * S \Longrightarrow a + S * S \Longrightarrow a + a * S \Longrightarrow a + a * S + S \Longrightarrow a + a * a + S$
$\Longrightarrow a + a * a + a$.

A $G_2 = (\{S, A\}, \{a, *, +\}, \{S \to A + S \mid A,\ A \to A * A \mid a\}, S)$ nyelvtan egyértelmű.

Be lehet bizonyítani, hogy $L(G_1) = L(G_2)$.

19.3.3. Pumpáló lemma környezetfüggetlen nyelvekre

Környezetfüggetlen nyelvekre is létezik a reguláris nyelveknél megismert pumpáló lemmához hasonló lemma.



19.36. ábra. A pumpáló lemma bizonyításában szereplő fa felbontása.

19.29. tétel (pumpáló lemma). *Tetszőleges L környezetfüggetlen nyelvhez megadható egy olyan n természetes szám (amely csak az adott nyelvtől függ) úgy, hogy a nyelv minden n-nél hosszabb z szavát fel lehet írni $uvwxy$ alakban, és igazak a következők:*

(1) $|w| \geq 1$,
(2) $|vx| \geq 1$,
(3) $|vwx| \leq n$,
(4) uv^iwx^iy *is eleme L-nek, minden* $i \geq 0$ *értékre.*

Bizonyítás. Legyen $G = (N, T, P, S)$ egy olyan, átnevezéseket nem tartalmazó környezetfüggetlen nyelvtan, amely L-et generálja. Legyen $m = |N|$ a nemterminális jelek száma, és legyen ℓ a P-beli szabályok jobb oldalai hosszának maximuma, azaz $\ell = \max\{|\alpha| \mid \exists A \in N : (A \to \alpha) \in P\}$. Legyen $n = \ell^{m+1}$ és $z \in L(G)$ úgy, hogy $|z| > n$. Ekkor létezik egy olyan F levezetési fa, amelynek eredménye z. Legyen F magassága (a gyökértől a levelekig vezető utak hosszának a maximuma) h. Mivel F-ben minden belső csúcsnak legfeljebb ℓ leszármazottja van, ezért F-nek legfeljebb ℓ^h levele van, vagyis $|z| \leq \ell^h$. Másrészt, mivel $|z| > \ell^{m+1}$, kapjuk, hogy $h > m + 1$. Ebből következik, hogy az F levezetési fában van olyan út gyökértől levélig, amelyben $(m + 1)$-nél több csúcs van. Tekintsünk egy ilyen utat. Mivel G nemterminálisainak száma m és ezen az úton minden, levéltől különböző csúcs nemterminálissal van címkézve, a skatulya-elv szerint van olyan nemterminális, amelyik legalább kétszer fordul elő ezen az úton.

Tekintsük azt a nemterminálist, amelyik a levéltől a gyökérig haladva legelőször ismétlődik az úton és jelöljük A-val. Jelöljük F'-vel azt részfát, amelynek gyökere A ezen előfordulása. Hasonlóképpen, jelöljük F''-vel azt a részfát, amelynek gyökere az A következő (második) előfordulása az úton. Legyen F' eredménye w. Akkor F'' eredménye vwx, míg F eredménye $uvwxy$ alakban írható. Az F levezetési fa és z ezen felbontása a 19.36. ábrán látható. Megmutatjuk, hogy z felbontása kielégíti a lemmában megkövetelt (1)–(4) feltételeket.

Mivel P-ben nincsenek ε-szabályok (kivéve esetleg az $S \to \varepsilon$ szabályt), ezért $|w| \geq 1$. Továbbá, mivel a levezetési fa minden csúcsából és így F'' gyökeréből is legalább két él fut ki (ugyanis nincsenek átnevezések sem), ezért $|vx| \geq 1$. Mivel A az a nemterminális, amelyik a vizsgált úton a levéltől a gyökérig haladva legelőször megismétlődik, ezért F'' magassága legfeljebb $m+1$, amiből következik, hogy $|vwx| \leq \ell^{m+1} = n$.

Ha F-ből eltávolítjuk F'' csúcsait, csak a gyökeret hagyva meg, akkor az így kapott fa eredménye uAy, azaz $S \overset{*}{\underset{G}{\Longrightarrow}} uAy$.

Hasonlóképpen kapjuk F''-ből eltávolítva F'-t, hogy $A \overset{*}{\underset{G}{\Longrightarrow}} vAx$, és végül F' definíciója miatt, hogy $A \overset{*}{\underset{G}{\Longrightarrow}} w$. Tehát $S \overset{*}{\underset{G}{\Longrightarrow}} uAy$, $A \overset{*}{\underset{G}{\Longrightarrow}} vAx$ és $A \overset{*}{\underset{G}{\Longrightarrow}} w$. Innen $S \overset{*}{\underset{G}{\Longrightarrow}} uAy \overset{*}{\underset{G}{\Longrightarrow}} uwy$ és $S \overset{*}{\underset{G}{\Longrightarrow}} uAy \overset{*}{\underset{G}{\Longrightarrow}} uvAxy \overset{*}{\underset{G}{\Longrightarrow}} \ldots \overset{*}{\underset{G}{\Longrightarrow}} uv^iAx^iy \overset{*}{\underset{G}{\Longrightarrow}} uv^iwx^iy$ tetszőleges $i \geq 1$ értékre. Tehát tetszőleges $i \geq 0$-ra $S \overset{*}{\Longrightarrow} uv^iwx^iy$, vagyis tetszőleges $i \geq 0$-ra $uv^iwx^iy \in L(G)$. ■

Bemutatjuk a pumpáló lemma két következményét.

19.30. következmény. $\mathcal{L}_2 \subset \mathcal{L}_1$.

Bizonyítás. A következmény azt állítja, hogy létezik olyan környezetfüggő nyelv, amely nem környezetfüggetlen, tehát a tartalmazás szigorú. Ennek bizonyításához elég, ha találunk egy olyan környezetfüggő nyelvet, amelyre az előbbi pumpáló lemma nem teljesül. Legyen ez $L = \{a^m b^m c^m \mid m \geq 1\}$.

Hogy ez a nyelv környezetfüggő, azt könnyen lehet igazolni, megfelelő nyelvtan megadásával. A 19.2. példában megadott két nyelvtan mindegyike kiterjesztett környezetfüggő nyelvtan. De tudjuk, hogy tetszőleges i típusú kiterjesztett nyelvtanhoz mindig hozzárendelhető ugyanolyan típusú és vele ekvivalens nyelvtan.

Legyen n a pumpáló lemmában az L-hez tartozó természetes szám, és tekintsük a $z = a^n b^n c^n$ szót. Mivel $|z| = 3n > n$, z felbontható $z = uvwxy$ alakban úgy, hogy teljesülnek a pumpáló lemmában szereplő (1)–(4) feltételek. Megmutatjuk, hogy ez ellentmondáshoz vezet.

Először megmutatjuk, hogy a v és x szavak mindegyike legfeljebb egyféle betűt tartalmaz. Valóban ha v és x valamelyike egynél többféle betűt tartalmaz, akkor az $uvvwxxy$ szóban a betűk sorrendje nem az a, b, c sorrend, tehát $uvvwxxy \notin L(G)$, ami ellentmond a pumpáló lemmában szereplő (4) feltételnek.

Ha viszont v és x mindegyike legfeljebb egyféle betűt tartalmaz, akkor az uwy szóban valamelyik betű többször fordul elő, mint a másik kettő, tehát $uwy \notin L(G)$. Ez is ellentmond a pumpáló lemmában szereplő (4) feltételnek, tehát L nem környezetfüggetlen. ■

19.31. következmény. *A környezetfüggetlen nyelvek osztálya nem zárt a metszetre.*

Bizonyítás. Megadunk két olyan környezetfüggetlen nyelvet, amelyeknek metszete nem környezetfüggetlen. Legyen $N = \{S, A, B\}$, $T = \{a, b, c\}$ és

$G_1 = (N, T, P_1, S)$ ahol P_1 :

$S \to AB$,
$A \to aAb \mid ab$,
$B \to cB \mid c$,

és $G_2 = (N, T, P_2, S)$, ahol P_2 :
$S \to AB$,
$A \to Aa \mid a$,
$B \to bBc \mid bc$.

Tehát az $L(G_1) = \{a^n b^n c^m \mid n \geq 1, m \geq 1\}$ és az $L(G_2) = \{a^n b^m c^m \mid n \geq 1, m \geq 1\}$ nyelvek környezetfüggetlenek. Ugyanakkor

$$L(G_1) \cap L(G_2) = \{a^n b^n c^n \mid n \geq 1\}$$

nem környezetfüggetlen, mint ahogy azt a 19.30 következmény bizonyításában láttuk. ■

19.3.4. Környezetfüggetlen nyelvtanok normálalakjai

Tetszőleges nyelvtanok esetében a normálalakot úgy értelmeztük (lásd 900. oldal), hogy a szabályok bal oldalán csak nemterminálisok szerepelnek. Környezetfüggetlen nyelvtanok esetében a normálalak a szabályok jobb oldalára adott bizonyos megkötéseket jelent. A következőkben a környezetfüggetlen nyelvtanoknak két normálalakját vizsgáljuk meg, a Chomsky-, illetve a Greibach-féle normálalakokat.

Chomsky-féle normálalak

19.32. definíció. *Egy $G = (N, T, P, S)$ környezetfüggetlen nyelvtan Chomsky-normálalakú, ha minden szabálya $A \to a$ vagy $A \to BC$ alakú, ahol $A, B, C \in N$, $a \in T$.*

19.31. példa. A $G = (\{S, A, B, C\}, \{a, b\}, \{S \to AB,\ S \to CB,\ C \to AS,\ A \to a,\ B \to b\}, S)$ nyelvtan Chomsky-normálalakú és $L(G) = \{a^n b^n \mid n \geq 1\}$.

Minden ε-mentes környezetfüggetlen nyelvtanhoz megadható egy vele ekvivalens Chomsky-normálalakú nyelvtan. Megadunk egy algoritmust, amely a $G = (N, T, P, S)$ ε-mentes környezetfüggetlen nyelvtant átalakítja a $G' = (N', T, P', S)$ Chomsky-normálalakúvá.

CHOMSKY-ALAK(G, G')

1 $N' \leftarrow N$
2 küszöböljük ki a szabályokban az átnevezéseket, és legyen P' az új szabályhalmaz (lásd ÁTNEVEZÉS-KIZÁRÁS algoritmus, 899. oldal)
3 a P' minden olyan szabályában, amelynek jobb oldala legalább két szimbólumból áll, minden a terminális jelet helyettesítsünk egy új A nemterminálissal amelyet vegyünk fel N'-be, és vegyük fel a szabályok közé az $A \to a$ új szabályt

4 minden $B \to A_1A_2 \dots A_k$ alakú szabályt, ahol $k \geq 3$ és $A_1, A_2, \dots, A_k \in N$, helyettesítsünk a következőkkel:

$$\begin{aligned} B &\to A_1C_1, \\ C_1 &\to A_2C_2, \\ &\dots \\ C_{k-3} &\to A_{k-2}C_{k-2}, \\ C_{k-2} &\to A_{k-1}A_k, \end{aligned}$$

ahol $C_1, C_2, \dots, C_{k-2}$ új nemterminális szimbólumok, amelyeket felveszünk N'-be.

19.32. példa. Legyen $G = (\{S, D\}, \{a, b, c\}, \{S \to aSc,\ S \to D,\ D \to bD,\ D \to b\}, S)$. Könnyű belátni, hogy $L(G) = \{a^n b^m c^n \mid n \geq 0, m \geq 1\}$. A Chomsky-féle normálalakú nyelvtanná való alakítás lépései a következők:

1. lépés: $N' = \{S, D\}$

2. lépés: Az $S \to D$ átnevezés kiküszöbölése után, a szabályok a következők:

$$\begin{aligned} S &\to aSc \mid bD \mid b, \\ D &\to bD \mid b. \end{aligned}$$

3. lépés: Mivel a szabályokban három terminális szerepel, három új nemterminálist vezetünk be (legyenek ezek A, B, C). Ekkor a szabályok:

$$\begin{aligned} S &\to ASC \mid BD \mid b, \\ D &\to BD \mid b, \\ A &\to a, \\ B &\to b, \\ C &\to c. \end{aligned}$$

4. lépés: Mivel egyetlen szabály kivételével, amelynek jobb oldala három szimbólumból áll, minden más szabály jobb oldala legfeljebb kettő hosszúságú, egyetlen új nemterminálist kell bevezetnünk, legyen ez E. Ezért $N' = \{S, A, B, C, D, E\}$, a P' szabályai pedig:

$$\begin{aligned} S &\to AE \mid BD \mid b, \\ D &\to BD \mid b, \\ A &\to a, \\ B &\to b, \\ C &\to c, \\ E &\to SC. \end{aligned}$$

Ezek a szabályok már mind megfelelő alakúak.

Greibach-féle normálalak

19.33. definíció. *Egy $G = (N, T, P, S)$ környezetfüggetlen nyelvtan* ***Greibach-normálalakú****, ha minden szabálya $A \to aw$ alakú, ahol $A \in N,\ a \in T,\ w \in N^*$.*

19.33. példa. A $G = (\{S, B\}, \{a, b\}, \{S \to aB,\ S \to aSB,\ B \to b\}, S)$ nyelvtan Greibach-normálalakú és $L(G) = \{a^n b^n \mid n \geq 1\}$.

Minden ε-mentes környezetfüggetlen nyelvtanhoz megadható egy vele ekvivalens Greibach-normálalakú nyelvtan. Megadunk egy algoritmust, amely a Chomsky-normálalakban levő $G = (N, T, P, S)$ környezetfüggetlen nyelvtant átalakítja a $G' =$

(N', T, P', S) Greibach-normálalakúvá.

Először rögzítjük a nemterminális jelek egy $A_1, A_2, \ldots, A_n$ sorrendjét úgy, hogy A_1 legyen a kezdőszimbólum. Az algoritmusban a következő jelöléseket használjuk: $x \in N'^{+}$, $\alpha \in TN'^{*} \cup N'^{+}$.

GREIBACH-ALAK(G,G')

```
1  N' ← N
2  P' ← P
3  for i ← 2 to n                                   ▷A_i → A_j x,  j < i esete.
4      do for j ← 1 to i − 1
5             do minden A_i → A_j x és minden A_j → α alakú szabályra
                  (ahol α nem kezdődik A_j-vel)
                     vegyük fel P'-be az A_i → αx szabályt,
                  töröljük P'-ből az A_i → A_j x szabályokat
6      if létezik A_i → A_i x alakú szabály          ▷A_i → A_i x esete.
7        then vegyük fel N'-be az új B_i nemterminálist,
             minden A_i → A_i x alakú szabályra vegyük fel P'-be a B_i → xB_i és B_i → x
             szabályokat,
             töröljük P'-ből az A_i → A_i x szabályt,
             minden A_i → α alakú szabályra (ahol α nem kezdődik A_i-vel)
             vegyük fel P'-be az A_i → αB_i szabályt
8  for i ← n − 1 downto 1                           ▷A_i → A_j x,  j > i esete.
9      do for j ← i + 1 to n
10        do minden A_i → A_j x és minden A_j → α alakú szabályra
                vegyük fel P'-be az A_i → αx szabályt és
                töröljük P'-ből az A_i → A_j x szabályokat,
11 for i ← 1 to n                                   ▷B_i → A_j x esete.
12     do for j ← 1 to n
13            do minden B_i → A_j x és minden A_j → α alakú szabályra
                  vegyük fel P'-be a B_i → αx szabályt és
                  töröljük P'-ből a B_i → A_j x szabályokat
```

Az algoritmus az első lépésben az $A_i \to A_j x,\ j < i$ alakú szabályokat átalakítja úgy, hogy azok $A_i \to A_j x,\ j \geq i$ vagy $A_i \to \alpha$ alakúak legyenek, ahol ez utóbbi már Greibach-normálalakú. A második lépésben, új nemterminális bevezetésével, kiküszöböli az $A_i \to A_i x$ alakú szabályokat, majd helyettesítésekkel eléri, hogy az $A_i \to A_j x,\ j > i$ és $B_i \to A_j x$ alakú szabályok is Greibach-normálalakúak legyenek.

19.34. példa. Alakítsuk át a következő Chomsky-normálalakú szabályokat Greibach-normálalakúvá:

$A_1 \to A_2A_3 \mid A_2A_4$
$A_2 \to A_2A_3 \mid a$
$A_3 \to A_2A_4 \mid b$
$A_4 \to c$

Az algoritmus lépései:

3–5: Az $A_3 \to A_2A_4$ szabályt kell átalakítani. Erre csak az $A_2 \to a$ szabály alkalmas. Ezért felvesszük a szabályok közé az $A_3 \to aA_4$ szabályt és töröljük az $A_3 \to A_2A_4$ szabályt.

Tehát a szabályok:

$A_1 \to A_2A_3 \mid A_2A_4$
$A_2 \to A_2A_3 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$

6-7: Az $A_2 \to A_2A_3$ kiküszöbölése a következő szabályokkal történik:

$B_2 \to A_3B_2$
$B_2 \to A_3$
$A_2 \to aB_2$

Tehát, a 6–7. lépések után, a szabályok:

$A_1 \to A_2A_3 \mid A_2A_4$
$A_2 \to aB_2 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$
$B_2 \to A_3B_2 \mid A_3$

8–10: Az A_1 baloldalú szabályoknál végzünk helyettesítéseket. Az eredmény:

$A_1 \to aA_3 \mid aB_2A_3 \mid aA_4 \mid aB_2A_4$

11–13: Hasonlóképpen járunk el a B_2 baloldalú szabályokkal:

$B_2 \to aA_4B_2 \mid aA_3A_4B_2 \mid aA_4 \mid aA_3A_4$

Miután kitöröltük a 8–13. lépésekben a helyettesített szabályokat, a következőket kapjuk, amelyek már mind Greibach-alakú szabályok:

$A_1 \to aA_3 \mid aB_2A_3 \mid aA_4 \mid aB_2A_4$
$A_2 \to aB_2 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$
$B_2 \to aA_4B_2 \mid aA_3A_4B_2 \mid aA_4 \mid aA_3A_4$

19.35. példa. Nézzünk meg egy másik példát. Megadunk egy nyelvtant a következő nyelv generálására.

$$L = \{a^n b^k c^{n+k} \mid n \geq 0, k \geq 0, n + k > 0.\}$$

Bebizonyítható, hogy a következő nyelvtan generálja L-et.

$$G = \{\{S, R\}, \{a, b, c\}, \{S \to aSc,\ S \to ac,\ S \to R, R \to bRc,\ R \to bc\}, S\}$$

Először kiküszöböljük az átnevezéseket (itt most csupán egy van), azután megadunk egy vele ekvivalens Chomsky-alakú nyelvtant, majd egy ezzel ekvivalens Greibach-alakút. Az $S \to R$ átnevezés kiküszöbölése után a következő szabályokot kapjuk:

$S \to aSc \mid ac \mid bRc \mid bc$
$R \to bRc \mid bc.$

Bevezetjük az $A \to a, B \to b, C \to c$ szabályokat, majd a terminálisokat helyettesítjük minden szabály jobb oldalán a megfelelő változóval:

$S \to ASC \mid AC \mid BRC \mid BC,$
$R \to BRC \mid BC,$
$A \to a,\ B \to b,\ C \to c.$

Két új változó (D, E) bevezetése után:

$S \to AD \mid AC \mid BE \mid BC,$
$D \to SC,$
$E \to RC,$
$R \to BE \mid BC,$
$A \to a,\ B \to b,\ C \to c.$

Ez már Chomsky-féle normálalak. Induljunk ki ebből a nyelvtanból, miután átírjuk a változókat A_i alakúra, hogy könnyebben alkalmazhassuk az algoritmust. Tehát, a következő átnevezés után

S helyett A_1, A helyett A_2, B helyett A_3, C helyett A_4, D helyett A_5,
E helyett A_6, R helyett A_7,

átnevezés után nyelvtanunk a következő szabályokat tartalmazza:

$A_1 \to A_2A_5 \mid A_2A_4 \mid A_3A_6 \mid A_3A_4$,
$A_2 \to a,\ A_3 \to b,\ A_4 \to c$,
$A_5 \to A_1A_4$,
$A_6 \to A_7A_4$,
$A_7 \to A_3A_6 \mid A_3A_4$.

Az algoritmus 3–5. lépéseinek alkalmazásakor a következő új szabályok jelennek meg:

$A_5 \to A_2A_5A_4 \mid A_2A_4A_4 \mid A_3A_6A_4 \mid A_3A_4A_4$ majd
$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$
$A_7 \to A_3A_6 \mid A_3A_4$, majd
$A_7 \to bA_6 \mid bA_4$.

Tehát:

$A_1 \to A_2A_5 \mid A_2A_4 \mid A_3A_6 \mid A_3A_4$,
$A_2 \to a,\ A_3 \to b,\ A_4 \to c$,
$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$
$A_6 \to A_7A_4$,
$A_7 \to bA_6 \mid bA_4$.

A 6–7. lépéseket átugorjuk, hisz nincs balrekurzív szabály. A 8–10. lépésekben a megfelelő helyettesítések után:

$A_1 \to aA_5 \mid aA_4 \mid bA_6 \mid bA_4$,
$A_2 \to a$,
$A_3 \to b$,
$A_4 \to c$,
$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$
$A_6 \to bA_6A_4 \mid bA_4A_4$,
$A_7 \to bA_6 \mid bA_4$.

Gyakorlatok

19.3-1. Adjunk meg egy-egy veremautomatát a következő nyelvek felismerésére:

$L_1 = \{a^n cb^n \mid n \geq 0\}$,
$L_2 = \{a^n b^{2n} \mid n \geq 1\}$,
$L_3 = \{a^{2n} b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}$,

19.3-2. Adjunk meg egy olyan környezetfüggetlen nyelvtant, amely az $L = \{a^n b^n c^m \mid n \geq 0, m \geq 0\}$ nyelvet generálja, majd írjuk át Chomsky-, illetve Greibach-normálalakúvá. Adjunk meg egy veremautomatát, amely felismeri az L nyelvet.

19.3-3. Milyen nyelvet generálnak a következő környezetfüggetlen nyelvtanok?

$G_1 = (\{S\}, \{a, b\}, \{S \to SSa,\ \to b\}, S)$, $G_2 = (\{S\}, \{a, b\}, \{S \to SaS,\ \to b\}, S)$

19.3-4. Adjunk meg egy környezetfüggetlen nyelvtant, amely olyan szavakat generál, amelyben egyenlő számban vannak az a és b betűk.

19.3-5. Bizonyítsuk be a pumpáló lemma alkalmazásával, hogy az a nyelv, amelynek minden szava ugyanannyi a, b és c betűt tartalmaz, nem környezetfüggetlen.

19.3-6. Adott a következő nyelvtan: $G = (V, T, P, S)$, ahol
$V = \{S\}$,
$T = \{\textit{if, then, else, a, c}\}$,
$P = \{S \to \textit{if a then } S,\quad S \to \textit{if a then } S \textit{ else } S,\quad S \to c\}$,
Mutassuk meg, hogy az *if a then if a then c else c* szónak létezik két különböző legbaloldalibb levezetése.

19.3-7. Bizonyítsuk be, hogy ha L környezetfüggetlen, akkor $L^{-1} = \{u^{-1} \mid u \in L\}$ is környezetfüggetlen.

Feladatok

19-1. Lineáris nyelvtanok

Az olyan $G = (N, T, P, S)$ nyelvtant, amelynek minden szabálya $A \to u_1 B u_2$ vagy $A \to u$ alakú, ahol $A, B \in N$, $u, u_1, u_2 \in T^*$, ***lineáris nyelvtannak*** nevezzük. Amennyiben egy lineáris nyelvtanban minden szabály $A \to Bu$ vagy $A \to v$ alakú, akkor ballineáris nyelvtanról beszélünk. Bizonyítsuk be, hogy minden ballineáris nyelvtan által generált nyelv reguláris.

19-2. Operátornyelvtanok

Egy ε-mentes környezetfüggetlen nyelvtant ***operátornyelvtannak*** nevezünk, ha a szabályai jobb oldalán nincs két nemterminális szimbólum egymás mellett. Igazoljuk, hogy minden ε-mentes környezetfüggetlen nyelvtanhoz megkonstruálható egy vele ekvivalens operátornyelvtan.

19-3. Környezetfüggetlen nyelvek komplementuma

Bizonyítsuk be, hogy a környezetfüggetlen nyelvek osztálya nem zárt a komplementumra.

Megjegyzések a fejezethez

A véges automata definíciójában eltértünk a hagyományos értelmezéstől, az átmenetfüggvény helyett az átmenetgráfot használtuk. Ezt a szemléletmódot követtük a veremautomata esetében is, amely hasznosnak bizonyult sok esetben, nagyban egyszerűsítvén a bizonyításokat.

Az automatákról és a formális nyelvekről sok klasszikus könyv létezik. Ezek közül megemlítjük a következőket: Aho és Ullman két könyve [11, 12] 1972-ből és 1973-ból, Gécseg Ferenc és Peák István [129] angol nyelvű könyve 1972-ből, Salomaa két könyve [304, 305] 1969-ból és 1973-ból, Hopcroft és Ullman [163] könyve 1979-ből, Harrison [157] könyve 1978-ból, Manna [241] könyve, amely 1981-ben magyar fordításban is megjelent. Megemlítjük még Sipser [315] 1997-es könyvét, valamint Rozenberg és Salomaa [300] monográfiáját. Lothaire (francia szerzők közös neve) [230] szókombinatorikai könyvében egyéb típusú automatákról is olvashatunk. Giammarresi és Montalbano cikke [135] az általánosított véges automatákkal foglalkozik. A témakör friss angol nyelvű monográfiája Hopcroft, Motwani és Ullman [162] műve. Német nyelven Asteroth és Baier [24] tankönyvét ajánljuk. A Greibach-féle normálalakra való hozás algoritmusának tömör leírása innen való.

További, a témával foglalkozó angol nyelvű könyvek: [54, 61, 96, 195, 211, 224, 228, 248, 252, 313, 314, 325, 326].

Magyar nyelven is több jegyzet és könyv tárgyalja az automaták és formális nyelvek témáját: Bach Iván [30], a Demetrovics–Denev–Pavlov szerzőhármas [86], Fülöp Zoltán [116], Peák István [269, 270, 271], Révész György [301] könyve, valamint Dömösi–Fazekas–Horváth–Mecsei [90] és Hunyadvári–Manhertz [167] digitális kézirata.

A fordítóprogramokról szóló fejezet végén további, a témához kapcsolódó könyvekre is találunk utalást.

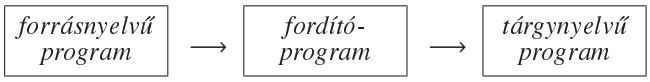
20. Fordítóprogramok elemzési algoritmusai

A programozó egy feladat számítógéppel történő megoldását valamilyen programozási nyelven írja le. Ez a nyelv azonban nagyon különbözik a számítógép nyelvétől, a gépi kódtól, ezért elő kell állítani a programozó által megadott programnak a számítógép által végrehajtható formáját. Szükség van tehát egy olyan hardver vagy szoftver eszközre, amelyik a magasszintű programnyelven, a ***forrásnyelven*** megírt programot „lefordítja” egy alacsonyabb szintű nyelven írt ***tárgynyelvű*** programra, többnyire a számítógép gépi kódú programjára.

Egy magasszintű programnyelven írt feladat számítógépes végrehajtásának alapvetően két módszere van. Az egyik az, amikor egy ***interpretert*** használunk, azaz amikor a lefordított kódot nem tároljuk el, hanem azonnal végrehajtjuk. Ez formálisan úgy tekinthető, hogy az interpreter egy olyan számítógép, amelynek gépi kódja ez a magasszintű nyelv. Az interpreterrel tehát egy kétszintű gépet hozunk létre, amelyiknek alsó szintje a tényleges fizikai számítógép, és erre épül rá a magasszintű nyelvet értelmező gép. Ezt a magasszintű gépet programmal valósíthatjuk meg, de egyes programnyelvekhez speciális hardver értelmező gépeket is készítenek.

A másik módszer esetén egy ***compilernek*** nevezett programot használunk, ami lényegében csak abban különbözik az interpretertől, hogy a fordítás eredményét nem hajtja végre, hanem egy közbülső állományban, a tárgyprogramban tárolja. Ezt a tárgyprogramot egy későbbi időpontban futtathatjuk le, és majd csak ekkor kapjuk meg a program eredményét. Látható, hogy itt, ellentétben az interpretálással, a fordítási időpont és a futtatási időpont egymástól jól elkülöníthető.

Mivel mind az interpretáláskor, mind a compiler használatakor elő kell állítani a tárgyprogramot, a fordítás szempontjából a két megoldási módszer azonos, és így a továbbiakban egyszerűen csak „fordításról” beszélünk, és a fordítást végrehajtó programot ***fordítóprogramnak*** nevezzük (20.1. ábra).



20.1. ábra. A fordítóprogram.

Feladatunk a fordítási algoritmusok vizsgálata. Ebben a fejezetben a magasszintű imperatív nyelvek fordítóprogramjait vizsgáljuk, és nem elemezzük például a logikai vagy a funkcionális programozási nyelvek fordítási módszereit. Először a fordítóprogramok felépítését adjuk meg, majd a lexikális elemzéssel foglalkozunk. A szintaktikus elemzés témakörében a két legsikeresebb algoritmust, az *LL*(1) és az *LALR*(1) elemzési módszert ismertetjük. A szemantikus elemzés korszerű módszerei *O-ATG* nyelvtanokat használnak, és a kódgenerálás feladata is ilyen típusú nyelvtannal írható le. Itt most nem foglalkozunk ezekkel, és olyan fontos és érdekes problémákkal sem, mint a szimbólumtábla szerkezete és használata, a fordítóprogramok hibajavító módszerei, vagy például a kódoptimalizálás. Ezekre a témakörökre az irodalomjegyzékben megadott könyvek adnak új, modern és nagyon hatékony módszereket.

20.1. A fordítóprogram szerkezete

A fordítóprogram a forrásnyelvű programot tárgynyelvű programmá alakítja át, és ezenkívül egy listát is készít, amely a programozó számára visszaigazolja a lefordított forrásnyelvű szöveget. Ez a lista tartalmazza a felfedezett hibákat is. A forrásnyelvű programot röviden forrásprogramnak, a tárgynyelvű programot tárgyprogramnak is nevezhetjük.

A *program (bemenet)(kimenet)* jelölést használva, a fordítóprogram a

fordítóprogram (forrásnyelvű program)(tárgynyelvű program, lista)

sorral írható le. A továbbiakban a fordítóprogramok felépítését vizsgáljuk, és a fenti jelölésmódot alkalmazva, megadjuk az egyes programelemek által végrehajtandó feladatokat.

Az első programelem a forrásnyelvű programot egy könnyen kezelhető karaktersorozattá alakítja át. Ez a program a ***bemenetkezelő:***

bemenetkezelő (forrásnyelvű program)(karaktersorozat).

A bemenetkezelő az operációs rendszertől függő formátumú fájlt az operációs rendszer rendszerhívásainak felhasználásával beolvassa, és közben már kihagyja a további feldolgozás szempontjából közömbös sorvéget jelző karaktereket. Ez a módosított, „ömlesztett” karaktersorozat lesz a fordítás további lépéseinek bemenő adata.

A fordítóprogram által készített listának azonban nem ezt a karaktersorozatot kell tartalmaznia, hanem a programozó által írt alakban az eredeti forrásnyelvű programot. Így meg kell adnunk egy ***listakezelő*** programot,

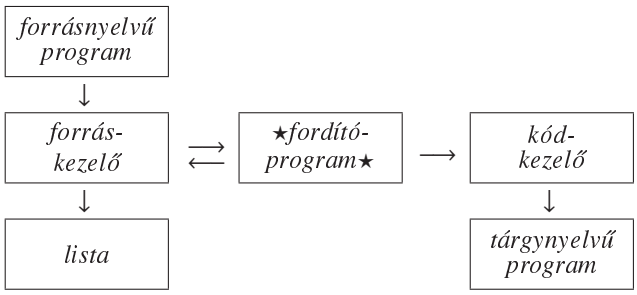
listakezelő (forrásnyelvű program, hibák)(lista),

ami a listát az operációs rendszernek megfelelő fájlformátumban, általában valamelyik háttértárolón helyezi el.

A bemenetkezelő és a listakezelő programokat, mivel mindkettőnek bemenete a forrásnyelvű program, célszerű összevonni egy programmá, ezt a programot ***forráskezelőnek*** nevezzük:

forráskezelő (forrásnyelvű program, hibák)(karaktersorozat, lista).

A fordítóprogram által készített tárgynyelvű programot a háttértárolón, egy fájlban, valamilyen relokálható bináris formátumban kell elhelyezni, a formátum természetesen ismét az operációs rendszertől függ. Ezt a műveletet a ***kódkezelő*** végzi el:



20.2. ábra. A fordítóprogram felépítése.

kódkezelő (tárgykód)(tárgynyelvű program).

Így tehát a fordítóprogram struktúrája a következő lesz (20.2. ábra):

forráskezelő (forrásnyelvű program, hibák) (karaktersorozat, lista),
⋆fordítóprogram⋆ (karaktersorozat)(tárgykód, hibák),
kódkezelő (tárgykód)(tárgynyelvű program).

Ez a felbontás nem szekvenciát jelöl, a három programelem nem szekvenciálisan hajtódik végre. A fenti felbontással a fordítóprogramot három egymástól jól elkülöníthető működési egységre bontottuk fel. Az egyes működési egységek kapcsolatát az egységek bemenete és kimenete jelzi.

A két kezelővel, elsősorban a számítógéptől, a perifériáktól és az operációs rendszerektől való függőségük miatt, a továbbiakban nem foglalkozunk, bár a fordítóprogramok külső jellemzőit, kezelhetőségét, a felhasználóval való kapcsolatot alapvetően ezek határozzák meg.

A *⋆fordítóprogram⋆* programelem most már ténylegesen csak a fordítással foglalkozik. Két nagy feladatot kell megoldania: analizálnia kell a bemenetként kapott karaktersorozatot, és ebből szintetizálnia kell a tárgykódot.

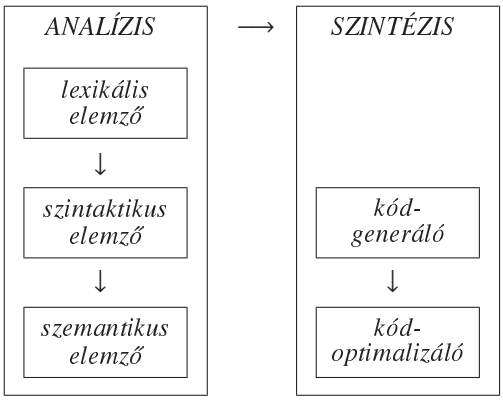
Az *analízis* első feladata az, hogy a karaktersorozatban meghatározza az egyes összefüggő sorozatokat, a szimbolikus egységeket. Ilyenek például a konstansok, változók, kulcsszavak, operátorok. Ezt a programot ***lexikális elemzőnek*** nevezzük. A lexikális elemző a karaktersorozatból egy ***szimbólumsorozatot*** készít, és közben ***lexikális hibákat*** fedezhet fel:

lexikális elemző (karaktersorozat)(szimbólumsorozat, lexikális hibák).

A lexikális elemző által készített szimbólumsorozat a bemenete a ***szintaktikus elemzőnek***. A szintaktikus elemző feladata a program struktúrájának vizsgálata. Ez a folyamat hasonlít ahhoz, amikor nyelvtan órán egy mondat alanyát, állítmányát, tárgyát, határozóit és jelzőit határozzuk meg. Az elemzés közben felfedezett hibák lesznek a ***szintaktikus hibák***. A szintaktikus elemző működésének az eredménye az elemzett program szintaxisfája, vagy valamilyen ezzel ekvivalens struktúra:

szintaktikus elemző (szimbólumsorozat)(szintaktikusan elemzett program, szintaktikus hibák).

Az analízis harmadik programja a ***szemantikus elemző***, amelynek feladata a statikus szemantika tulajdonságainak vizsgálata. A szemantikus elemző feladata például az, hogy az



20.3. ábra. Az analízis és szintézis programjai.

a+b kifejezés elemzésekor megvizsgálja, az a és b változók deklarálva vannak-e, azonos típusúak-e, és hogy van-e értékük. Az itt felfedezett hibákat ***szemantikus hibáknak*** nevezzük.

szemantikus elemző (szintaktikusan elemzett program)(analizált program, szemantikus hibák).

A szemantikus elemző kimenete lesz a ***szintetizálást*** végző programok bemenő adata. A szintézis első lépése a ***kódgenerálás,*** amelyet a kódgeneráló program végez el:

kódgeneráló (analizált program)(tárgykód).

A tárgykód gépfüggő, operációs rendszertől függő, gyakran assembly nyelvű vagy gépi kódú program. A szintetizálás következő lépése a ***kódoptimalizálás***:

kódoptimalizáló (tárgykód)(tárgykód).

A kódoptimalizáló a létrehozott tárgykódot úgy alakítja át, hogy a tárgykód adott szempontok, általában a tárgykód mérete és futási ideje szerint optimális legyen.

A fentiek alapján egy fordítóprogram a következő részekre bontható (az analízist és a szintézist végző, korábban ⋆fordítóprogram⋆-gal jelölt program szerkezete a 20.3. ábrán látható):

forráskezelő (forrásnyelvű program, hibák)(karaktersorozat, lista),
lexikális elemző (karaktersorozat)(szimbólumsorozat, lexikális hibák),
szintaktikus elemző (szimbólumsorozat)(szintaktikusan elemzett program, szintaktikus hibák),
szemantikus elemző (szintaktikusan elemzett program)(analizált program, szemantikus hibák),
kódgeneráló (analizált program)(tárgykód),
kódoptimalizáló (tárgykód)(tárgykód),
kódkezelő(tárgykód)(tárgyprogram).

Ennek megfelelően a fordítóprogramok analízist és szintézist végző részének algoritmusa a következőképpen írható le:

★FORDÍTÓPROGRAM★

1 határozzuk meg a forrásnyelvű program szövegében a lexikális szimbólumokat
2 ellenőrizzük a szimbólumsorozat szintaktikus helyességét
3 ellenőrizzük a szimbólumsorozat szemantikus helyességét
4 határozzuk meg a tárgyprogramba kerülő kódot
5 optimalizáljuk a tárgyprogram kódját

Ezekkel a programokkal foglalkozunk a következő alfejezetekben.

Gyakorlatok

20.1-1. A megadott jelölésrendszert használva, adjuk meg az interpreterek szerkezetét.
20.1-2. Válasszunk egy programnyelvet, és ezt a nyelvet használva mutassunk néhány olyan programrészletet, amiben lexikális, szintaktikus vagy szemantikus hiba van.
20.1-3. Adjunk meg olyan szempontokat, amelyek szerint a *kódoptimalizáló* optimalizálhatja a tárgykódot.

20.2. Lexikális elemzés

A forrásnyelvű programból a *forráskezelő* egy karaktersorozatot készít. A lexikális elemző alapvető feladata az, hogy ebben a karaktersorozatban felismerje a ***szimbolikus egységeket,*** ezeket röviden ***szimbólumoknak*** nevezzük.

Sajnos különböző programnyelvekben az azonos fogalmakat jelentő szimbolikus egységek gyakran lényegesen különböznek, és különböző programnyelvekben két különböző szimbólumhoz azonos karaktersorozatok is tartozhatnak.

Van olyan programnyelv, amelyben az `1.` és `.10` karaktersorozatok valós számokat jelölnek. Ha ezt a két számot egymás mellé írjuk, akkor az `1..10` karaktersorozatot kapjuk. Azt, hogy a két szám között legalább egy műveleti jelnek is kell lennie, majd a szintaktikus elemző fedezi fel. Vannak azonban olyan programnyelvek, amelyek szerint az `1..10` karaktersorozat nem két számra bontható, hanem egy intervallum típusú változó alsó és felső határát definiáló három szimbólumra.

Az elemző nem csak a szimbólumok szövegét határozza meg, hanem a szövegből kikövetkeztethető jellemző adatokat is. Ilyen például a szimbólum típusa, vagy például a szimbólum értéke.

A lexikális elemző a szimbólumokhoz kódokat rendel, az azonos fajta szimbólumokhoz azonos kódokat. Például azonos kódot kapnak az egész számok, és ugyanazt az egyedi kódot kapják a változók. A lexikális elemző a karaktersorozatot egy ***szimbólumsorozatra,*** pontosabban szimbólumkódok sorozatára alakítja át, és a szimbólumra vonatkozó adatokat rendszerint a szimbólum kódja után helyezi el.

A lexikális elemzés után a program szövege már nem „olvasható”. Megjegyezzük, hogy a fordítás szempontjából ettől kezdve már teljesen mindegy, hogy az adott szimbólum milyen karaktersorozatból származott, azaz hogy egy *if* szimbólum az angol *if*, a magyar *ha* vagy a német *wenn* szó karaktereiből készült. Ez azt jelenti, hogy egy meglévő és például

angol kulcsszavakat használó programnyelvhez könnyen készíthetünk egy más nyelv szavait használó programnyelvet, az új nyelv fordítóprogramjában csak a lexikális elemzőt kell az új nyelvre átalakítani, a fordítóprogram többi része változatlan maradhat.

20.2.1. Az elemzés automatája

A szimbolikus egységek precíz definíciója *reguláris nyelvtannal*, *reguláris kifejezésekkel* vagy *determinisztikus véges automatával* adható meg. A reguláris nyelvtanok, reguláris kifejezések, determinisztikus véges automaták definícióját és a rájuk vonatkozó tételeket a kötet első fejezetében már ismertettük.

A lexikális elemző tulajdonképpen lehetne a szintaktikus elemző része is, de a lexikális elemző és a szintaktikus elemző megkülönböztetésének alapvető oka éppen az, hogy a reguláris nyelvtannal megadható lexikális elemző programja sokkal egyszerűbb lesz annál, mintha erre a feladatra is a szintaktikus elemzés környezetfüggetlen nyelvtanát használnánk.

A lexikális elemzők létrehozásának egyik módszere a következő:

1. a szimbolikus egységek leírását a reguláris kifejezések nyelvén adjuk meg, és megkonstruáljuk az ekvivalens véges determinisztikus automatát,
2. elkészítjük a determinisztikus véges automata implementációját.

Megjegyezzük, hogy a szimbólumok leírására azért használunk inkább reguláris kifejezéseket, mert ezekkel a szimbólumok kényelmesebben és olvashatóbban adhatók meg, mint a reguláris nyelvtanokkal. Vannak olyan programok (például a UNIX `lex` programja), amelyek reguláris kifejezésekből generálják a teljes lexikális elemző programot, és vannak olyan programok is, amelyek kimenetként még az elemző automatáját is megadják.

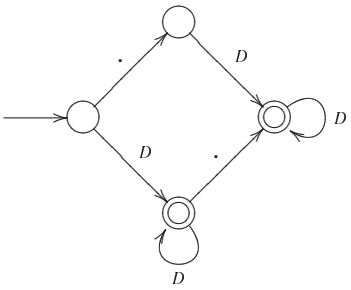
A determinisztikus véges automata könnyen implementálható a többirányú elágazást végző **case** utasítás felhasználásával. Az elágazások értékei az állapot-átmenetek karakterei, az értékekhez tartozó utasítások pedig az automata azon állapotait reprezentálják, amelybe a szimbólumhoz tartozó karakter hatására az automata kerül.

A lexikális elemző működésének alapelve az, hogy egy szimbólumot mindig a lehető *leghosszabb karaktersorozatból* kell felépíteni, például az `ABC` szöveg nem három egybetűs, hanem egy hárombetűs szimbólum lesz. Ez azt jelenti, hogy a **case** elágazás alternatív utasításai addig dolgozzák fel a karaktereket, amíg azok az éppen építés alatt álló szimbólum részeként értelmezhetők.

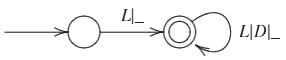
Az automata végállapotaihoz szimbólumfeldolgozó funkciók is tartozhatnak, ilyen funkció például az, ha egy felismert konstans szimbólum értékét egy belső ábrázolási formára kell átalakítani, vagy ha egy azonosító szimbólumot a szimbólumtáblába kell felírni.

A lexikális elemző bemenet karaktersorozatában benne van az összes szóköz és tabulátor jel, hiszen a forráskezelőről csak annyit tételeztünk fel, hogy a kocsivissza és soremelés karaktereket hagyja el. A legtöbb programozási nyelv tetszőlegesen sok szóköz és tabulátor karaktert megenged az egyes szimbólumok között. Ezeknek a karaktereknek a fordítás szempontjából a szimbólumok felismerése után már nincs szerepük, ezért ezeket ***fehér szóközöknek*** nevezzük. A fehér szóközök kiszűrése szintén a lexikális elemző feladata, a fehér szóközök leírására a következő reguláris kifejezés adható meg:

$$(space \mid tab)^* \ ,$$



20.4. ábra. A pozitív egész és valós szám.



20.5. ábra. Az azonosító szimbólum.

ahol a *space* a szóköz, a *tab* a tabulátor karaktert jelenti. (Ebben a fejezetben a „vagy” műveletet a | jellel jelöljük.) A fehér szóközökkel a felismerés után a lexikális elemzőnek semmi teendője nincs, ezt a szimbólumot nem kell továbbadnia a szintaktikus elemzőnek.

A következőkben néhány példát adunk reguláris kifejezésekre.

20.1. példa. Vezessük be a következő jelöléseket: jelöljön D egy tetszőleges számjegyet és L egy tetszőleges betűt, azaz

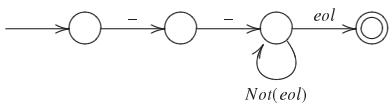
$$D \in \{0, 1, \ldots, 9\}, \text{ és } L \in \{a, b, \ldots, z, A, B, \ldots, Z\} ,$$

a nem látható karaktereket jelöljük a rövid nevükkel, és legyen ε az üres karaktersorozat jele. $Not(a)$ jelentsen egy a-tól különböző karaktert. A reguláris kifejezések:

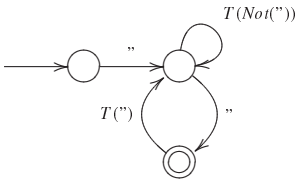
1. valós szám: $(+ \mid - \mid \varepsilon)D^+.D^+(\mathsf{e}(+ \mid - \mid \varepsilon)D^+ \mid \varepsilon)$,
2. pozitív egész és valós szám: $(D^+(\varepsilon \mid .)) \mid (D^*.D^+)$,
3. azonosító szimbólum: $(L \mid \_)(L \mid D \mid \_)^*$,
4. komment: - -$(Not(eol))^* eol$,
5. ## karakterpárokkal határolt komment: ##$((\# \mid \varepsilon)Not(\#))^*$##,
6. karakterstring: ”$(Not(”) \mid ”\,”)^*$ ”.

A 2. és a 3. reguláris kifejezésekhez konstruálható véges determinisztikus automaták a 20.4. és a 20.5. ábrán láthatók.

A lexikális elemző feladata a szimbólum szövegének a meghatározása, azonban például a fenti 6. reguláris kifejezésben nem minden karakter tartozik a szimbólumhoz, a kezdő és a befejező " karakterek nem elemei a szimbólumnak. Ezért a lexikális elemzőhöz rendeljünk hozzá egy puffert, egy szimbólum felismerése után a szimbólumot alkotó karakterek ebben a pufferben lesznek. A determinisztikus véges automatát pedig egészítsük ki egy T átviteli függvénnyel, ahol $T(a)$ jelentse azt, hogy az a karakter a pufferbe kerül.



20.6. ábra. A komment.



20.7. ábra. A karakterstring.

20.2. példa. A 20.1. példa 4. és 6. reguláris kifejezéseire a T függvénnyel kiegészített automaták a 20.6. és a 20.7. ábrákon láthatók. A 4. reguláris kifejezés automatája, mivel egy kommentet ismer fel, egyáltalán nem tartalmaz T függvényt. A 6. kifejezés automatája például az `"Ez egy ""string"""` szövegből az `Ez egy "string"` szöveget viszi a pufferbe.

Most adjuk meg egy determinisztikus véges automatával megadott lexikális elemzés algoritmusát (az egyszerűség kedvéért az egyelemű állapot halmazokat a halmazok egyetlen elemével jelöljük).

Az elemzést végző $A = (Q, \Sigma, \delta, q_0, F)$ determinisztikus véges automata Σ ábécéjét egészítsük ki egy új jellel, jelöljük *egyéb*-bel a nem Σ-beli karaktereket. Ennek megfelelően a δ átmenetfüggvény a következőképpen módosul:

$$\delta'(q,a) = \begin{cases} \delta(q,a), & \text{ha } a \neq \textit{egyéb}\ , \\ \emptyset, & \text{egyébként}\ . \end{cases}$$

Az így kapott A' kiegészített automatával az elemzés a következő lesz:

LEX-ELEMEZ($x\#, A'$)

```
1  q ← q_0, a ← x első karaktere
2  s' ← elemez
3  while a ≠ # és s' = elemez
4        do if δ'(q,a) ≠ ∅
5              then q ← δ'(q,a)
6                   a ← x következő karaktere
7              else  s' ← hiba
8  if s' = elemez és q ∈ F
9     then s' ← O.K.
10    else  s' ← HIBA
11 return s', a
```

Az algoritmus bemenő paramétere a # jellel lezárt elemezendő karaktersorozat és az elemző automata. Az 1. sorban az elemző állapotát az automata q_0 állapotára állítjuk, és meghatározzuk az elemezendő szöveg első karakterét. Az s' változó az elemző működését

jelzi, a 2. sorban a változóba az *elemez* szöveget töltjük. Az 5. sorban az automata állapotátmeneteit hajtjuk végre, és látható, hogy az eredeti automata fenti kiegészítésére azért volt szükség, hogy az automata működése az *egyéb* hibás karakterre is befejeződjék. A 8–10. sorokban *O.K.* azt jelenti, hogy az elemzett karaktersorozat helyes, *HIBA* a lexikális hiba megjelenését jelzi. Sikeres elemzés esetén a a # karaktert, hiba esetén éppen a hibás karaktert tartalmazza.

Megjegyezzük, hogy a LEX-ELEMEZ algoritmus csak egy szimbólumot ismer fel, és ezután működése befejeződik. Egy program nyilvánvalóan sok szimbólumból áll, és egy szimbólum meghatározása után a lexikális elemzést a következő szimbólum meghatározásával kell folytatni, azaz az elemzést az automata kezdőállapotával újra kell indítani. A teljes lexikális elemzés algoritmusának meghatározását a feladatok között tűzzük ki (20-1. feladat).

20.3. példa. Tekintsük a 20.1. példa 3. pontjában szereplő azonosító szimbólumot, a hozzá tartozó automata gráfja a 20.5. ábrán látható. Ha az automata kezdőállapotát 0-val, a végállapotát 1-gyel jelöljük, az automata átmenetfüggvénye a következő táblázattal adható meg:

δ	L	_	D
0	1	1	$\emptyset$
1	1	1	1

Egészítsük ki az automata átmenetfüggvényét:

δ'	L	_	D	*egyéb*
0	1	1	$\emptyset$	$\emptyset$
1	1	1	1	$\emptyset$

Ekkor a LEX-ELEMEZ algoritmus az `abc123#` karaktersorozatra 0111111 állapotsorozatot és *O.K.* jelzést, a `9abc#` bemenetre hibajelzést, az `abc`χ`123` karakterekre 0111 állapotsorozatot és hibajelzést ad.

20.2.2. Speciális problémák

A következőkben a lexikális elemző működése közben fellépő problémákat vizsgáljuk, és ezek megoldásait adjuk meg.

Kulcsszavak, standard szavak

Minden programozási nyelvben vannak olyan azonosítók, amelyeknek speciális célra fenntartott nevük, előre definiált jelentésük van, ezek a ***kulcsszavak.*** A kulcsszavak eredeti jelentésüktől eltérően nem használhatók. Vannak azonban olyan azonosítók is, amelyekre szintén fennáll, hogy előre definiált jelentésük van, de ez a jelentés a programban megváltoztatható. Ezeket a szavakat ***standard szavaknak*** nevezzük.

A kulcsszavak és standard szavak száma programnyelvenként változik. Van olyan programnyelv, amelyben a nulla érték jelölésére például három kulcsszó is van: `zero, zeros` és `zeroes.`

Nézzük meg, hogy a lexikális elemző hogyan ismeri fel a kulcsszavakat és a standard

szavakat, és hogy hogyan különbözteti meg őket a felhasználó által használt azonosítóktól.

A standard szavaknak az eredeti értelemtől eltérő felhasználása nem csak a lexikális elemző feladatát nehezíti meg, hanem a program olvashatóságát is erősen rontja, mint például a következő utasításban:

```
if if then else = then;
```

vagy, hogyha egy `begin` és egy `end` nevű eljárást deklarálunk:

```
begin
  begin; begin end; end; begin end;
end;
```

A kulcsszavak és standard szavak felismerése akkor nagyon egyszerű, ha azokat speciális karakterekkel írjuk, vagy speciális elő- és utókarakterekkel jelöljük meg.

A kulcsszavak kezelésére két módszert adunk.

1. Minden kulcsszót egy reguláris kifejezéssel írunk le, és megadjuk a reguláris kifejezéshez tartozó automata implementációját. Ennek a módszernek a hátránya az, hogy még akkor is nagyon nagyméretű programot kapunk, ha a kulcsszavak leírását az azonos kezdőbetűk szerint összevonjuk.

2. A kulcsszavakat egy külön táblázatban tároljuk. A karaktersorozatban a szavakat egy általános azonosító-felismerővel határozzuk meg, majd egy kereső algoritmust alkalmazva megnézzük, hogy az azonosító benne van-e a táblázatban. Ha igen, akkor a szimbólum egy kulcsszó, ellenkező esetben egy, a felhasználó által definiált azonosító. Ez a módszer nagyon egyszerű, de a keresés sebessége függ a táblázat felépítésétől, a keresési algoritmustól. Egy jól megválasztott leképező függvény és egy lineárisan szétszórt altáblákból felépített kulcsszó-táblázat nagyon hatékony lehet.

Ha a programnyelv lehetővé teszi standard szavak használatát, akkor a lexikális elemző a kulcsszavakra alkalmazott módszerrel meghatározhatja, hogy a vizsgált szimbólum standard szó-e. Az, hogy a standard szó az eredeti jelentésben használt szimbólum, vagy hogy a szimbólumot a felhasználó újradefiniálta, a szimbólum környezetétől függ. Ennek eldöntése a szintaktikus elemző feladata lesz.

Az előreolvasás

Mivel a lexikális elemző a leghosszabb karaktersorozatból álló szimbólum felismerésére törekszik, a szimbólum jobboldali végpontjának meghatározására gyakran egy vagy több karaktert is előre kell olvasnia. Erre egy klasszikus példa a következő két FORTRAN utasítás:

```
DO 10 I = 1.1000
DO 10 I = 1,1000
```

ahol, mivel a FORTRAN nyelvben a szóköz karakterek semmilyen szerepet nem játszanak, az `1` és `1000` közötti jel dönti el, hogy az utasítás egy `DO` kulcsszóval kezdődő ciklusutasítás, vagy a `DO10I` azonosítóra vonatkozó értékadás.

A reguláris kifejezések leírásában vezessük be a szimbólum jobb oldali végpontjának jelölésére a `/` jelet, és nevezzük ezt ***előreolvasási operátornak.*** Így a fenti `DO` kulcsszó

definíciója a következő:

$$\texttt{DO} \,/\, (\textit{betű} \mid \textit{számjegy})^* = (\textit{betű} \mid \textit{számjegy})^* \texttt{,}$$

Ez a definíció tehát azt jelenti, hogy a lexikális elemző csak akkor tudja eldönteni, hogy az első két karakter `D` és `O` betűje a `DO` kulcsszó, ha előreolvasva, betűk vagy számjegyek, egy egyenlőségjel és ismét betűk vagy számjegyek után egy „ `,` ” karaktert talál. Az előreolvasási operátor azt is jelenti, hogy a következő szimbólum keresését a `DO` utáni karakterrel kell kezdeni. Megjegyezzük, hogy az elemző ezzel az előreolvasással a `DO` szimbólumot azonosítja akkor is, ha a `DO` után programhiba van, mint például a `DO2A=3B,` karaktersorozatban, de helyes értékadó utasításban sohasem fogja az azonosító első két `D` és `O` karakterét a `DO` kulcsszónak értelmezni.

Nézzünk egy más típusú példát, ahol az egyszerűség kedvéért csak a pozitív számokat vizsgáljuk. Az egész számok definíciója a valós számok definíciójának prefixe, és a valós számok definíciója a hatványkitevős részt is tartalmazó valós számok definíciójának a prefixe:

pozitív egész : D^+
pozitív valós : $D^+.D^+$
és $D^+.D^+\texttt{e}(+ \mid - \mid \varepsilon)D^+$

Mindhárom reguláris kifejezésre a felismerő automata legyen a leghosszabb szimbólum, azaz a hatványkitevős részt is tartalmazó valós szám automatája.

Az előreolvasás problémája ekkor a következő módon oldható meg. Tegyük a beolvasott karaktereket egy pufferbe, és minden karaktersorozat mellé helyezzünk egy információt: azt, hogy a puffer elejétől vett szöveg nem értelmezhető, „érvénytelen”, vagy ellenkező esetben írjuk ide a felismert szimbólum típusát. Ha az automatával végállapotba jutunk, akkor a felismert szimbólum egy hatványkitevős részt is tartalmazó valós szám. Ha az automatával eljutunk egy olyan állapotba, amelyik nem végállapot, és nem tudunk további karaktert olvasni, akkor a legutolsó érvényes bejegyzéshez tartozó szöveg lesz a felismert szimbólum.

20.4. példa. Tekintsük a `12.3e+f#` karaktersorozatot, ahol a `#` karakter jelzi az elemezendő szöveg végét. Ha az `f` helyén egy pozitív egész szám állna, akkor ez a karaktersorozat egy valós szám lenne. A lexikális elemző pufferének tartalma:

`1`	*egész szám*
`12`	*egész szám*
`12.`	*érvénytelen*
`12.3`	*valós szám*
`12.3e`	*érvénytelen*
`12.3e+`	*érvénytelen*
`12.3e+f`	*érvénytelen*
`12.3e+f#`	

A felismert szimbólum tehát a `12.3` *valós szám*. A lexikális elemzés ezután az `e+f` szöveg elemzésével folytatódik.

Az előreolvasás karakterszámát a programozási nyelv definíciójából lehet meghatározni. A modern programnyelveknél egy szimbólum felismeréséhez a szimbólum karakterein kívül legfeljebb két karakter előreolvasása szükséges.

A szimbólumtábla
Vannak olyan programnyelvek, mint például a C, amelyek a kisbetűs és nagybetűs karaktereket különbözőknek tekintik. Ebben az esetben az azonosító szimbólum betűkaraktereit változtatás nélkül kell felhasználni. Ha a kisbetűs és nagybetűs alakok között a nyelv nem tesz különbséget, akkor az összes karaktert vagy kisbetűs, vagy nagybetűs alakra kell hozni, és ilyen alakban kell tovább feldolgozni. A karakterkonverziót már a forráskezelőben célszerű elvégezni.

Egyszerűbb programnyelveknél a lexikális elemző azonnal felírhatja a megtalált szimbólum karaktereit egy ***szimbólumtáblába,*** ha az még nincs ott. A beírás után, vagy ha a szimbólum már a szimbólumtáblában van, visszaadhatja a szimbólum táblabeli címét, és ez az információ belekerülhet a szimbólumsorozatba is. A szimbólum karaktereire, azaz a szimbólum azonosítására majd a szemantikus elemzésnek és a kódgenerálásnak lesz szüksége.

Direktívák
A forrásnyelvekben a ***direktívák*** a fordítóprogram működésének vezérlésére szolgálnak. A direktívákat és a direktívák operandusaiban szereplő szimbólumokat is a lexikális elemzőnek kell meghatároznia, de ezekkel az elemzőnek további teendői is vannak.

Ha a direktíva például egy feltételes fordítás `if` direktívája, akkor fel kell ismernie a direktíva összes szimbólumát, majd ki kell értékelnie az elágazás feltételét. Ha ez `false`, akkor a további sorokban szereplő szimbólumokat nem szabad elemeznie egészen addig, amíg egy `else`, vagy a feltétel végét jelentő `endif` direktívát nem talál. Ez azt jelenti, hogy a lexikális elemzőnek már szintaktikus és szemantikus ellenőrzéseket is kell végeznie, és kódjellegű információt kell előállítania. A feladatot különösen bonyolíthatja, ha a nyelv lehetőséget ad a feltételek skatulyázására.

Egy másik tipikus direktíva a makróhelyettesítés, vagy egy adott nevű állomány bemásolása a forrásnyelvű szövegbe, aminek a végrehajtása szintén távol áll a lexikális elemző eredeti alapfeladatától.

A legtöbb fordítóprogramban ezeket a problémákat úgy oldják meg, hogy a szintaktikus elemző előtt egy előfeldolgozó programot működtetnek, amelynek a feladata a direktívák által megadott feladatok végrehajtása.

Gyakorlatok

20.2-1. Adjuk meg egy nyelv kommentjének reguláris kifejezését, ha a kommentet a /$*$ és $*$/ karakterpárok határolják, a komment belsejében a / és $*$ karakterek előfordulhatnak, de a $*$/ karakterpár nem.

20.2-2. Vizsgáljuk meg, hogy az előző példában szereplő reguláris kifejezést hogyan kell megváltoztatni, ha a kommentek skatulyázhatók.

20.2-3. Írjunk pozitív valós számokhoz olyan reguláris kifejezést, ami nem engedi meg az elő- és utónullákat. Adjuk meg a reguláris kifejezéshez tartozó determinisztikus véges automatát is.

20.2-4. Írjunk egy olyan programot, ami a lexikális elemzés szimbólumsorozat kimenetéből visszaállítja az eredeti programszöveget. Ügyeljünk a visszaállított karaktersorozatok helyes és szép pozícionálására.

20.3. A szintaktikus elemzés

Egy programnyelv teljes definíciója magában foglalja a nyelv *szintaxisának* és *szemantikájának* definícióját is.

A gyakorlatban használt programnyelvek szintaxisát nem lehet leírni környezetfüggetlen nyelvtanokkal, ez környezetfüggő, kétszintű vagy attribútum nyelvtanokkal lehetséges. Ezekre a nyelvtanokra azonban nincsenek hatékony elemző módszerek, ezért a programnyelvek szintaxisának megadása két részből áll: a szintaxis nagy részét környezetfüggetlen nyelvtannal, egy kisebb részét például környezetfüggő vagy attribútum nyelvtannal írják le. Az első részhez tartozik például a programok struktúrájának és az utasítások felépítésének leírása, a másodikhoz pedig az olyan megkötések, mint például a típusok azonossága egy értékadó utasítás két oldalán, a szimbólumok érvényességi tartományának megadása, vagy az, hogy egy eljárás definíciójában és hívásában a formális és aktuális paraméterek darabszámának meg kell egyeznie.

A környezetfüggetlen nyelvtanokkal leírható tulajdonságok vizsgálatát *szintaktikus elemzésnek* nevezzük. A programnyelv szintaktikájának azon követelményei, amelyek nem írhatók le környezetfüggetlen nyelvtannal, a *statikus szemantikát* alkotják. Ezeknek a tulajdonságoknak az ellenőrzésével a *szemantikus elemző* program foglalkozik.

A hagyományos értelemben vett *szemantikát* a statikus szemantikától való megkülönböztetésére gyakran *run-time, végrehajtási* vagy *dinamikus szemantikának* nevezik. Megadása történhet verbálisan vagy valamilyen interpretáló módszerrel, ahol a program működése az interpreter és környezetének állapotváltozás-sorozatával adható meg.

A továbbiakban először a *környezetfüggetlen nyelvtanokkal* foglalkozunk, ebben az alfejezetben a szintaktikus elemzésre *kiterjesztett* nyelvtanokat fogunk használni. Megvizsgáljuk azt, hogy a környezetfüggetlen nyelvtanokkal leírható tulajdonságok milyen módszerekkel ellenőrizhetők. Először a szintaktikus elemzés alapfogalmat adjuk meg, majd néhány, a gyakorlatban is alkalmazható elemzési algoritmust tanulmányozunk.

20.1. definíció. *Legyen* $G = (N, T, P, S)$ *egy nyelvtan. Ha* $S \overset{*}{\Longrightarrow} \alpha$, *ahol* $\alpha \in (N \cup T)^*$, *akkor az* α*-t* ***mondatformának*** *nevezzük. Ha* $S \overset{*}{\Longrightarrow} x$, *ahol* $x \in T^*$, *akkor az* x *a nyelvtan által definiált nyelv egy* ***mondata****.*

Az elemzés szempontjából a mondatnak kiemelt szerepe van, hiszen a programozó által írt program is terminális szimbólumok sorozata, de ez a szimbólumsorozat csak akkor lesz a nyelvnek egy mondata, ha a program szintaktikusan helyes.

20.2. definíció. *Legyen a* $G = (N, T, P, S)$ *nyelvtannak* $\alpha = \alpha_1\beta\alpha_2$ *egy mondatformája* $(\alpha, \alpha_1, \alpha_2, \beta \in (N \cup T)^*)$. *A* β*-t az* α *egy* ***részmondatának*** *nevezzük, ha van olyan* $A \in N$ *szimbólum, amelyre* $S \overset{*}{\Longrightarrow} \alpha_1 A \alpha_2$ *és* $A \overset{*}{\Longrightarrow} \beta$. *Az* α*-nak* β *egy* ***egyszerű részmondata****, ha a fentiekben az* $A \to \beta \in P$ *teljesül.*

Megjegyezzük, hogy minden mondat egyben mondatforma is, és felhívjuk a figyelmet arra, hogy a mondatforma „részét” is részmondatnak, és nem részmondatformának nevezzük. A legbaloldalibb egyszerű részmondat fontos szerepet játszik a szintaktikus elemzésben, külön elnevezést is kapott.

20.3. definíció. *Egy mondatforma legbaloldalibb egyszerű részmondatát a mondatforma* ***nyelének*** *nevezzük.*

A mondat *szintaxisfájának* levelei a nyelvtan terminális szimbólumai, a szintaxisfa többi pontja a nemterminális szimbólumokat reprezentálja, a gyökérelem pedig a nyelvtan kezdőszimbóluma.

Egy nemegyértelmű nyelvtanban van legalább egy olyan mondat, amelyhez több szintaxisfa tartozik. Ez az elemzés szempontjából azt jelenti, hogy ezt a mondatot többféleképpen is lehet elemezni, azaz a különböző elemzésekhez különböző tárgyprogramok is tartozhatnak. Ezért fordítóprogramokkal csak az *egyértelmű nyelvtanok* által definiált nyelvek fordítását végezzük el.

A továbbiakban feltesszük azt is, hogy a G nyelvtanra a következő feltételek teljesülnek:

1. a nyelvtan *ciklusmentes,* azaz nem tartalmaz $A \overset{+}{\Longrightarrow} A$ $(A \in N)$ helyettesítési szabálysorozatot,

2. a nyelvtan *redukált*, azaz nincs benne „felesleges” nemterminális szimbólum, minden nemterminális szimbólum előfordul legalább egy levezetésben, és minden nemterminális szimbólumból levezethető legalább egy mondatnak egy része, azaz minden $A \in N$-re fennáll, hogy $S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} \alpha y \beta \overset{*}{\Longrightarrow} xyz$, ahol $A \overset{*}{\Longrightarrow} y$ és $|y| > 0$ $(\alpha, \beta \in (N \cup T)^*,\ x, y, z \in T^*)$.

Mint láttuk, a programozó által írt programot a lexikális elemző egy terminális szimbólumokból álló sorozattá alakítja, ez a terminálisokból álló sorozat a szintaktikus elemzés bemenete. A szintaktikus elemzés feladata eldönteni azt, hogy ez a szimbólumsorozat a nyelv egy mondata-e. A szintaktikus elemzőnek ehhez például meg kell határoznia a szimbólumsorozat szintaxisfáját, ismerve a szintaxisfa gyökérelemét és a leveleit, elő kell állítania a szintaxisfa többi pontját és élét, vagyis meg kell határoznia a program egy levezetését.

Ha ez sikerül, akkor azt mondjuk, hogy a program eleme a nyelvnek, azaz a program ***szintaktikusan helyes.***

A továbbiakban csak a *balról-jobbra* haladó elemzésekkel foglalkozunk, azaz azokkal az elemzésekkel, amelyek a program jeleit balról jobbra olvasva dolgozzák fel. A gyakorlatban használt fordítóprogramok mindegyike balról-jobbra történő elemzéssel működik.

A szintaxisfa belső részének felépítésére több módszer létezik. Az egyik az, amikor az S szimbólumból kiindulva építjük fel a szintaxisfát, ezt *felülről-lefelé* haladó elemzésnek nevezzük. Ha a szintaxisfa építése a levelekből kiindulva halad az S szimbólum felé, akkor *alulról-felfelé* elemzésről beszélünk.

A felülről-lefelé haladó modern elemzési módszerekkel a 20.3.1. pontban foglalkozunk, az „igazi” compilerekben jelenleg is használt alulról-felfelé haladó elemzéseket pedig a 20.3.2. pontban tárgyaljuk.

20.3.1. *LL*(1) elemzés

Felülről-lefelé elemezve a nyelvtan kezdőszimbólumától, a szintaxisfa gyökérpontjától indulunk el, és így próbáljuk felépíteni a szintaxisfát. A célunk az, hogy a szintaxisfa levelein az elemezendő programszöveg terminális szimbólumai legyenek.

Most először áttekintjük azokat a fogalmakat, amelyeket a felülről-lefelé elemzésben használunk, majd a táblázatos $LL(1)$ elemzéseket és a rekurzív leszállás módszerét tanulmányozzuk.

Az *LL*(*k*) nyelvtanok

Mivel felülről-lefelé építjük a szintaxisfát és a szövegben balról-jobbra haladunk, ezért arra kell törekednünk, hogy az elemzéskor kapott mondatformák bal oldalán a lehető leghamarabb megjelenjenek az elemezendő szöveg terminálisai.

20.4. definíció. *Ha* $A \to \alpha \in P$, *akkor az* $xA\beta$ *mondatforma* ($x \in T^*$, $\alpha,\beta \in (N \cup T)^*$) ***legbaloldalibb helyettesítése*** $x\alpha\beta$, *azaz*

$$xA\beta \underset{legbal}{\Longrightarrow} x\alpha\beta\ .$$

20.5. definíció. *Ha az* $S \overset{*}{\Longrightarrow} x$ ($x \in T^*$) *levezetésben minden helyettesítés legbaloldalibb helyettesítés, akkor ezt a levezetést* ***legbaloldalibb levezetésnek*** *nevezzük, és így jelöljük:*

$$S \underset{legbal}{\overset{*}{\Longrightarrow}} x\ .$$

A legbaloldalibb levezetésben a terminális szimbólumok a mondatforma bal oldalán jelennek meg, ezért a felülről-lefelé levezetésekben mindig legbaloldalibb helyettesítéseket alkalmazunk, így a felülről-lefelé elemzés a legbaloldalibb levezetésnek felel meg. Ezért a továbbiakban, amikor felülről-lefelé elemzésről lesz szó, a helyettesítéseket és a levezetéseket jelölő nyilak alá már nem is írjuk oda a „*legbal*" szót.

A felülről-lefelé elemzés egyik módszere az lehetne, hogy előállítjuk az összes lehetséges szintaxisfát. Egy szintaxisfa levelein levő szimbólumokat balról-jobbra olvasva a nyelv egy mondatát kapjuk meg. Ha az elemezendő szöveg megegyezik valamelyik mondattal, akkor a mondathoz tartozó szintaxisfáról az elemzés lépései már leolvashatók. Ezt a módszert természetesen nem célszerű és nem is lehet a gyakorlatban megvalósítani.

A gyakorlatban megvalósítható módszer a következő:

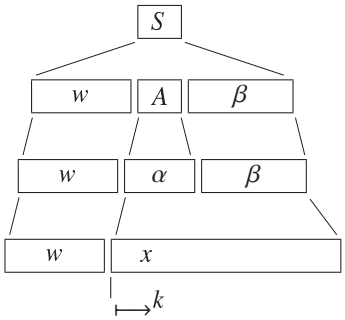
Kiindulunk a kezdőszimbólumból, és megpróbálunk legbaloldalibb helyettesítések egymás utáni alkalmazásával eljutni az elemezendő szöveghez. A szintaxisfa építésekor azonban egyáltalán nem biztos, hogy jó helyettesítési szabályokat alkalmazunk, mert például egy lépés után előfordulhat, hogy a következő lépésben nem találunk alkalmazható szabályt, vagy a mondatforma elejére kerülő terminális szimbólumok nem egyeznek meg az elemezendő szöveg terminális szimbólumaival. A terminális szimbólumokra a következőket állíthatjuk:

20.6. tétel. *Ha* $S \overset{*}{\Longrightarrow} x\alpha \overset{*}{\Longrightarrow} yz$ ($\alpha \in (N \cup T)^*$, $x,y,z \in T^*$) *és* $|x| = |y|$, *akkor* $x = y$.

A tétel állítása triviális, egy mondatforma bal oldalán a terminálisokból álló x sorozatot a környezetfüggetlen nyelvtan helyettesítési szabályai nem változtathatják meg.

A fenti állítás az elemzésben arra használható, hogy megállapíthassuk, ha a szintaxisfa építésekor a bal oldali terminálisok nem egyeznek meg az elemezendő szöveg bal oldalán álló terminálisokkal, akkor a szintaxisfa építése biztosan rossz irányban halad. Ekkor egy lépést vissza kell lépni, és ott egy másik helyettesítési szabályt kell alkalmazni, és még egy lépést vissza kell lépni akkor, ha az adott pontban már nincs több alkalmazható szabály.

Az általános felülről-lefelé elemzést tehát visszalépéses algoritmussal lehet megvalósítani. A visszalépések azonban rendkívül lelassíthatják az elemzést, ezért a továbbiakban csak olyan nyelvtanokkal foglalkozunk, amelyekre visszalépés nélküli elemzések adha-



20.8. ábra. Az $LL(k)$ nyelvtan.

tók meg.

Az $LL(k)$ nyelvtanok alaptulajdonsága az, hogy ha az $S \overset{*}{\Longrightarrow} wx$ $(w, x \in T^*)$ legbaloldalibb levezetés építésekor eljutunk az $S \overset{*}{\Longrightarrow} wA\beta$ mondatformáig $(A \in N,\ \beta \in (N \cup T)^*)$, és az $A\beta \overset{*}{\Longrightarrow} x$-t szeretnénk elérni, akkor az A-ra alkalmazható $A \to \alpha$ helyettesítést egyértelműen meghatározhatjuk, ha ismerjük az x első k darab szimbólumát.

A k szimbólum előreolvasására definiáljuk az $Első_k$ függvényt.

20.7. definíció. *Legyen* $\boldsymbol{Első_k(\alpha)}$ $(k \geq 0,\ \alpha \in (N \cup T)^*)$ *az* α*-ból levezethető szimbólumsorozatok* k *hosszúságú kezdő terminális sorozatainak halmaza, azaz*

$$Első_k(\alpha) = \{x \mid \alpha \overset{*}{\Longrightarrow} x\beta \text{ és } |x| = k\} \cup \{x \mid \alpha \overset{*}{\Longrightarrow} x \text{ és } |x| < k\} \quad (x \in T^*,\ \beta \in (N \cup T)^*)\ .$$

Tehát az $Első_k(x)$ halmaz az x első k darab szimbólumát, $|x| < k$ esetén pedig a teljes x-t tartalmazza. Ha $\alpha \overset{*}{\Longrightarrow} \varepsilon$, akkor természetesen $\varepsilon \in Első_k(\alpha)$.

20.8. definíció. *A* G *nyelvtan* $\boldsymbol{LL(k)}$ ***nyelvtan*** $(k \geq 0)$*, ha bármely két*

$$S \overset{*}{\Longrightarrow} wA\beta \Longrightarrow w\alpha_1\beta \overset{*}{\Longrightarrow} wx\ ,$$
$$S \overset{*}{\Longrightarrow} wA\beta \Longrightarrow w\alpha_2\beta \overset{*}{\Longrightarrow} wy$$

$(A \in N,\ x, y, w \in T^*,\ \alpha_1, \alpha_2, \beta \in (N \cup T)^*)$ *levezetésre*

$$Első_k(x) = Első_k(y)$$

esetén

$$\alpha_1 = \alpha_2\ .$$

A fenti definíció szerint, ha egy nyelvtan $LL(k)$ nyelvtan, akkor a már elemzett w utáni k darab terminális szimbólum az A-ra alkalmazható helyettesítési szabályt egyértelműen meghatározza (20.8. ábra).

A definícióból az is látható, hogy ha egy nyelvtan $LL(k_0)$ nyelvtan, akkor minden $k > k_0$-ra is $LL(k)$ nyelvtan. Ha $LL(k)$ nyelvtanról beszélünk, akkor k alatt mindig azt a legkisebb k-t értjük, amelyre a definícióban megadott tulajdonság teljesül.

20.5. példa. A következő nyelvtan egy $LL(1)$ nyelvtan. Legyen $G = (\{A, S\}, \{a, b\}, P, S)$, ahol a helyettesítési szabályok a következők:

$S \to AS \mid \varepsilon$
$A \to aA \mid b$

Az S szimbólumra az $S \to AS$ szabályt kell alkalmazni, ha az elemezendő szöveg következő szimbóluma a vagy b, és az $S \to \varepsilon$ szabályt, ha a következő szimbólum a # jel.

20.6. példa. A következő nyelvtan egy $LL(2)$ nyelvtan. Legyen $G = (\{A, S\}, \{a, b\}, P, S)$, ahol a helyettesítési szabályok a következők:

$S \to abA \mid \varepsilon$
$A \to Saa \mid b$

Látható, hogy például az

$$S \Longrightarrow abA \Longrightarrow abSaa \overset{S \to abA}{\Longrightarrow} ababAaa$$

és az

$$S \Longrightarrow abA \Longrightarrow abSaa \overset{S \to \varepsilon}{\Longrightarrow} abaa$$

levezetések utolsó lépésében egy szimbólum előreolvasásával mindkét esetben az a-t kapjuk, az S-re alkalmazott szabály csak két szimbólum (az ab és az aa) előreolvasásával határozható meg.

Nem minden környezetfüggetlen nyelvtan $LL(k)$ nyelvtan. Például a következő nyelvtan semmilyen k-ra sem $LL(k)$ nyelvtan.

20.7. példa. A $G = (\{A, B, S\}, \{a, b, c\}, P, S)$ nyelvtan helyettesítési szabályai a következők:

$S \to A \mid B$
$A \to aAb \mid ab$
$B \to aBc \mid ac$

Az $L(G)$ az $a^i b^i$ és $a^i c^i$ $(i \geq 1)$ alakú mondatokat tartalmazza. Az $a^{k+1} b^{k+1}$ elemzésének már a kezdetekor sem lehet eldönteni k szimbólum előreolvasásával, hogy az $S \to A$ és az $S \to B$ közül melyik helyettesítést kell először alkalmazni, mivel minden k-ra $Első_k(a^k b^k) = Első_k(a^k c^k) = a^k$.

Az $LL(k)$ nyelvtan definíciója szerint, ha legbaloldalibb helyettesítésekkel a $wA\beta$ mondatformát kaptuk, akkor a w-t követő k darab terminális szimbólum egyértelműen meghatározza az A-ra alkalmazható szabályt. Ezt mondja ki a következő tétel.

20.9. tétel. *A G nyelvtan akkor és csak akkor $LL(k)$ nyelvtan, ha minden*

$$S \overset{*}{\Longrightarrow} wA\beta, \text{ és } A \to \gamma \mid \delta \quad (\gamma \neq \delta,\ w \in T^*,\ A \in N,\ \beta, \gamma, \delta \in (N \cup T)^*)$$

esetén

$$Első_k(\gamma\beta) \cap Első_k(\delta\beta) = \emptyset\ .$$

Ha a nyelvtanban az A-ra van egy $A \to \varepsilon$ szabály is, akkor a $Első_k$ halmazokban a β-ból származó terminális sorozatok k hosszúságú kezdősorozatai is szerepelnek. Ez pedig azt jelenti, hogy az $LL(k)$ tulajdonság eldöntéséhez nem elegendő csak a szabályokat vizsgálni, hanem a nem feltétlenül véges darabszámú levezetéseket is figyelembe kell venni.

A gyakorlatban jól használható vizsgálati módszert csak az $LL(1)$ nyelvtanokra lehet adni. Ehhez egy új fogalmat definiálunk, megadjuk az egy szimbólumot vagy szimbólumsorozatot követő k hosszúságú terminális sorozatok halmazát.

20.10. definíció. ***Követő$_k(\beta)$***$= \{x \mid S \overset{*}{\Longrightarrow} \alpha\beta\gamma$ *és* $x \in Első_k(\gamma)\}$, *és ha* $\varepsilon \in Követő_k(\beta)$, *akkor legyen* $Követő_k(\beta) = Követő_k(\beta) \setminus \{\varepsilon\} \cup \{\#\}$ $\quad(\alpha, \beta, \gamma \in (N \cup T)^*,\ x \in T^*)$.

A definíció második részében levő átalakítás azért szükséges, mert ha az $\alpha\beta\gamma$ levezetésben a β után nem áll semmilyen szimbólum, azaz $\gamma = \varepsilon$, akkor a β utáni jel csak a mondatokat lezáró # jel lehet.

A $Követő_1(A)$ $(A \in N)$ tehát azokat a terminális szimbólumokat tartalmazza, amelyek az

$$S \overset{*}{\Longrightarrow} \alpha A\gamma \overset{*}{\Longrightarrow} \alpha Aw \ (\alpha, \gamma \in (N \cup T)^*,\ w \in T^*)$$

levezetésben közvetlenül az A szimbólum mögött állhatnak.

20.11. tétel. *A G nyelvtan akkor és csak akkor $LL(1)$ nyelvtan, ha minden A nemterminális szimbólum $A \to \gamma \mid \delta$ helyettesítési szabályaira*

$$Első_1(\gamma Követő_1(A)) \cap Első_1(\delta Követő_1(A)) = \emptyset\ .$$

A tételben az $Első_1(\gamma Követő_1(A))$ kifejezés azt jelenti, hogy a γ-t a $Követő_1(A)$ halmaz minden elemével külön-külön konkatenálni kell, és az így kapott halmaz minden elemére alkalmazni kell az $Első_1$ függvényt.

Látható, hogy az 20.11. tétel jól használható annak eldöntésére, hogy egy nyelvtan vajon $LL(1)$-es-e, hiszen legfeljebb csak annyi halmazt kell a vizsgálathoz előállítani, ahány szabálya van a nyelvtannak.

A továbbiakban az $LL(1)$ nyelvtanok által meghatározott $LL(1)$ nyelvekkel foglalkozunk, az $LL(1)$ nyelvek elemzési módszereit vizsgáljuk. Az egyszerűbb jelölés érdekében az $Első_1$ és $Követő_1$ függvények nevéből az indexet elhagyjuk.

Az $Első(\alpha)$ halmaz elemei a következő algoritmussal határozhatók meg.

Első(α)

```
 1  if α = ε
 2     then F ← {ε}
 3  if α = a, ahol a ∈ T
 4     then F ← {a}
 5  if α = A, ahol A ∈ N
 6     then if A → ε ∈ P
 7             then F ← {ε}
 8             else  F ← ∅
 9          for minden A → Y1Y2 ... Ym ∈ P-re  (m ≥ 1)
10             do F ← F ∪ (Első(Y1) \ {ε})
11                for k ← 1 to m − 1
12                   do if Y1Y2 ... Yk ⟹* ε
13                         then F ← F ∪ (Első(Yk+1) \ {ε})
14                if Y1Y2 ... Ym ⟹* ε
15                   then F ← F ∪ {ε}
```

```
16 if α = Y1Y2 ... Ym (m ≥ 2)
17   then F ← (Első(Y1) \ {ε})
18        for k ← 1 to m − 1
19             do if Y1Y2 ... Yk ⟹* ε
20                  then F ← F ∪ (Első(Yk+1) \ {ε})
21        if Y1Y2 ... Ym ⟹* ε
22          then F ← F ∪ {ε}
23 return F
```

Az 1–4. sorokban az ε és egy a terminális szimbólum argumentumra adjuk meg a halmazt, az 5–15. sorokban az A nemterminális szimbólumra határozzuk meg halmaz elemeit. A 6–7. sorokban és a 14–15. sorokban a halmazba betesszük az ε szimbólumot, ha az A-ból az ε levezethető. Az algoritmus 16–22. sorai arra az esetre adják meg a halmaz elemeit, ha az argumentum egy szimbólumsorozat. Megjegyezzük, hogy a 11. és 18. sor **for** ciklusát már akkor is befejezhetjük, ha $Y_k \in T$, mivel ekkor $Y_1Y_2 \ldots Y_k$-ból biztosan nem vezethető le az ε.

A 20.11 tételben, és a továbbiakban is, szükséges a *Követő*(A) halmaz elemeinek ismerete. Ezeknek a meghatározására a következő algoritmust adjuk.

Követő(A)

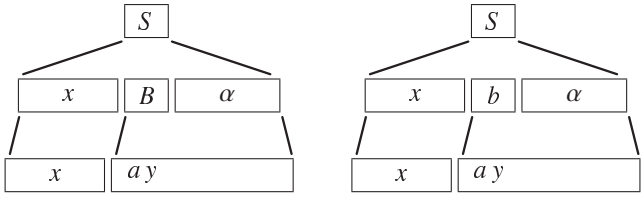
```
 1 if A = S
 2   then F ← {#}
 3   else F ← ∅
 4 for minden B → αAβ ∈ P szabályra
 5     do if |β| > 0
 6          then F ← F ∪ (Első(β) \ {ε})
 7               if β ⟹* ε
 8                 then F ← F ∪ Követő(B)
 9          else F ← F ∪ Követő(B)
10 return F
```

A *Követő*(A) halmaz elemeit az F halmazba helyezzük. A 4–9. sorokban megvizsgáljuk, hogy ha az argumentum egy szabály jobb oldalán szerepel, milyen terminális szimbólumok állhatnak közvetlenül utána. Látható, hogy az ε nem kerülhet bele a halmazba, és a # is csak akkor, ha az argumentum egy mondatforma legjobboldalibb szimbóluma.

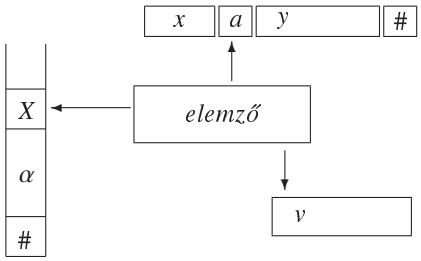
Táblázatos elemzés

Tegyük fel, hogy az elemezendő terminális sorozat xay, amiből az x szöveget már szintaktikus hiba detektálása nélkül elemeztük. Mivel felülről lefelé elemzünk, tehát legbaloldalibb helyettesítéseket alkalmazunk, az elemezendő mondatformánk $xY\alpha$, azaz $xB\alpha$ vagy $xb\alpha$ alakú ($Y \in (N \cup T)$, $B \in N$, $a, b \in T$, $x, y \in T^*$, $\alpha \in (N \cup T)^*$) (20.9. ábra).

Az első esetben a szintaxisfa építésében most a B egy helyettesítése a következő lépés. Ismerjük a bemenő szimbólumsorozat következő elemét, azaz az a-t, ezért egyértelműen meghatározhatjuk, hogy B melyik helyettesítési szabályát kell alkalmaznunk. Pontosan azt a $B \to \beta$ szabályt, amelyikre $a \in$ *Első*$(\beta$*Követő*$(B))$. Ha van ilyen szabály, akkor az $LL(1)$



20.9. ábra. A mondatforma és az elemezendő szöveg.



20.10. ábra. Az *LL*(1) elemző szerkezete.

nyelvtan definíciója alapján pontosan egy van, ha nincs, akkor ez az eset egy *szintaktikus hiba* megtalálását jelenti.

A második esetben a mondatforma következő szimbóluma a b terminális szimbólum, tehát azt várjuk, hogy az elemezendő szöveg következő szimbóluma is b legyen. Ha ez teljesül, azaz $a = b$, akkor az a szimbólum egy helyes szimbólum, továbbléphetünk, mind a mondatformában, mind az elemezendő szövegben az a szimbólum átkerülhet a már elemzett jelsorozatba. Ha $a \neq b$, akkor ez egy *szintaktikus hibát* jelent. Láthatjuk, hogy mindkét szintaktikus hiba esetén ismerjük a hiba helyét is, az a a hibás szimbólum.

Az elemző működését a következőképpen írjuk le. Jelöljük a # jellel az elemezendő szöveg jobb oldali végpontját, azaz legyen a # az elemezendő szöveg utolsó szimbóluma. Az elemzéshez egy vermet is használunk, a verem alját jelöljük szintén egy # jellel. A helyettesítési szabályokat valamilyen sorrendben, például a felsorolásuk sorrendjében sorszámozzuk meg. Az elemzés során alkalmazott szabályok sorszámát felírjuk egy listába, az elemzés végén ez a lista arra fog szolgálni, hogy az elemzett szöveg szintaxisfáját felépíthessük (20.10. ábra).

Az *elemzés állapotait* az $(ay\#, X\alpha\#, v)$ hármassal jelöljük. Az $ay\#$ a még nem elemzett szöveg, $X\alpha\#$ az elemzés mondatformájának még nem elemzett része, ez van a veremben, úgyhogy X van a verem tetején, v pedig a szabályok sorszámait tartalmazó lista. Az elemzés úgy fog működni, hogy mindig a verem tetején levő X szimbólumot és a még nem elemzett szöveg első szimbólumát, az a-t fogjuk vizsgálni. Az a-t *aktuális szimbólumnak* nevezzük. A verem tetejére és az aktuális szimbólumra az elemzőből egy-egy pointer mutat.

Mivel felülről lefelé elemzünk, a verem kezdeti tartalma legyen $S\#$. Ha a teljes elemezendő szöveget xay-nal jelöljük, akkor az elemzés kezdetén az elemzés állapotát, azaz a *kezdőállapotot* az $(xay\#, S\#, \varepsilon)$ hármassal írhatjuk le, ahol most ε az üres listát jelöli.

Az elemzést egy T *elemző táblázat* segítségével fogjuk végezni. A táblázat sorai a verem tetején álló szimbólumot, az oszlopai pedig az input következő elemezendő szimbólumát jelölik, a # jelet a táblázat utolsó sorához és utolsó oszlopához írjuk. Így tehát a táblázat sorainak a száma eggyel nagyobb a nyelvtan szimbólumainak a számánál, az oszlopok száma pedig eggyel nagyobb a terminális szimbólumok számánál.

A táblázat $T[X, a]$ eleme legyen a következő:

$$T[X,a] = \begin{cases} (\beta, i), & \text{ha } X \to \beta \text{ az } i\text{-edik helyettesítési szabály ,} \\ & a \in \mathit{Első}(\beta) \text{ vagy} \\ & (\varepsilon \in \mathit{Első}(\beta) \text{ és } a \in \mathit{Követő}(X)) \text{ ,} \\ \mathit{pop}, & \text{ha } X = a \text{ ,} \\ \mathit{elfogad}, & \text{ha } X = \# \text{ és } a = \# \text{ ,} \\ \mathit{hiba} & \text{egyébként .} \end{cases}$$

A táblázat kitöltését a következő algoritmussal végezhetjük:

LL(1)-TÁBLÁZATOT-KITÖLT(G)

```
 1  for minden A ∈ N-re
 2      do if A → α ∈ P az i-edik szabály
 3          then for minden a ∈ Első(α)-ra
 4                  do T[A, a] ← (α, i)
 5                  if ε ∈ Első(α)
 6                     then for minden a ∈ Követő(A)-ra
 7                             do T[A, a] ← (α, i)
 8  for minden a ∈ T-re
 9      do T[a, a] ← pop
10  T[#, #] ← elfogad
11  for minden X ∈ (N ∪ T ∪ {#}) és minden a ∈ (T ∪ {#})-ra
12      do if T[X, a] = „üres”
13          then T[X, a] ← hiba
14  return T
```

A 10. sorban a táblázat jobb alsó sorába az *elfogad* szöveget írjuk, a 8–9. sorokban a terminálisokkal címkézett sorok és oszlopok rész-táblázatának főátlójába a *pop* szöveget írjuk, az 1–7. sorokban levő algoritmussal a megadott pozícióra a szabály jobb oldalának szimbólumai és a szabály sorszáma kerül. Az algoritmus 12–13. sorában a táblázat üresen maradt helyeire a szintaktikus hibát jelző *hiba* szöveget írjuk.

Az elemzés működése állapotátmenetekkel adható meg. A kezdőállapot tehát $(x\#, S\#, \varepsilon)$, ha az elemezendő szöveg x, és az elemzés sikeresen befejeződik akkor, ha az elemző a $(\#, \#, w)$ állapotba, a *végállapotba* kerül. Ha a még nem elemzett szöveg az $ay\#$, és a verem tetején az X szimbólum áll, az állapotátmenetek a következők:

$$(ay\#, X\alpha\#, v) \to \begin{cases} (ay\#, \beta\alpha\#, vi), & \text{ha } T[X, a] = (\beta, i) \text{ ,} \\ (y\#, \alpha\#, v), & \text{ha } T[X, a] = \mathit{pop} \text{ ,} \\ \mathit{O.K.}, & \text{ha } T[X, a] = \mathit{elfogad} \text{ ,} \\ \mathit{HIBA}, & \text{ha } T[X, a] = \mathit{hiba} \text{ .} \end{cases}$$

Az *O.K.* azt jelenti, hogy az elemzett szimbólumsorozat szintaktikusan helyes, a *HIBA* pedig

egy szintaktikus hiba detektálását jelzi.

Az elemző működésére a következő algoritmust adhatjuk meg.

LL(1)-ELEMEZ($xay\#, T$)

```
 1  s ← (xay#, S#, ε), s' ← elemez
 2  repeat
 3        if s = (ay#, Aα#, v) és T[A, a] = (β, i)
 4          then s ← (ay#, βα#, vi)
 5          else if s = (ay#, aα#, v)
 6                  then s ← (y#, α#, v)              ▷ Ekkor T[a, a] = pop.
 7                  else if s = (#, #, v)
 8                          then s' ← O.K.            ▷ Ekkor T[#, #] = elfogad.
 9                          else s' ← HIBA            ▷ Ekkor T[A, a] = hiba.
10  until s' = O.K. vagy s' = HIBA
11  return s', s
```

Az algoritmus bemenő paramétere az xay elemezendő szöveg és a T elemző táblázat. Az s' változó az elemző működését jelzi, működés közben az s' értéke *elemez,* az elemzés befejezésekor *O.K.* vagy *HIBA*. Az elemző az elemzett szöveg a aktuális szimbóluma és a verem tetején levő szimbólum alapján a T táblázatból meghatározza az elvégzendő műveletet. A 3–4. sorban a szintaxisfát építi az $A \to \beta$ szabály alapján. Az 5–6. sorban léptetés történik, mivel a verem tetején is az a szimbólum található. Az algoritmus a 8–9. sorban az elemzés befejezését jelzi, ha a verem kiürült és az elemezendő szöveg végére ért, akkor az elemzett szöveg helyes, egyebként az elemző egy szintaktikus hibát fedezett fel. Az algoritmus végeredménye ennek megfelelően az *O.K.* vagy *HIBA* jelzés, és kimenetként mindkét esetben megjelenik az elemző állapotának s hármasa is. Helyes szöveg esetén a hármas harmadik eleméből, v-ből a szabályok sorszámai alapján felépíthető a szintaxisfa, szintaktikus hiba esetén a hármas első elemének első szimbóluma a hiba helyét adja meg.

20.8. példa. Legyen a G nyelvtan a következő: $G = (\{E, E', T, T', F\}, \{+, *, (,), i\}, P, E)$, ahol a P helyettesítési szabályok a következők:

$E \to TE'$

$E' \to +TE' \mid \varepsilon$

$T \to FT'$

$T' \to *FT' \mid \varepsilon$

$F \to (E) \mid i$

A szabályokból a *Követő*(A) halmazok meghatározhatók, az elemző táblázat kitöltéséhez a következő halmazok szükségesek:

Első(TE') = {(, i},

Első($+TE'$) = {+},

Első(FT') = {(, i},

Első($*FT'$) = {*},

Első((E)) = {(},

Első(i) = {i},

Követő(E') = {), #},

Követő(T') = {+,), #}.

Az elemző táblázat a következő, a táblázatban az üres helyek a *hibát* jelentik.

	+	*	(	)	i	#
E			$(TE',1)$		$(TE',1)$	
E'	$(+TE',2)$			$(\varepsilon,3)$		$(\varepsilon,3)$
T			$(FT',4)$		$(FT',4)$	
T'	$(\varepsilon,6)$	$(*FT',5)$		$(\varepsilon,6)$		$(\varepsilon,6)$
F			$((E),7)$		$(i,8)$	
+	*pop*					
*		*pop*				
(			*pop*			
)				*pop*		
i					*pop*	
#						*elfogad*

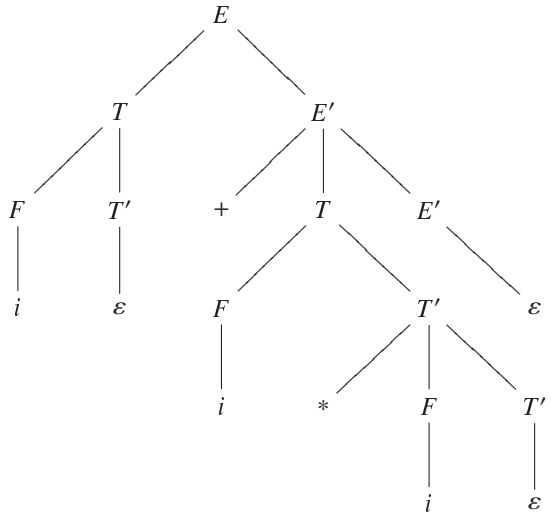
20.9. példa. Az előző példában szereplő nyelvtan elemző táblázatának felhasználásával elemezzük az $i+i*i$ szöveget. Az elemzés a következő:

$$
\begin{array}{lcllll}
(i+i*i\#, S\#, \varepsilon) & \xrightarrow{(TE',1)} & (\ i+i*i\#, & TE'\#, & 1 &) \\
& \xrightarrow{(FT',4)} & (\ i+i*i\#, & FT'E'\#, & 14 &) \\
& \xrightarrow{(i,8)} & (\ i+i*i\#, & iT'E'\#, & 148 &) \\
& \xrightarrow{pop} & (\ +i*i\#, & T'E'\#, & 148 &) \\
& \xrightarrow{(\varepsilon,6)} & (\ +i*i\#, & E'\#, & 1486 &) \\
& \xrightarrow{(+TE',2)} & (\ +i*i\#, & +TE'\#, & 14862 &) \\
& \xrightarrow{pop} & (\ i*i\#, & TE'\#, & 14862 &) \\
& \xrightarrow{(FT',4)} & (\ i*i\#, & FT'E'\#, & 148624 &) \\
& \xrightarrow{(i,8)} & (\ i*i\#, & iT'E'\#, & 1486248 &) \\
& \xrightarrow{pop} & (\ *i\#, & T'E'\#, & 1486248 &) \\
& \xrightarrow{(*FT',5)} & (\ *i\#, & *FT'E'\#, & 14862485 &) \\
& \xrightarrow{pop} & (\ i\#, & FT'E'\#, & 14862485 &) \\
& \xrightarrow{(i,8)} & (\ i\#, & iT'E'\#, & 148624858 &) \\
& \xrightarrow{pop} & (\ \#, & T'E'\#, & 148624858 &) \\
& \xrightarrow{(\varepsilon,6)} & (\ \#, & E'\#, & 1486248586 &) \\
& \xrightarrow{(\varepsilon,3)} & (\ \#, & \#, & 14862485863 &) \\
& \xrightarrow{elfogad} & O.K. & & &
\end{array}
$$

Az elemzett mondat szintaxisfája az 20.11. ábrán látható.

A rekurzív leszállás módszere

A visszalépés nélküli felülről-lefelé elemzésekre a táblázatos módszeren kívül gyakran alkalmazunk egy olyan módszert, amelynek lényege az, hogy a nyelvtanhoz egy programot rendelünk. A nyelvtan szimbólumaihoz eljárásokat adunk meg, és elemzés közben a rekurzív eljáráshívásokon keresztül a programnyelv implementációja valósítja meg az elemző



20.11. ábra. Az $i + i * i$ mondat szintaxisfája.

vermét és a veremkezelést. A felülről-lefelé elemzés és a rekurzív eljáráshívások miatt ezt a módszert a ***rekurzív leszállás módszerének*** nevezzük.

A terminális szimbólumok vizsgálatára vezessük be az *Vizsgál* eljárást. Legyen az eljárás paramétere a „várt szimbólum”, azaz a mondatforma legbaloldalibb, még nem vizsgált terminális szimbóluma, és tartalmazza az *aktuális_szimbólum* globális változó a vizsgált terminális sorozat soron következő szimbólumát.

```
procedure Vizsgal(a);
begin
   if aktu lis_szimb lum = a
      then K vetkez _szimb lum
      else Hibajelz s
end;
```

A *Következő_szimbólum* az előreolvasásra szolgál, ez a lexikális elemzőt meghívó eljárás neve. Ez az eljárás a bemenő szimbólumsorozatból meghatározza a következő szimbólumot és ezt az *aktuális_szimbólum* változóba tölti. A *Hibajelzés* eljárás szintaktikus hibajelzést ad, és ezután befejezi az elemző program futását.

A nyelvtan minden nemterminális szimbólumához rendeljünk hozzá egy eljárást. Az A szimbólumhoz tartozó eljárás legyen a következő:

```
procedure  A;
begin
   T(A)
end;
```

ahol `T(A)`-t az A-ra vonatkozó helyettesítési szabályok jobb oldalán álló szimbólumok határozzák meg.

Az elemzéshez használt nyelvtanok *redukáltak,* ami többek között azt jelenti, hogy nincs bennük „felesleges” nemterminális szimbólum, minden nemterminális szimbólum szerepel legalább egy helyettesítési szabály bal oldalán. Tehát ha az A szimbólumot vizsgáljuk, biztosan van legalább egy $A \to \alpha$ helyettesítési szabály.

1. Ha az A szimbólumra csak egy helyettesítési szabály van:

   (a) az $A \to a$ szabályhoz rendelt program legyen a `Vizsgal(a)`,

   (b) az $A \to B$ szabályhoz rendeljük hozzá a `B` eljáráshívást,

   (c) az $A \to X_1 X_2 \ldots X_n$ $(n \geq 2)$ szabályhoz tartozzon a következő blokk:

```
begin
   T(X_1);
   T(X_2);
   ...
   T(X_n)
end;
```

2. Ha az A szimbólumra több helyettesítési szabály van:

   (a) Ha az $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ szabályok ε-mentesek, azaz α_i-ből $(1 \leq i \leq n)$ nem vezethető le az ε, akkor `T(A)` legyen

```
case aktualis_szimbolum of
   Elso(alpha_1) : T(alpha_1);
   Elso(alpha_2) : T(alpha_2);
   ...
   Elso(alpha_n) : T(alpha_n)
end;
```

   ahol `Elso(alpha_i)` az *Első*(α_i) programbeli jelölése. Felhívjuk a figyelmet arra, hogy a rekurzív leszállás módszerében most először használjuk ki azt, hogy a nyelvtan $LL(1)$-es.

   (b) Az $LL(1)$ nyelvtan programnyelvet ír le, ezért a nyelvtan ε-mentességét nem célszerű megkövetelni. Az $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_{n-1} \mid \varepsilon$ szabályokhoz a következő `T(A)` programot rendeljük:

```
case aktualis_szimbolum of
   Elso(alpha_1)     : T(alpha_1);
   Elso(alpha_2)     : T(alpha_2);
   ...
   Elso(alpha_(n-1)) : T(alpha_(n-1));
   Koveto(A)         : skip
end;
```

   ahol `Koveto(A)` a *Követő*(A)-nak felel meg.

   Speciálisan, ha az $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ szabály esetén egy i-re $(1 \leq i \leq n)$ $\alpha_i \overset{*}{\Longrightarrow} \varepsilon$, azaz $\varepsilon \in$ *Első*(α_i), akkor a `case` utasítás i-edik sora lesz a `Koveto(A) : skip` sor.

A `T(A)`-ban a `case` helyett, ha lehetséges, használhatunk `if-then-else` vagy `while` utasítást is.

A rekurzív leszállás módszerével készített elemző program kezdő eljárása, azaz főprogramja a nyelvtan kezdőszimbólumához írt eljárás lesz.

A rekurzív leszállás módszerével működő elemző programot a következő REK-LESZÁLL-KÉSZÍT algoritmussal hozhatjuk létre. Az algoritmus bemenete a G nyelvtan, és az algoritmus eredményül az elemző P programját adja. Az algoritmusban használunk egy PROGRAMOT-ÍR eljárást, ami az argumentumában megadott programsorokat a már meglévő P programhoz fűzi. Ezt az algoritmust nem részletezzük.

REK-LESZÁLL-KÉSZÍT(G)

```
 1  P ← ∅
 2  PROGRAMOT-ÍR(
 3       procedure Vizsgal(a);
 4       begin
 5       if aktualis_szimbolum = a
 6          then Kovetkezo_szimbolum
 7          else Hibajelzes
 8       end;
 9       )
10  for a G nyelvtan minden A ∈ N szimbólumára
11      do if A = S
12           then PROGRAMOT-ÍR(
13                     program S;
14                     begin
15                      REK-LESZÁLL-UT(S, P)
16                     end.
17                     )
18           else PROGRAMOT-ÍR(
19                    procedure A;
20                    begin
21                     REK-LESZÁLL-UT(A, P)
22                    end;
23                    )
24  return P
```

Az algoritmus a 2–9. sorokban elkészíti a *Vizsgál* eljárást, majd a bemeneteként megadott G nyelvtan minden nemterminális szimbólumára a REK-LESZÁLL-UT algoritmus felhasználásával készíti a szimbólumhoz tartozó eljárást. A 11–17. sorokban látható, hogy a nyelvtan kezdőszimbólumához az elemző főprogramja fog tartozni. Az algoritmus kimenete az elemző program lesz.

REK-LESZÁLL-UT(A, P)

```
1  if csak egy A → α szabály van
2     then REK-LESZÁLL-UT1(α, P)                          ▷ A → α.
3     else REK-LESZÁLL-UT2(A, (α_1, ..., α_n), P)         ▷ A → α_1 | ··· | α_n.
4  return P
```

Mivel a létrehozandó elemző program utasításai lényegesen függnek attól, hogy az A nemterminális szimbólumra a nyelvtanban hány helyettesítési szabály van, a REK-LESZÁLL-UT algoritmus a további műveleteket két részre osztja. A REK-LESZÁLL-UT1 algoritmus foglalkozik azzal az esettel, amikor csak egy helyettesítési szabály van, és a REK-LESZÁLL-UT2 készíti az alternatívákra vonatkozó elemző programot.

REK-LESZÁLL-UT1(α, P)

```
 1  if α = a
 2     then PROGRAMOT-ÍR(
 3               Vizsgal(a)
 4               )
 5  if α = B
 6     then PROGRAMOT-ÍR(
 7               B
 8               )
 9  if α = X_1 X_2 ... X_n (n ≥ 2)
10     then PROGRAMOT-ÍR(
11               begin
12                REK-LESZÁLL-UT1(X_1, P) ;
13                REK-LESZÁLL-UT1(X_2, P) ;
14                ...
15                REK-LESZÁLL-UT1(X_n, P)
16               end;
17  return P
```

REK-LESZÁLL-UT2($A, (\alpha_1, \ldots, \alpha_n), P$)

```
1  if α_1, ..., α_n szabályok ε-mentesek
2     then PROGRAMOT-ÍR(
3                case aktualis_szimbolum of
4                 Elso(alpha_1) : REK-LESZÁLL-UT1(α_1, P) ;
5                 ...
6                 Elso(alpha_n) : REK-LESZÁLL-UT1(α_n, P)
7                end;
8                )
```

```
 9  if van ε-szabály, α_i = ε (1 ≤ i ≤ n)
10     then Programot-ír(
11                  case aktualis_szimbolum of
12                   Elso(alpha_1)      : Rek-leszáll-ut1(α_1, P) ;
13                   ...
14                   Elso(alpha_(i-1))  : Rek-leszáll-ut1(α_{i-1}, P) ;
15                   Koveto(A)          : skip;
16                   Elso(alpha_(i1)) : 2Rek-leszáll-ut1(α_{i+1}, P) ;
17                   ...
18                   Elso(alpha_n)      : Rek-leszáll-ut1(α_1, P)
19                  end;
20                  )
21  return P
```

A fenti két algoritmus a korábban már részletesen leírt programot hozza létre.

Az elemezendő szöveg végének az ellenőrzése a rekurzív leszállás módszerével úgy valósítható meg, hogy a szöveg végét jelző # szimbólumot beépítjük egy új helyettesítési szabályba. Ha a nyelvtan kezdőszimbóluma S, akkor létrehozunk egy $S' \to S\#$ szabályt, és az új S' lesz az új nyelvtan kezdőszimbóluma. A # szimbólumot terminális szimbólumnak tekintjük. Az így kibővített nyelvtanra készítjük el a rekurzív leszállás elemző programját.

20.10. példa. A 20.8. példában szereplő nyelvtant egészítsük ki a fenti módon. A szabályok tehát a következők.

$S' \to E\#$
$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to (E) \mid i$

A 20.8. példában megadtuk a táblázatos elemző létrehozásához szükséges *Első* és *Követő* halmazokat. Ezek közül most a következőkre van szükség:

Első$(+TE') = \{+\}$,
Első$(*FT') = \{*\}$,
Első$((E)) = \{(\}$,
Első$(i) = \{i\}$,
Követő$(E') = \{), \#\}$,
Követő$(T') = \{+,), \#\}$.

A halmazok felhasználásának helyét a program sorainak kommentjeiben adjuk meg, a kommenteket a -- karakterpárral kezdjük.

A rekurzív leszállás módszerének elemző programja a következő lesz.

```
program S';
begin
   E;
   Vizsgal(#)
end.
procedure E;
begin
   T;
   E'
end;
procedure E';
begin
   case aktualis_szimbolum of
   +     : begin                -- Elso(+TE')
               Vizsgal(+);
               T;
               E'
           end;
   ),#   : skip                 -- Koveto(E')
   end
end;
procedure T;
begin
   F;
   T'
end;
procedure T';
begin
   case aktualis_szimbolum of
   *     : begin                -- Elso(*FT')
               Vizsgal(*);
               F;
               T'
           end;
   +,),# : skip                 -- Koveto(T')
   end
end;
procedure F;
begin
   case aktualis_szimbolum of
   (     : begin                -- Elso((E))
               Vizsgal(();
               E;
               Vizsgal())
           end;
   i     : Vizsgal(i)            -- Elso(i)
   end
end;
```

Látható, hogy az elemző főprogramja a nyelvtan S' kezdőszimbólumához tartozó eljárás lett.

20.3.2. *LR*(1) elemzés

Alulról-felfelé elemezve az elemezendő szimbólumsorozatból indulunk ki, megkeressük a mondatforma nyelét (a nyél definícióját már a 20.3. definícióban megadtuk), és ezt a nyelet helyettesítjük a hozzátartozó nemterminális szimbólummal. Ezt ismételve próbáljuk felépíteni a szintaxisfát. A célunk az, hogy elérjük a nyelvtan kezdőszimbólumát, ez lesz majd a szintaxisfa gyökérpontja, a fa levelein pedig az elemezendő programszöveg terminális szimbólumai lesznek.

Először áttekintjük azokat a fogalmakat, amelyeket az alulról-felfelé haladó elemzésekben használunk.

Alulról-felfelé elemezve mindig a mondatforma nyelét kell meghatároznunk. A probléma tehát az, hogy hogyan lehet a nyelet meghatározni, és ha a nyél helyettesítésére több lehetőség is van, akkor a nyelet melyik nemterminális szimbólummal kell helyettesíteni.

20.12. definíció. *Ha* $A \to \alpha \in P$, *akkor a* βAx *mondatforma* ($x \in T^*$, $\alpha,\beta \in (N \cup T)^*$) ***legjobboldalibb helyettesítése*** $\beta\alpha x$, *azaz*

$$\beta Ax \underset{legjobb}{\Longrightarrow} \beta\alpha x\ .$$

20.13. definíció. *Ha az* $S \overset{*}{\Longrightarrow} x$ ($x \in T^*$) *levezetésben minden helyettesítés legjobboldalibb helyettesítés, akkor ezt a levezetést* ***legjobboldalibb levezetésnek*** *nevezzük, és így jelöljük:*

$$S \underset{legjobb}{\overset{*}{\Longrightarrow}} x\ .$$

A legjobboldalibb levezetésben a terminális szimbólumok a mondatforma jobb oldalán jelennek meg. A nyél és a legjobboldalibb helyettesítés kapcsolata alapján, ha a legjobboldalibb levezetés lépéseit „visszafelé” alkalmazzuk, akkor éppen az alulról-felfelé haladó elemzés lépéseit kapjuk meg. Az alulról-felfelé elemzés tehát a legjobboldalibb levezetés „inverzének” felel meg. Ezért a továbbiakban, amikor alulról-felfelé elemzésről lesz szó, a helyettesítéseket és a levezetéseket jelölő nyilak alá már nem is írjuk oda a „*legjobb*” szót.

Az általános alulról-felfelé elemzést, a felülről-lefelé elemzésekhez hasonlóan, visszalépéses algoritmussal lehet megvalósítani. A visszalépések azonban rendkívül lelassíthatják az elemzést, ezért csak olyan nyelvtanokkal fogunk foglalkozni, amelyekre visszalépés nélküli elemzések adhatók meg.

A további alfejezetekben bemutatunk egy hatékony, a környezetfüggetlen nyelvtanok igen nagy osztályára alkalmazható elemzési módszert. Ez a nyelvtan-osztály a gyakorlatban használt programnyelvek nyelvtanait is tartalmazza.

Az elemzést *LR*(k) elemzésnek, a nyelvtant *LR*(k) nyelvtannak nevezzük, ahol az *LR* a balról jobbra („**L**eft to **R**ight”) történő elemzésre utal, a k pedig azt jelenti, hogy k szimbólumot előreolvasva egyértelműen meghatározható a mondatforma nyele. Az *LR*(k) elemzés visszalépés nélküli, léptetés-redukálás típusú elemzés.

Mint majd látni fogjuk, elegendő az *LR*(1)-es elemzőkkel foglalkoznunk, mivel minden *LR*(k) ($k > 1$) nyelvtanhoz létezik vele ekvivalens *LR*(1) nyelvtan. Ez rendkívül fontos számunkra, mivel így egy szöveg elemzésekor mindig elég csak egy szimbólumot előreolvasni.

Az $LR(k)$ elemzés hátrányának hozható fel, hogy az elemző táblázatának „kézi” megkonstruálása nem könnyű. Léteznek azonban olyan programok (például a UNIX yacc programja), amelyek egy adott nyelvtan szabályaiból létrehozzák a teljes elemző programot, és így az elemző megírása sem jelent problémát.

Az $LR(k)$ nyelvtanok vizsgálata után az $LALR(1)$ elemzést, a programnyelvek fordítóprogramjaiban jelenleg használt elemzési módszert tanulmányozzuk.

Az $LR(k)$ nyelvtanok

Mint már korábban is tettük, jelöljük az elemezendő szöveg, az elemezendő terminális sorozat jobb oldalát a # szimbólummal. Vezessünk be egy új S' nemterminális szimbólumot és egy új $S' \to S$ szabályt.

20.14. definíció. *Legyen a $G = (N, T, P, S)$ nyelvtanhoz tartozó G' **kiegészített nyelvtan** a következő:*

$$G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S') \ .$$

Sorszámozzuk meg a helyettesítési szabályokat, az $S' \to S$ szabály legyen a nulladik szabály. Így, ha redukáláskor a nulladik szabályt kell alkalmazni, akkor ez az elemzés végét, és az elemzett szöveg szintaktikus helyességét fogja jelenteni.

Megjegyezzük, hogy ha az eredeti S kezdőszimbólum nem szerepel egyik helyettesítési szabály jobb oldalán sem, akkor az $S' \to S$ kiegészítésre nincs is szükség. Az általánosság kedvéért azonban az $LR(k)$ tulajdonságot csak kiegészített nyelvtanokra definiáljuk.

20.15. definíció. *Egy G' kiegészített nyelvtan **$LR(k)$ nyelvtan** ($k \geq 0$), ha bármely két*

$$S' \overset{*}{\Longrightarrow} \alpha A w \Longrightarrow \alpha\beta w \ ,$$
$$S' \overset{*}{\Longrightarrow} \gamma B x \Longrightarrow \gamma\delta x = \alpha\beta y$$

$(A, B \in N,\ x, y, w \in T^*,\ \alpha, \beta, \gamma, \delta \in (N \cup T)^*)$ levezetésre

$$Első_k(w) = Első_k(y)$$

esetén

$$\alpha = \gamma, A = B \text{ és } x = y \ .$$

Az $LR(k)$ nyelvtanokra az a jellemző, hogy az $\alpha\beta w$ mondatformában a w első szimbólumától kezdve előreolvasva k darab szimbólumot, egyértelműen meghatározható, hogy valóban β a nyél, és az, hogy az $A \to \beta$ szabállyal kell redukálni, azaz az $\alpha\beta w$ mondatforma az $\alpha A w$ mondatformára redukálható.

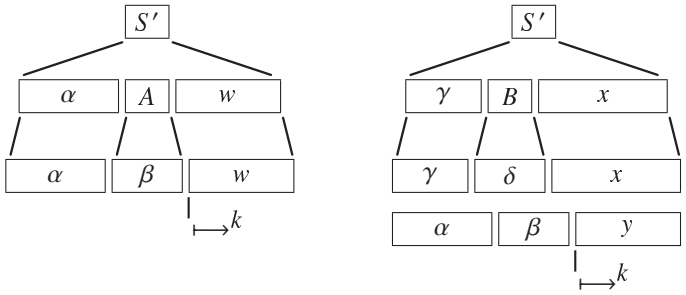
Tegyük fel ugyanis, hogy az $\alpha\beta w$ és az $\alpha\beta y$ mondatformákban, amelyeknek tehát az $\alpha\beta$ prefixük azonos, $Első_k(w) = Első_k(y)$, és mégis az $\alpha\beta w$ az $\alpha A w$-re, az $\alpha\beta y$ pedig a $\gamma B x$-re redukálható. Az $LR(k)$ tulajdonság miatt ekkor csak $\alpha = \gamma$ és $A = B$ lehet. Ez azt jelenti, hogy a nyél vagy soha nem a β, vagy pedig mindig az (20.12. ábra).

20.11. példa. A $G' = (\{S', S\}, \{a\}, P', S')$ nyelvtan, ahol a helyettesítési szabályok

$S' \to S$

$S \to Sa \mid a$

nem $LR(0)$ nyelvtan, mivel, feltüntetve a definíció jelöléseit,



20.12. ábra. Az $LR(k)$ nyelvtan.

$$S' \overset{*}{\Longrightarrow} \begin{matrix}\varepsilon & S' & \varepsilon\\ \alpha & A & w\end{matrix} \Longrightarrow \begin{matrix}\varepsilon & S & \varepsilon,\\ \alpha & \beta & w\end{matrix}$$

$$S' \overset{*}{\Longrightarrow} \begin{matrix}\varepsilon & S' & \varepsilon\\ \gamma & B & x\end{matrix} \Longrightarrow \begin{matrix}\varepsilon & Sa & \varepsilon\\ \gamma & \delta & x\end{matrix} = \begin{matrix}\varepsilon & S & a,\\ \alpha & \beta & y\end{matrix}$$

esetén $Első_0(\varepsilon) = Első_0(a) = \varepsilon$, de $\gamma Bx \neq \alpha Ay$.

20.12. példa. A következő nyelvtan egy $LR(1)$ nyelvtan. $G = (\{S', S\}, \{a, b\}, P', S')$, ahol a helyettesítési szabályok:

$S' \to S$

$S \to SaSb \mid \varepsilon$

A következő példában megmutatjuk, hogy van olyan környezetfüggetlen nyelvtan, amelyik nem $LR(k)$ nyelvtan egyetlen k-ra sem ($k \geq 0$).

20.13. példa. Legyenek a $G' = (\{S', S\}, \{a\}, P', S')$ nyelvtan helyettesítési szabályai a következők:

$S' \to S$

$S \to aSa \mid a$

Ekkor minden k-ra ($k \geq 0$)

$$S' \overset{*}{\Longrightarrow} a^k S a^k \Longrightarrow a^k a a^k = a^{2k+1},$$

$$S' \overset{*}{\Longrightarrow} a^{k+1} S a^{k+1} \Longrightarrow a^{k+1} a a^{k+1} = a^{2k+3},$$

és

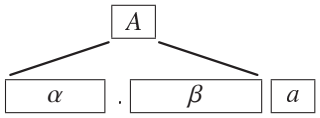
$$Első_k(a^k) = Első_k(aa^{k+1}) = a^k,$$

de

$$a^{k+1} S a^{k+1} \neq a^k S a^{k+2}.$$

Míg egy tetszőleges $LL(k)$ ($k > 1$) nyelvtanra nem biztos, hogy lehet vele ekvivalens $LL(1)$ nyelvtant megadni, addig az $LR(k)$ nyelvtanokra jobb eredményt lehet elérni:

20.16. tétel. *Minden $LR(k)$ ($k > 1$) nyelvtanhoz létezik vele ekvivalens LR(1) nyelvtan.*



20.13. ábra. Az $[A \to \alpha.\beta, a]$ $LR(1)$-elem.

A fenti tétel rendkívül nagy jelentősége az, hogy $LR(k)$ $(k > 1)$ nyelvtanok és nyelvek helyett elegendő csak az $LR(1)$ nyelvtanokkal és nyelvekkel foglalkozni.

$LR(1)$ kanonikus halmazok

Definiáljuk az LR elemzések egyik központi fogalmát.

20.17. definíció. *Legyen az $\alpha\beta x$ $(\alpha, \beta \in (N \cup T)^*, x \in T^*)$ mondatforma nyele β. Ekkor az $\alpha\beta$ jelsorozat prefixeit az $\alpha\beta x$* ***járható prefixeinek*** *nevezzük.*

20.14. példa. Tekintsük a következő nyelvtant: $G' = (\{E, T, S'\}, \{i, +, (,)\}, P', S')$, ahol a helyettesítési szabályok a következők (a helyettesítési szabályokat sorszámmal láttuk el):

(0) $S' \to E$
(1) $E \to T$
(2) $E \to E + T$
(3) $T \to i$
(4) $T \to (E)$

A nyelvtan egy mondatformája $E + (i + i)$, ahol az első i a mondatforma nyele. Ennek a mondatformának a járható prefixei a következők: E, $E+$, $E + ($, $E + (i$.

A definíció szerint a járható prefixek a mondatforma nyele utáni szimbólumokat nem tartalmazhatják. Így, mivel az alulról-felfelé elemzésben a feladat a mondatforma nyelének a meghatározása, ez a feladat visszavezethető a mondatforma leghosszabb járható prefixének meghatározására.

Ha adott egy nyelvtan, akkor a nyelvtan helyettesítési szabályaiból a járható prefixek halmaza meghatározható. Ugyanakkor nyilvánvaló, hogy az egy nyelvtanhoz tartozó járható prefixek darabszáma nem feltétlenül véges.

A járható prefixek jelentősége az elemzésben a következő: a nyelvtan járható prefixeiből képezett halmazokhoz hozzárendelhetők egy determinisztikus véges automata állapotai, az állapotátmenetekhez pedig a nyelvtan szimbólumai úgy, hogy kiindulva az automata kezdőállapotából, egy állapothoz mindig egy járható prefix szimbólumain keresztül jutunk el. Ezt a tulajdonságot ismerve fogunk egy olyan módszert adni, amellyel az elemzést végző automata meghatározható.

20.18. definíció. *Ha a G' nyelvtan egy helyettesítési szabálya $A \to \alpha\beta$, akkor a nyelvtan* ***$LR(1)$-eleme***

$$[A \to \alpha.\beta, a], \quad (a \in T \cup \{\#\}),$$

ahol az $A \to \alpha.\beta$ az $LR(1)$-elem ***magja,*** *és a az $LR(1)$-elem* ***előreolvasási szimbóluma.***

Az előreolvasási szimbólumnak csak akkor van szerepe, ha az $LR(1)$-elem redukciót ír elő, azaz $[A \to \alpha., a]$ alakú. Ez azt jelenti, hogy redukciót majd csak abban az esetben szabad végrehajtani, ha az α-t, azaz a mondat nyelét az a szimbólum követi.

20.19. definíció. *Egy G' nyelvtan $[A \to \alpha.\beta, a]$ $LR(1)$-eleme* ***érvényes*** *a $\gamma\alpha$ járható prefixre nézve, ha*

$$S' \overset{*}{\Longrightarrow} \gamma A x \Longrightarrow \gamma\alpha\beta x \ (\gamma \in (N \cup T)^*,\ x \in T^*) ,$$

és az a az x első szimbóluma, vagy ha $x = \varepsilon$, akkor $a = \#$.

20.15. példa. Legyenek a $G' = (\{S', S, A\}, \{a, b\}, P', S')$ nyelvtan helyettesítési szabályai a következők:

(0) $S' \to S$
(1) $S \to AA$
(2) $A \to aA$
(3) $A \to b$

Ekkor $S' \overset{*}{\Longrightarrow} aaAab \Longrightarrow aaaAab$. Az aaa egy járható prefix, és az $[A \to a.A, a]$ érvényes elem erre a járható prefixre nézve. Hasonlóan, $S' \overset{*}{\Longrightarrow} AaA \Longrightarrow AaaA$. Az Aaa járható prefixre nézve az $[A \to a.A, \#]$ $LR(1)$-elem érvényes.

Az $LR(1)$ elemző felépítéséhez meg kell konstruálni az $LR(1)$-elemek kanonikus halmazait, és ehhez definiálni kell az $LR(1)$-elemhalmazokra a *closure* „lezárás” és a *read* „olvasás” függvényeket.

20.20. definíció. *Legyen a $\mathcal{H}$ halmaz egy nyelvtan egy $LR(1)$-elemhalmaza. Ekkor a* ***closure($\mathcal{H}$)*** *halmaz a következő $LR(1)$-elemeket tartalmazza:*

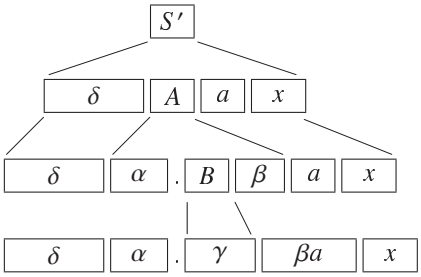
1. *a $\mathcal{H}$ halmaz minden eleme legyen eleme a closure($\mathcal{H}$) halmaznak is,*
2. *ha $[A \to \alpha.B\beta, a] \in closure(\mathcal{H})$ és $B \to \gamma$ a nyelvtan egy helyettesítési szabálya, akkor legyen $[B \to .\gamma, b] \in closure(\mathcal{H})$ minden $b \in Első(\beta a)$-ra,*
3. *a closure($\mathcal{H}$) halmazt a 2. pontban leírt művelettel addig kell bővíteni, ameddig az lehetséges.*

A definíció szerint, ha a $\delta\alpha$ járható prefixre nézve az $[A \to \alpha.B\beta, a]$ egy érvényes $LR(1)$-elem, akkor ugyanerre a prefixre a $[B \to .\gamma, b]$ is egy érvényes $LR(1)$-elem lesz, ahol $b \in Első(\beta a)$. (20.14. ábra). Látható az is, hogy a *closure* művelet a $\delta\alpha$ prefixre az összes érvényes LR(1)-elemet meghatározza.

Egy $\mathcal{H}$ $LR(1)$-elemhalmaz lezárását, azaz a *closure*($\mathcal{H}$) halmazt a következő algoritmussal határozhatjuk meg. A lezárás eredménye a $\mathcal{K}$-ba kerül.

ELEMHALMAZ-LEZÁR($\mathcal{H}$)

```
1  K ← ∅
2  for minden E ∈ H LR(1)-elemre
3      do K ← K ∪ ELEM-LEZÁR(E)
4  return K
```



20.14. ábra. A *closure*($[A \to \alpha.B\beta, a]$) függvény.

ELEM-LEZÁR(E)

```
 1  K_E ← {E}
 2  if E LR(1)-elem [A → α.Bβ, a] alakú
 3     then I ← ∅
 4          J ← K_E
 5          repeat
 6                for minden [C → γ.Dδ, b] ∈ J alakú LR(1)-elemre
 7                    do for minden D → η ∈ P szabályra
 8                           do for minden c ∈ ELSŐ(δb) szimbólumra
 9                                  do I ← I ∪ [D → .η, c]
10                J ← I
11                if I ≠ ∅
12                   then K_E ← K_E ∪ I
13                        I ← ∅
14          until J ≠ ∅
15  return K_E
```

Az ELEM-LEZÁR algoritmus egy E elem $\mathcal{K}_E$ lezárását adja meg. Ha az argumentumban a „pont” után egy terminális szimbólum van, akkor az eredmény halmazában csak ez az egy elem lesz (1. sor). Ha ez a szimbólum egy B nemterminális szimbólum, akkor a B bal oldalú szabályok mindegyikéből tudunk egy új LR(1)-elemet készíteni, ez található a 9. sorban. Mivel az elemek vizsgálatát minden új elemre is el kell végezni, az 5–14. sorok egy **repeat** ciklust tartalmaznak. Ezeket a lépéseket addig kell végezni, amíg új elemeket kapunk (14. sor). A J halmaz tartalmazza a megvizsgálandó elemeket, és I az új elemeket, a $J \leftarrow I$ művelet a 10. sorban látható.

20.21. definíció. *Legyen a $\mathcal{H}$ halmaz egy nyelvtan egy LR(1)-elemhalmaza. Ekkor a* ***read($\mathcal{H}$, X)*** *($X \in (N \cup T)$) halmaz a következő LR(1)-elemeket tartalmazza:*

1. *ha $[A \to \alpha.X\beta, a] \in \mathcal{H}$, akkor a closure($[A \to \alpha X.\beta, a]$) minden eleme legyen a read($\mathcal{H}, X$) halmaz eleme,*
2. *a read($\mathcal{H}, X$) halmazt az 1. művelettel addig kell bővíteni, ameddig az lehetséges.*

Szemléletesen, a $read(\mathcal{H}, X)$ függvény a $\mathcal{H}$ halmaz elemeiben az X szimbólumot olvassa, a „pont” jel az eredmény halmaz elemeiben már az X jobb oldalán van. Ha $\mathcal{H}$ a γ járható prefixekre nézve érvényes $LR(1)$-elemeket tartalmazza, akkor a $read(\mathcal{H}, X)$ a γX-re nézve érvényes $LR(1)$-elemek halmaza lesz.

A *read* műveletet az ELEMHALMAZ-OLVAS algoritmus valósítja meg, az eredményt a $\mathcal{K}$-ban kapjuk meg.

ELEMHALMAZ-OLVAS($\mathcal{H}, Y$)

1 $\mathcal{K} \leftarrow \emptyset$
2 **for** minden $E \in H$
3 **do** $\mathcal{K} \leftarrow \mathcal{K} \cup$ ELEM-OLVAS(E, Y)
4 **return** $\mathcal{K}$

ELEM-OLVAS(E, Y)

1 **if** $E = [A \to \alpha.X\beta, a]$ és $X = Y$
2 **then** $\mathcal{K}_{E,Y} \leftarrow$ ELEM-LEZÁR($[A \to \alpha X.\beta, a]$)
3 **else** $\mathcal{K}_{E,Y} \leftarrow \emptyset$
4 **return** $\mathcal{K}_{E,Y}$

A második algoritmus 2. sorában látható, hogy az összes olyan LR(1)-elemet meghatározzuk, ami az olvasás utáni állapotot írja le.

Az $LR(1)$-elemek felsorolásának rövidebb leírása érdekében vezessük be a következő jelölést:

$$[A \to \alpha.X\beta, a/b]$$

jelentse az

$$[A \to \alpha.X\beta, a] \text{ és } [A \to \alpha.X\beta, b]$$

$LR(1)$-elemeket.

20.16. példa. A 20.15. példában szereplő nyelvtan egy $LR(1)$-eleme $[S' \to .S, \#]$. Erre

$$closure([S' \to .S, \#]) = \{[S' \to .S, \#], [S \to .AA, \#], [A \to .aA, a/b], [A \to .b, a/b]\} .$$

Az $LR(1)$-*elemek kanonikus halmazait,* vagy röviden az $LR(1)$-*kanonikus halmazokat* a következő módszerrel határozzuk meg:

20.22. definíció. *A $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_m$ $LR(1)$-elemek* ***kanonikus halmazai*** *a következők:*

- *Legyen $\mathcal{H}_0 = closure([S' \to .S, \#])$,*
- *Ezután képezzük egy X szimbólumra a $read(\mathcal{H}_0, X)$ halmazt. Ha az így kapott halmaz nem üres, és nem egyezik meg a $\mathcal{H}_0$ kanonikus halmazzal, akkor legyen ez a következő kanonikus halmaz, azaz $\mathcal{H}_1$.*

  Ismételjük meg ezt a műveletet az összes lehetséges X terminális és nemterminális szimbólumra úgy, hogy ha olyan nem üres halmazt kapunk, amelyik nem egyezik meg egyik

korábbi kanonikus halmazzal sem, akkor ez a halmaz legyen egy új kanonikus halmaz, és indexe legyen 1-gyel nagyobb, mint az eddigi maximális index.

- *Ezután ismételjük meg ezt a műveletet a már korábban előállított összes kanonikus halmazra és a nyelvtan minden szimbólumára, egészen addig, amíg csak új kanonikus halmazt kapunk.*

Az így létrehozott

$$\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_m$$

halmazokat nevezzük a G nyelvtan $LR(1)$-kanonikus halmazainak.

Mivel egy nyelvtanra az $LR(1)$-elemek darabszáma véges, az $LR(1)$-kanonikus halmazok létrehozása biztosan véges lépésben befejeződik.

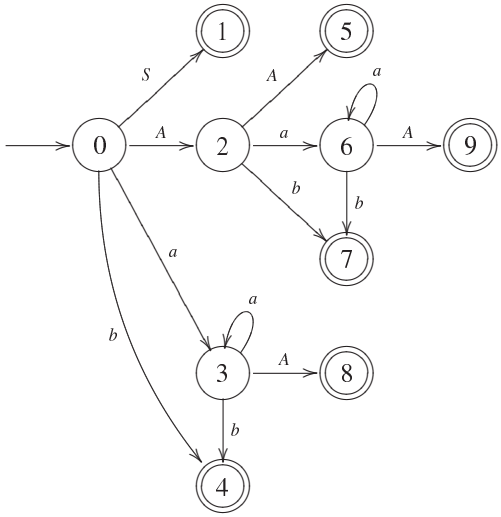
A G nyelvtan kanonikus halmazait a következő algoritmus állítja elő:

KANONIKUS-HALMAZOKAT-KÉSZÍT(G)

```
 1  i ← 0
 2  H_i ← ELEM-LEZÁR([S' → .S, #])
 3  I ← {H_i}, K ← {H_i}
 4  repeat
 5        L ← K
 6        for minden M ∈ I-re
 7           do I ← I \ M
 8              for minden X ∈ T ∪ N-re
 9                 do J ← ELEMHALMAZ-LEZÁR(ELEMHALMAZ-OLVAS(M, X))
10                    if J ≠ ∅ és J ∉ K
11                       then i ← i + 1
12                            H_i ← J
13                            K ← K ∪ {H_i}
14                            I ← I ∪ {H_i}
15  until K = L
16  return K
```

Az algoritmus a K-ban adja a kanonikus halmazokat, a 2. sorban látható, hogy az első kanonikus halmaz a $\mathcal{H}_0$ lesz. További halmazokat a már meglévő kanonikus halmazokból az ELEMHALMAZ-LEZÁR(ELEMHALMAZ-OLVAS) függvénnyel képezünk a 9. sorban. A 10. sor programja azt vizsgálja, hogy ez az új halmaz vajon különbözik-e az eddigiektől, és ha igen, akkor a 11–12. sorban ez a halmaz egy új kanonikus halmaz lesz. A 6–14. sorok **for** ciklusa azt biztosítja, hogy ezeket a műveleteket a már meglévő minden kanonikus halmazra elvégezzük. A 3–14. sorokban lévő **repeat** ciklus szerint a kanonikus halmazok generálását addig végezzük, amíg új kanonikus halmazokat kapunk.

20.17. példa. A 20.15. példában szereplő nyelvtan $LR(1)$-elemeinek kanonikus halmazai a következők:



20.15. ábra. A 20.15. példa járható prefixeit felismerő automata.

$$
\begin{array}{lcll}
\mathcal{H}_0 & & = closure([S' \to .S]) & = \{[S' \to .S, \#], [S \to .AA, \#], \\
 & & & [A \to .aA, a/b], [A \to .b, a/b]\} \\
\mathcal{H}_1 & = read(\mathcal{H}_0, S) & = closure([S' \to S., \#]) & = \{[S' \to S., \#]\} \\
\mathcal{H}_2 & = read(\mathcal{H}_0, A) & = closure([S' \to A.A, \#]) & = \{[S \to A.A, \#], [A \to .aA, \#], \\
 & & & [A \to .b, \#]\} \\
\mathcal{H}_3 & = read(\mathcal{H}_0, a) & = closure([A \to a.A, a/b]) & = \{[A \to a.A, a/b], [A \to .aA, a/b], \\
 & & & [A \to .b, a/b]\} \\
\mathcal{H}_4 & = read(\mathcal{H}_0, b) & = closure([A \to b., a/b]) & = \{[A \to b., a/b]\} \\
\mathcal{H}_5 & = read(\mathcal{H}_2, A) & = closure([S \to AA., \#]) & = \{[S \to AA., \#]\} \\
\mathcal{H}_6 & = read(\mathcal{H}_2, a) & = closure([A \to a.A, \#]) & = \{[A \to a.A, \#], [A \to .aA, \#], \\
 & & & [A \to .b, \#]\} \\
\mathcal{H}_7 & = read(\mathcal{H}_2, b) & = closure([A \to b., \#]) & = \{[A \to b., \#]\} \\
\mathcal{H}_8 & = read(\mathcal{H}_3, A) & = closure([A \to aA., a/b]) & = \{[A \to aA., a/b]\} \\
 & read(\mathcal{H}_3, a) & = \mathcal{H}_3 & \\
 & read(\mathcal{H}_3, b) & = \mathcal{H}_4 & \\
\mathcal{H}_9 & = read(\mathcal{H}_6, A) & = closure([A \to aA., \#]) & = \{[A \to aA., \#]\} \\
 & read(\mathcal{H}_6, a) & = \mathcal{H}_6 & \\
 & read(\mathcal{H}_6, b) & = \mathcal{H}_7 &
\end{array}
$$

Az elemző automatája a 20.15. ábrán látható.

Az *LR*(1) elemző

Ha egy G' kiegészített nyelvtanhoz meghatároztuk az $LR(1)$-elemek

$$\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_m$$

kanonikus halmazait, akkor egy automata k állapotához rendeljük hozzá a $\mathcal{H}_k$ halmazt. Az automata állapotai és az $LR(1)$-elemek kanonikus halmazai közötti kapcsolatot a következő, az ***LR(1)-elemzés nagy tételének*** is nevezett állítás mondja ki:

20.23. tétel. *Egy γ járható prefixre érvényes $LR(1)$-elemek halmaza az a $\mathcal{H}_k$ kanonikus elemhalmaz, amelyik az elemző véges determinisztikus automatájának ahhoz a k állapotához tartozik, amelyikbe az automata a kezdőállapotból a γ hatására kerül.*

A tétel azt mondja ki, hogy a járható perfixeket felismerő automata felépíthető a kanonikus halmazok ismeretében. Állítsuk elő tehát az $LR(1)$-elemek kanonikus halmazaiból az $LR(1)$ elemzőt.

A járható prefixeket felismerő determinisztikus véges automata leírható egy táblázattal, ezt $LR(1)$ *elemző táblázatnak* nevezzük. A táblázat sorait az automata állapotaihoz rendeljük hozzá.

Az elemző táblázat két részből áll. Az első neve az *action* táblázat. Mivel az elemezendő szöveg szimbóluma határozza meg az elvégzendő műveletet, az *action* táblázatot oszlopokra bontjuk, és az oszlopokhoz a terminális szimbólumokat rendeljük. Az *action* táblázat azt tartalmazza, hogy az adott állapotban, ha az oszlophoz tartozó terminális szimbólum a bemenő jel, léptetést vagy redukciót kell-e végrehajtani. A léptetés műveletét jelöljük sj-vel, ahol s a léptetést, j a léptetés utáni állapotot jelenti. A redukció jele legyen ri, ahol i az alkalmazott helyettesítési szabály sorszáma. Mivel a nulladik szabály szerinti redukció azt jelenti, hogy elemzés befejeződött és az elemzett szöveg szintaktikusan helyes, jelöljük ezt a táblázatban az *elfogad* szóval.

A második rész a *goto* táblázat. Ebbe az az információ kerül, hogy a nemterminális szimbólumok hatására az automata egy adott állapotból melyik állapotba megy át. (A terminális szimbólumok állapot-átmeneteit az *action* táblázat sj bejegyzései tartalmazzák.)

Az automata állapotainak halmaza legyen a $\{0, 1, \ldots, m\}$ halmaz, az elemző táblázatok i-edik sorát a $\mathcal{H}_i$ $LR(1)$-elemeiből töltjük ki.

Az *action* táblázat i-edik sora:

- ha $[A \to \alpha.a\beta, b] \in \mathcal{H}_i$ és $read(\mathcal{H}_i, a) = \mathcal{H}_j$, akkor legyen $action[i, a] = sj$,
- ha $[A \to \alpha., a] \in \mathcal{H}_i$ és $A \neq S'$, akkor legyen $action[i, a] = rl$, ahol az $A \to \alpha$ a nyelvtan l-edik szabálya,
- ha $[S' \to S., \#] \in \mathcal{H}_i$, akkor legyen $action[i, \#] = elfogad$.

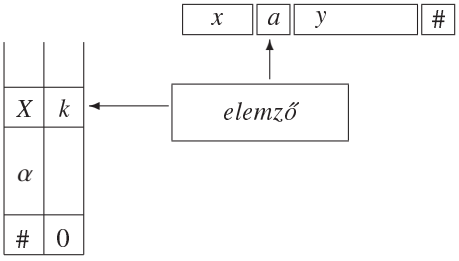
A *goto* táblázat kitöltésének módszere:

- ha $read(\mathcal{H}_i, A) = \mathcal{H}_j$, akkor legyen $goto[i, A] = j$.
- Mindkét táblázatban az üresen maradt helyeket a *hiba* szöveggel töltsük ki.

Az $LR(1)$-elemek kanonikus halmazaiból létrehozott *action* és *goto* táblázatokat $LR(1)$ vagy *kanonikus elemző táblázatoknak* nevezzük.

20.24. tétel. *A G' kiegészített nyelvtan akkor és csak akkor $LR(1)$ nyelvtan, ha a nyelvtanhoz készített kanonikus elemző táblázatok kitöltése konfliktusmentes.*

A táblázat kitöltését a következő algoritmussal végezhetjük:



20.16. ábra. Az *LR*(1) elemző szerkezete.

LR(1)-TÁBLÁZATOT-KITÖLT(G)

```
 1  for minden H_i LR(1) kanonikus halmazra
 2      do for minden LR(1)-elemre
 3             if [A → α.aβ, b] ∈ H_i és read(H_i, a) = H_j
 4                then action[i, a] = sj
 5             if [A → α., a] ∈ H_i és A ≠ S' és A → α az l-edik szabály
 6                then action[i, a] = rl
 7             if [S' → S., #] ∈ H_i
 8                then action[i, #] = elfogad
 9             if read(H_i, A) = H_j
10                then goto[i, A] = j
11      for minden a ∈ (T ∪ {#})
12          do if action[i, a] = „üres”
13                then action[i, a] ← hiba
14      for minden X ∈ N
15          do if goto[i, X] = „üres”
16             then goto[i, X] ← hiba
17  return action, goto
```

A táblázatokat soronként töltjük ki, a 2–6. sorokban az *action* táblázatot, a 9–10. sorokban a *goto* táblázatot. Az algoritmus 11–13. soraiban a táblázatok sorainak üresen maradt helyeire a szintaktikus hibát jelző *hiba* szöveget írjuk.

Az *LR*(1) elemző működése a következőképpen adható meg (20.16. ábra).

Az elemző verme egy „dupla verem”, azaz egy *push* vagy *pop* művelettel két információt írunk vagy olvasunk. A verem szimbólumpárokat tartalmaz, a párok első elemében egy terminális vagy nemterminális szimbólumot tárolunk, a második elemben pedig az automata állapotának sorszámát. A verem kezdeti tartalma legyen #0.

Az *elemző állapotát* egy kettőssel írjuk le, a kettős első eleme legyen a verem tartalma, a második elem pedig a bemenő szimbólumsorozat még nem elemzett része. Az elemző *kezdőállapota* tehát (#0, z#), ahol z az elemezendő szimbólumsorozat. Az elemzés sikeresen befejeződik, azaz az elemző a *végállapotba* kerül, ha a verem tartalma ismét #0, és az elemzéssel az elemezendő szimbólumsorozat végére értünk.

Tegyük fel, hogy az elemző pillanatnyi állapota a $(\#0 \ldots Y_k i_k, ay\#)$ kettőssel írható le. Ekkor az elemző következő lépését az $action[i_k, a]$ adat határozza meg.

Az állapotátmenetek a következők:

- Ha $action[i_k, a] = sl$, azaz az automata egy léptetést hajt végre, akkor a bemenet soron következő a szimbóluma és az új állapot i_l sorszáma kerüljön a verembe, azaz

$$(\#0 \ldots Y_k i_k, ay\#) \to (\#0 \ldots Y_k i_k a i_l, y\#) \; .$$

- Ha $action[i_k, a] = rl$, akkor az l-edik szabály, az $A \to \alpha$ szabály szerint kell redukálni. Először töröljük a verem $|\alpha|$ darab sorát, azaz $2|\alpha|$ elemét. Ezután határozzuk meg a $goto$ táblázatból, hogy az automata a törlés után a verem tetejére kerülő állapotból az A hatására melyik állapotba kerül, majd az A szimbólumot és a meghatározott állapotsorszámot írjuk be a verembe.

$$(\#0 \ldots Y_{k-r} i_{k-r} Y_{k-r+1} i_{k-r+1} \ldots Y_k i_k, y\#) \to (\#0 \ldots Y_{k-r} i_{k-r} A i_l, y\#) \; ,$$

  ahol $|\alpha| = r$, és $goto[i_{k-r}, A] = i_l$.
- Ha $action[i_k, a] = elfogad$, akkor az elemzés a veremből való törlés után befejeződik, az elemző az elemzett szöveget elfogadja.
- Ha $action[i_k, a] = hiba$, akkor az elemzés befejeződik, az elemző az elemzett szövegben az a szimbólumnál egy *szintaktikus hibát* detektált.

Az $LR(1)$ elemzőt gyakran *kanonikus* $LR(1)$ *elemzőnek* is nevezik.

Ha T-vel jelöljük az *action* és *goto* táblákat, az elemző működésére a következő algoritmust adhatjuk meg.

LR(1)-ELEMEZ($xay\#, T$)

```
 1  s ← (#0, xay#), s' ← elemez
 2  repeat
 3        s = (#0 ... Y_{k-r} i_{k-r} Y_{k-r+1} i_{k-r+1} ... Y_k i_k, ay#)
 4        if action[i_k, a] = sl
 5           then s ← (#0 ... Y_k i_k a i_l, y#)
 6           else if action[i_k, a] = rl és A → α az l-edik szabály és
 7                    |α| = r és goto[i_{k-r}, A] = i_l
 8                    then s ← (#0 ... Y_{k-r} i_{k-r} A i_l, ay#)
 9                    else if action[i_k, a] = elfogad
10                             then s' ← O.K.
11                             else  s' ← HIBA
12  until s' = O.K. vagy s' = HIBA
13  return s', s
```

Az algoritmus bemenő paramétere az xay elemezendő szöveg és a T elemző táblázat. Az s' változó az elemző működését jelzi, működés közben az s' értéke *elemez*, az elemzés befejezésekor *O.K.* vagy *HIBA*. A 3. sorban az elemző állapotát írjuk fel részletesen, erre majd a 6–8. sorokban levő művelet esetén lesz szükség. Az elemző az automatának a verem tetején levő x_k állapota és az a aktuális szimbólum alapján a *action* táblázatból meghatározza az elvégzendő műveletet. A 4–5. sorban a léptetés műveletét hajtjuk végre,

a 6–8. sorokban a redukálás művelete található. Az algoritmus a 9–11. sorban az elemzés befejezését jelzi, ha az elemezendő szöveg végére ért és a verem tetején a 0 állapot van, akkor az elemzett szöveg helyes, egyebként az elemző egy szintaktikus hibát fedezett fel. Az algoritmus végeredménye ennek megfelelően az *O.K.* vagy *HIBA* jelzés, és kimenetként mindkét esetben megjelenik az elemző állapota is. Szintaktikus hiba esetén az elemző állapot második elemének első szimbóluma a hiba helyét adja meg.

20.18. példa. A 20.15. példában megadott nyelvtan *LR*(1) elemzőjének *action* és *goto* táblázatai a következők, az üres helyek most is a *hibát* jelentik.

állapot	*action*			*goto*	
	a	*b*	#	*S*	*A*
0	*s*3	*s*4		1	2
1			*elfogad*		
2	*s*6	*s*7			5
3	*s*3	*s*4			8
4	*r*3	*r*3			
5			*r*1		
6	*s*6	*s*7			9
7			*r*3		
8	*r*2	*r*2			
9			*r*2		

20.19. példa. Elemezzük az előző példában megadott táblázat felhasználásával az *abb*# szöveget.

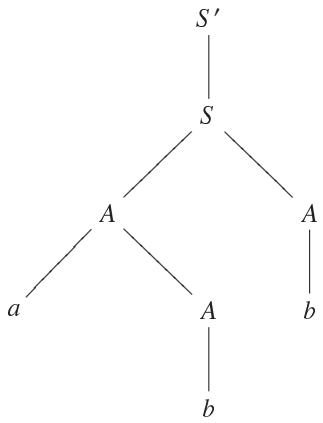
				szabály
(#0,	*aab*#)	$\xrightarrow{s3}$	(#0*a*3, *bb*#)	
		$\xrightarrow{s4}$	(#0*a*3*b*4, *b*#)	
		$\xrightarrow{r3}$	(#0*a*3*A*8, *b*#)	$A \to b$
		$\xrightarrow{r2}$	(#0*A*2, *b*#)	$A \to aA$
		$\xrightarrow{s7}$	(#0*A*2*b*7, #)	
		$\xrightarrow{r3}$	(#0*A*2*A*5, #)	$A \to b$
		$\xrightarrow{r1}$	(#0*S* 1, #)	$S \to AA$
		$\xrightarrow{elfogad}$	*O.K.*	

Az elemzett mondat szintaxisfája a 20.17. ábrán látható.

Az *LALR*(1) elemző

Mivel az elemző program állapotszámától nemcsak az elemző mérete, hanem a sebessége is függ, most célul az állapotok számának csökkentését tűzzük ki, és arra törekszünk, hogy ezzel az elemezhető nyelvek halmaza az *LR*(1) nyelvekhez viszonyítva lényegesen ne csökkenjen.

Vegyük azt észre, hogy az *LR*(1)-elemek kanonikus halmazai között vannak olyan halmazpárok, hogy az egyik halmazban levő minden *LR*(1)-elemnek van egy megfelelője a



20.17. ábra. Az *aab* mondat szintaxisfája.

másik halmazban, úgy, hogy ezeknek az elemeknek a magjuk azonos, és legfeljebb csak az előreolvasási szimbólumokban különböznek. Egyesítsük ezeket a halmazpárokat.

Ha a $\mathcal{H}_i$ és a $\mathcal{H}_j$ halmazok egyesíthetők, akkor legyen $\mathcal{K}_{[i,j]} = \mathcal{H}_i \cup \mathcal{H}_j$.

Végezzük el az *LR*(1)-kanonikus halmazok összes lehetséges egyesítését, az indexek átsorszámozása után az így kapott $\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_n$ halmazokat nevezzük *egyesített LR*(1) *kanonikus halmazoknak,* vagy *LALR*(1)*-kanonikus halmazoknak*.

Ezekből az egyesített halmazokból létrehozható elemzőt fogjuk majd *LALR*(1) elemzőnek nevezni.

20.20. példa. A 20.17. példában szereplő kanonikus halmazok közül a következőket lehet egyesíteni:
$\mathcal{H}_3$ és $\mathcal{H}_6$,
$\mathcal{H}_4$ és $\mathcal{H}_7$,
$\mathcal{H}_8$ és $\mathcal{H}_9$.
A 20.15. ábrán is látható, hogy az összevonható halmazok az automatában azonos, vagy legalábbis hasonló részstruktúrát alkotnak.

A *read* függvény az egyesített halmazokra nem okozhat problémát, azaz ha

$$\mathcal{K} = \mathcal{H}_1 \cup \mathcal{H}_2 \cup \ldots \cup \mathcal{H}_k \ ,$$

$$read(\mathcal{H}_1, X) = \mathcal{H}_1', read(\mathcal{H}_2, X) = \mathcal{H}_2', \ldots, read(\mathcal{H}_k, X) = \mathcal{H}_k' \ ,$$

és

$$\mathcal{K}' = \mathcal{H}_1' \cup \mathcal{H}_2' \cup \ldots \cup \mathcal{H}_k' \ ,$$

akkor

$$read(\mathcal{K}, X) = \mathcal{K}' \ .$$

Ezt a következőképpen lehet belátni. A *read* függvény definíciója szerint a $read(\mathcal{H}, X)$ csak a $\mathcal{H}$ *LR*(1)-elemeinek magjaitól függ, és nem függ az előreolvasási szimbólumoktól. Így, mivel a $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k$ halmazokban az *LR*(1)-elemek magjai azonos halmazokat alkotnak, a

$$read(\mathcal{H}_1, X), read(\mathcal{H}_2, X), \ldots, read(\mathcal{H}_k, X)$$

LR(1)-elemeinek magjaiból alkotott halmazok is azonosak, tehát ezek a halmazok is egyesíthetők egy $\mathcal{K}'$ halmazba, és így valóban $read(\mathcal{K}, X) = \mathcal{K}'$.

Az *LR*(1)-elemek kanonikus halmazainak egyesítése után azonban az egyesített halmazon belül maguk az *LR*(1)-elemek okozhatnak problémát. Tegyük fel, hogy

$\mathcal{K}_{[i,j]} = \mathcal{H}_i \cup \mathcal{H}_j$.

- *Léptetés-léptetés konfliktus* az összevonás után nem léphet fel. Ha

  $$[A \to \alpha.a\beta, b] \in \mathcal{H}_i \ ,$$

  és

  $$[B \to \gamma.a\delta, c] \in \mathcal{H}_j \ ,$$

  akkor az összevonás után az a szimbólumra továbbra is egy léptetést írunk elő, és a fentiekben láttuk, hogy a *read* függvény sem okoz problémát, azaz a $read(\mathcal{K}_{[i,j]}, a)$ éppen a $read(\mathcal{H}_i, a) \cup read(\mathcal{H}_j, a)$-val egyenlő.
- Ha $\mathcal{H}_i$ kanonikus halmazban egy

  $$[A \to \alpha.a\beta, b] \ ,$$

  a $\mathcal{H}_j$-ben egy

  $$[B \to \gamma., a]$$

  elem szerepelne, akkor az egyesítés után az a szimbólum miatt egy inadekvát állapotot kapnánk, *léptetés-redukálás konfliktus* lépne fel. Ez az eset azonban sohasem állhat fenn, mivel ekkor mindkét elemnek szerepelnie kell mind a $\mathcal{H}_i$, mind a $\mathcal{H}_j$ halmazban, legfeljebb csak az előreolvasási szimbólumokban különbözhetnek, hiszen ezért tudtuk egyesíteni őket. Tehát a $\mathcal{H}_j$ halmazban is kell lennie egy $[A \to \alpha.a\beta, c]$ elemnek. Ekkor pedig a 20.24. tétel alapján a nyelvtan nem lenne *LR*(1) nyelvtan; már a $\mathcal{H}_j$ halmazból léptetés-redukálás konfliktus következne, az a szimbólumot előreolvasva nem lehetne eldönteni, hogy az elemzésben milyen műveletet kell alkalmazni.
- Az összevonás után azonban *redukálás-redukálás konfliktus* előfordulhat, az *LR*(1) nyelvtan tulajdonságai ezt nem zárják ki. A következő példában egy ilyen esetet mutatunk be.

20.21. példa. Tekintsük a $G' = (\{S', S, A, B\}, \{a, b, c, d, e\}, P', S')$ nyelvtant, ahol a helyettesítési szabályok:

$S' \to S$
$S \to aAd \mid bBd \mid aBe \mid bAe$
$A \to c$
$B \to c$

A nyelvtan egy *LR*(1) nyelvtan. Az ac járható prefixre az

$$\{[A \to c., d], [B \to c., e]\} \ ,$$

valamint a bc járható prefixre az

$$\{[A \to c., e], [B \to c., d]\}$$

LR(1)-elemek egy-egy kanonikus halmazt alkotnak.

A két halmaz egyesítése után redukálás-redukálás konfliktus lép fel. Ha a bemenő szimbólum d vagy e, a c mondatnyél azonosítható, de nem dönthető el, hogy az $A \to c$ és a $B \to c$ szabály szerinti redukciók közül melyiket kell végrehajtani.

Most az *LALR*(1) elemző táblázatainak kitöltési szabályait adjuk meg. Miután meghatároztuk az *LR*(1)-elemek

$$\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$$

kanonikus halmazait, egyesítsük egy halmazba azokat a kanonikus halmazokat, amelyekben az *LR*(1)-elemek magjaiból alkotott halmazok azonosak. Legyenek ezek a halmazok

$$\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n \quad (n \leq m) .$$

Az *action* és a *goto* táblázatok méretének a meghatározására és a táblázatok kitöltésére a $\mathcal{K}_i$ $(1 \leq i \leq n)$ halmazokat kell használni, a táblázatok kitöltésének módszere teljesen megegyezik az *LR*(1) elemzőnél leírtakkal. A táblázatokat *LALR*(1) *elemző táblázatoknak* nevezzük.

20.25. definíció. *Ha a G' kiegészített nyelvtanra a LALR(1) elemző táblázatok kitöltése konfliktusmentes, akkor a nyelvtant* ***LALR(1) nyelvtannak*** *nevezzük.*

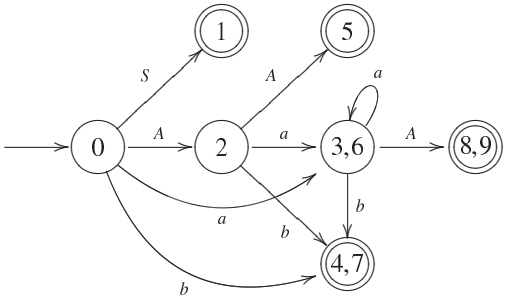
Az *LALR*(1) elemző működése az *LR*(1) elemző működésével egyezik meg.

20.22. példa. A $\mathcal{H}_i$ és a $\mathcal{H}_j$ kanonikus halmazok egyesítéséből származó $\mathcal{K}_{[i,j]}$ halmazhoz tartozó állapotot jelöljük $[i, j]$ -vel.

A 20.15. példában szereplő nyelvtan *LR*(1)-elemeinek kanonikus halmazait a 20.17. példában adtuk meg, és a 20.20. példában láttuk az egyesíthető halmazpárokat. Így a nyelvtanhoz a következő *LALR*(1) elemző táblázatokat lehet elkészíteni.

állapot	*action*			*goto*	
	a	*b*	#	*S*	*A*
0	*s* [3, 6]	*s* [4, 7]		1	2
1			*accept*		
2	*s* [3, 6]	*s* [4, 7]			5
[3, 6]	*s* [3, 6]	*s* [4, 7]			[8, 9]
[4, 7]	*r*3	*r*3	*r*3		
5			*r*1		
[8, 9]	*r*2	*r*2	*r*2		

Az *LALR*(1)-táblázatok kitöltése konfliktusmentes, a nyelvtan tehát egy *LALR*(1) nyelvtan. Az elemző automatája a 20.18. ábrán látható.



20.18. ábra. A 20.22. példa járható prefixeit felismerő automatája.

20.23. példa. Elemezzük az előző példában megadott táblázat felhasználásával az *abb*# szöveget.

					szabály
(#0,	*aab*#)	$\xrightarrow{s[3,6]}$	(#0*a* [3, 6] ,	*bb*#)	
		$\xrightarrow{s[4,7]}$	(#0*a* [3, 6] *b* [4, 7] ,	*b*#)	
		$\xrightarrow{r3}$	(#0*a* [3, 6] *A*[8, 9],	*b*#)	$A \to b$
		$\xrightarrow{r2}$	(#0*A*2,	*b*#)	$A \to aA$
		$\xrightarrow{s[4,7]}$	(#0*A*2*b* [4, 7] ,	#)	
		$\xrightarrow{r3}$	(#0*A*2*A*5,	#)	$A \to b$
		$\xrightarrow{r1}$	(#0*S* 1,	#)	$S \to AA$
		$\xrightarrow{elfogad}$	*O.K.*		

Az elemzett mondat szintaxisfája a 20.17. ábrán látható.

Az *LALR*(1) nyelvtanok egyben *LR*(1) nyelvtanok is, mint az a fenti példából is látható, de ez fordítva nem áll fenn. A 20.21. példában éppen egy olyan nyelvtan szerepelt, amelyik *LR*(1), de nem *LALR*(1) nyelvtan.

A programnyelvek generálhatók *LALR*(1) nyelvtannal, a programnyelvek fordítóprogramjaiban leggyakrabban alkalmazott elemzési módszer az *LALR*(1) elemzés. Az *LALR*(1) elemző előnye az *LR*(1) elemzővel szemben az, hogy táblázatainak mérete lényegesen kisebb.

Például, a Pascal nyelvre az *LALR*(1) -táblázatok néhányszor száz sort tartalmaznak, míg az *LR*(1) elemző táblázatai több ezer sorból állnak.

Gyakorlatok

20.3-1. Határozzuk meg, hogy a következő nyelvtanok közül melyek *LL*(1) nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S \to ABc$
   $A \to a \mid \varepsilon$
   $B \to b \mid \varepsilon$

2. $S \to Ab$
   $A \to a \mid B \mid \varepsilon$
   $B \to b \mid \varepsilon$

3. $S \to ABBA$
   $A \to a \mid \varepsilon$
   $B \to b \mid \varepsilon$

4. $S \to aSe \mid A$
   $A \to bAe \mid B$
   $B \to cBe \mid d$

20.3-2. Bizonyítsuk be, hogy a következő nyelvtanok $LL(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S \to Bb \mid Cd$
   $B \to aB \mid \varepsilon$
   $C \to cC \mid \varepsilon$

2. $S \to aSA \mid \varepsilon$
   $A \to c \mid bS$

3. $S \to AB$
   $A \to a \mid \varepsilon$
   $B \to b \mid \varepsilon$

20.3-3. Bizonyítsuk be, hogy a következő nyelvtanok nem $LL(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S \to aAa \mid Cd$
   $A \to abS \mid c$

2. $S \to aAaa \mid bAba$
   $A \to b \mid \varepsilon$

3. $S \to abA \mid \varepsilon$
   $A \to Saa \mid b$

20.3-4. Mutassuk meg, hogy az $LL(0)$ nyelvtannak csak egy mondata van.

20.3-5. Bizonyítsuk be, hogy a következő nyelvtanok $LR(0)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S' \to S$
   $S \to aSa \mid aSb \mid c$

2. $S' \to S$
   $S \to aAc$
   $A \to Abb \mid b$

20.3-6. Bizonyítsuk be, hogy a következő nyelvtanok $LR(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S' \to S$
   $S \to aSS \mid b$

2. $S' \to S$
   $S \to SSa \mid b$

20.3-7. Bizonyítsuk be, hogy a következő nyelvtanok nem $LR(k)$ nyelvtanok egyetlen k-ra sem. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S' \to S$
   $S \to aSa \mid bSb \mid a \mid b$

2. $S' \to S$
   $S \to aSa \mid bSa \mid ab \mid ba$

20.3-8. Bizonyítsuk be, hogy a következő nyelvtanok $LR(1)$, de nem $LALR(1)$ nyelvtanok. A nyelvtanoknak csak a helyettesítési szabályait adjuk meg.

1. $S' \to S$
   $S \to Aa \mid bAc \mid Bc \mid bBa$
   $A \to d$
   $B \to d$

2. $S' \to S$
   $S \to aAcA \mid A \mid B$
   $A \to b \mid Ce$
   $B \to dD$
   $C \to b$
   $D \to CcS \mid CcD$

20.3-9. A fenti példákban szereplő $LL(1)$ nyelvtanokhoz készítsük el az elemző táblázatokat.
20.3-10. A fenti példákban szereplő $LL(1)$ nyelvtanokhoz írjuk meg a rekurzív leszállás elemző programjait.
20.3-11. A fenti példákban szereplő $LR(1)$ nyelvtanokhoz készítsük el a kanonikus halmazokat és az elemző táblázatokat.
20.3-12. A fenti példákban szereplő $LALR(1)$ nyelvtanokhoz készítsük el az összevont kanonikus halmazokat és az elemző táblázatokat.

Feladatok

20-1. Programszöveg lexikális elemzése
A 20.2. alfejezetben szereplő LEX-ELEMEZ algoritmus csak *egy* reguláris kifejezéssel, vagy a megfelelő véges determinisztikus automatával leírható szövegek elemzését adta meg, azaz csak egy szimbólumot ismert fel. Készítsünk egy olyan automatát, ami egy teljes programnyelv lexikális elemzését elvégzi, és adjuk meg a teljes elemzés LEX-ELEMEZ-NYELV algoritmusát. Az algoritmus egyik bemenete legyen egy programszöveg, kimenete pedig a szimbólumsorozat. Nyilvánvaló, hogy ha az automata egy végállapotba került, azaz felismert egy szimbólumot, akkor ezután a működését a kezdőállapottal kell folytatnia, hogy a programszöveg következő szimbólumát meghatározza. Az algoritmus működésének csak akkor kell befejeződnie, ha az elemzés a programszöveg végére ért, vagy ha egy lexikális hibát talált.

20-2. Szimbólumsorozat a szimbólumok adataival
Módosítsuk az előző feladat algoritmusát úgy, hogy a szimbólumsorozat tartalmazza a felismert szimbólum jellemző adatait is, például egy azonosító szimbólum mellé adjuk meg a szimbólumot alkotó karaktersorozatot, vagy egy szám mellé a szám típusát és értékét is. A szimbólumok kódja mellé, az egységes kezelés érdekében, célszerű nem az adatokat, hanem az adatokra mutató pointereket írni.

20-3. LALR(1) elemző LR(0)-kanonikus halmazokból
Ha az $LR(1)$-elemekből elhagyjuk az előreolvasási szimbólumokat, akkor az ***LR(0)-elemeket*** kapjuk meg. Az $LR(0)$-elemekre is definiálhatjuk a *closure* és *read* függvényeket, úgy, hogy az előreolvasási szimbólumokat nem vesszük figyelembe. Az $LR(1)$ kanonikus halmazokhoz hasonló módszerrel meghatározhatjuk az

$$\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n$$

LR(0) kanonikus halmazokat is. Megfigyelhetjük, hogy $LALR(1)$ nyelvtanok esetén az egyesítés után kapott $LALR(1)$ kanonikus halmazok darabszáma megegyezik az $LR(0)$-kanonikus halmazok darabszámával, hiszen az egyesített halmazokban az $LR(1)$-elemek magjai éppen az $LR(0)$-kanonikus halmazok elemeinek felelnek meg. Így az $LALR(1)$ elemzőnek pontosan ugyanannyi állapota lesz, mint egy $LR(0)$ elemzőnek lenne.

Ez a tulajdonság adja az $LALR(1)$-kanonikus halmazoknak az $LR(0)$-kanonikus halmazokból történő meghatározását, hiszen ha az $LR(0)$-kanonikus halmazok elemeit kiegészítjük a megfelelő előreolvasási szimbólumokkal, akkor az egyesített $LALR(1)$-kanonikus halmazokat fogjuk megkapni.

Vegyük észre, hogy a nem $\mathcal{H}_0$ $LR(1)$-kanonikus halmazokban csak az olyan $LR(1)$-elemekben kezdődik a mag jobb oldala ponttal, amelyek a kanonikus halmazban levő $LR(1)$-elemekből a *closure* függvény alkalmazásával származnak. Így az $LR(1)$-elemek egy kanonikus halmazát nem kell az összes elemének felsorolásával megadni. Nevezzük a $\mathcal{H}_0$ kanonikus halmaz ***törzsének*** az $[S' \to .S, \#]$ $LR(1)$-elemet, egy nem $\mathcal{H}_0$ kanonikus halmaz *törzsének* pedig a kanonikus halmaz azon elemeit, amelyekben a mag jobb oldala nem a pont metaszimbólummal kezdődik. Egy $LR(1)$-kanonikus halmazt tehát a törzsével is megadhatunk, hiszen a törzsből az összes többi $LR(1)$-elem előállítható.

Könnyen belátható, hogy a kanonikus halmaz törzséből a léptetés és a redukció műveletek is meghatározhatók.

Ha az $LR(0)$-kanonikus halmazok törzseinek elemeit kiegészítjük az előreolvasási szimbólumokkal, akkor az egyesített $LR(1)$-kanonikus halmazok törzseit fogjuk megkapni, azaz ha $\mathcal{I}_j$ az $LR(0)$-elemek kanonikus halmazának a törzse, $\mathcal{I}_j$-ből a kiegészítéssel létrehozott egyesített $LR(1)$-kanonikus halmaz törzse $\mathcal{K}_j$ lesz.

Az $LR(0)$-elemekre, ha ismerjük $\mathcal{I}_j$-t, akkor a $read(\mathcal{I}_j, X)$ könnyen meghatározható, hiszen ha $[B \to \gamma.C\delta] \in \mathcal{I}_j$, $C \overset{*}{\to} A\eta$ és $A \to X\alpha$, akkor nyilván $[A \to X.\alpha] \in read(\mathcal{I}_j, X)$. Az $LR(1)$-elemekre ez már nem ilyen egyszerű, ha $[B \to \gamma.C\delta, b] \in \mathcal{K}_j$, $C \overset{*}{\to} A\eta$ és $A \to X\alpha$, akkor még az előreolvasási szimbólumot is meg kell határozni, azaz azt, hogy milyen a-ra lesz $[A \to X.\alpha, a] \in read(\mathcal{K}_j, X)$.

Ha $\eta\delta \neq \varepsilon$, és $a \in Első(\eta\delta b)$, akkor biztosan $[A \to X.\alpha, a] \in read(\mathcal{K}_j, X)$, és azt mondjuk, hogy az a előreolvasási szimbólum a $read(\mathcal{K}_j, X)$ halmaznak ehhez az eleméhez ***spontán generálható,*** hiszen a b szimbólumnak semmilyen szerepe sincs az új előreolvasási szimbólum meghatározásában.

Ha $\eta\delta = \varepsilon$, akkor $[A \to X.\alpha, b]$ lesz a $read(\mathcal{K}_j, X)$ eleme, azaz ebben az elemben a b lesz az előreolvasási szimbólum. Ekkor azt mondjuk, hogy a b előreolvasási szimbólum a $\mathcal{K}_j$-ből ***öröklődik*** a $read(\mathcal{K}_j, X)$ halmaznak ebbe az elemébe.

Ha adott egy $LR(0)$-kanonikus halmaznak az $\mathcal{I}_j$ törzse, akkor tetszőleges X szimbólumra a $read(\mathcal{K}_j, X)$-hez spontán generálható és öröklődő előreolvasási szimbólumok a következő módszerrel határozhatók meg: minden $[B \to \gamma.\delta] \in \mathcal{I}_j$-re határozzuk meg a $\mathcal{K}_j = closure([B \to \gamma.\delta, @])$ halmazt, ahol @ egy dummy-szimbólum,

- ha $[A \to \alpha.X\beta, a] \in \mathcal{K}_j$ és $a \neq @$, akkor $[A \to \alpha X.\beta, a] \in read(\mathcal{K}_j, X)$ és az a szimbólum a $read(\mathcal{K}_j, X)$ halmaznak ebbe az elemébe spontán generálható,
- ha $[A \to \alpha.X\beta, @] \in \mathcal{K}_j$, akkor $[A \to \alpha X.\beta, @] \in read(\mathcal{K}_j, X)$ és a @ öröklődik $\mathcal{K}_j$-ből a $read(\mathcal{K}_j, X)$ halmaznak ebbe az elemébe.

A $\mathcal{K}_0$ kanonikus halmaz törzsének egy eleme van, az elem magja $[S' \to .S]$, és ehhez rendeljük hozzá a # előreolvasási szimbólumot. Mivel az összes $\mathcal{K}_j$ kanonikus halmaz törzsének magja már adott, a fenti módszerrel meghatározható, hogy milyen szimbólumok lesznek a spontán generálható és öröklődő előreolvasási szimbólumok.

Adjuk meg azt az algoritmust, ami a spontán generálás és az öröklődés, valamint az $[S' \to .S, \#]$ elem ismeretében meghatározza az $LR(0)$-kanonikus halmazokból az $LALR(1)$-kanonikus halmazokat.

Megjegyzések a fejezethez

A fordítóprogramok elmélete és írásának gyakorlata egyidős a számítógépekkel, az első programnyelvekkel. Az első fordítóprogramok megírása az 50-es évek elejére tehető. A fordítóprogramok írása sokáig igen nehéz feladat volt, például az első FORTRAN compiler létrehozása 18 emberév munkáját emésztette fel [12]. Ettől kezdve fokozatosan egyre pontosabban definiálták a fordítás problémáit, egyre jobb fordítási módszereket dolgoztak ki, és egyre jobb program-eszközöket hoztak létre a fordítóprogram írás megkönnyítésére.

A munkában nagy előrelépést jelentett a formális nyelvek és az automaták elméletének fejlődése, és azt lehet mondani, hogy ezek fejlődését elsősorban a fordítóprogramok írásának igénye ösztönözte. Napjainkra az elemző programok írása egyszerű rutin-feladattá vált, lényeges új eredmények most már elsősorban a kódoptimalizálás területén találhatók.

A nemdeterminisztikus, visszalépéses algoritmusok az 1960-as évek elején jelentek meg. Az első két sikeres algoritmus a CYK (Cocke–Younger–Kasami) algoritmus volt 1965–67-ből, és az Earley-algoritmus 1965-ből. A precedencia elemzések különböző fajtáinak kialakulása az 1960-as évek végére, az 1970-es évek elejére tehető. Az LR(k) nyelvtanokat Knuth definiálta 1965-ben, az LL(k) nyelvtanok első definíciója az 1970-es évek elejéről származik. Az $LALR(1)$ nyelvtanokat először De Remer tanulmányozta 1971-ben, az $LALR(1)$ elemzés kidolgozása és tanulmányozása az 1980-as évek elejére fejeződött be [10, 11, 12].

Az 1980-as évek közepére nyilvánvalóvá vált, hogy a fordítóprogramokban az LR elemzési módszerek a valóban hatékony módszerek, és azóta a fordítóprogramokban ezeket a módszereket, elsősorban az $LALR(1)$ elemzést alkalmazzák [10].

A fordítóprogramok elméletével és a programok írásával nagyon sok kiváló könyv foglalkozott, közülük az első talán legsikeresebb, de ma már túlhaladott módszereket tárgyaló

könyv Gries [147] könyve volt, ebben elsősorban a precedencia nyelvtanokra vonatkozó eredmények találhatók meg. Az új fordítási módszerekkel foglalkozó első, nagy sikert aratott könyv Aho és Ullman [11] kétkötetes műve volt, ebben részletesen megtalálható a CYK- és Earley-algoritmus is. Ezt az úgynevezett „sárkányos könyv" követte, szintén Aho és Ullman munkája [12]. A könyv bővített, javított kiadása 1986-ban jelent meg, az Aho–Ullman–Sethi szerzőhármastól [10].

A teljesség igénye nélkül megemlítjük még Fischer és LeBlanc [107], Tremblay és Sorenson [340], Waite és Goos [349], Hunter[166], Pittman [282] és Mak [239] könyveit. A legújabb eredményeket tartalmazzák a mostanában megjelent könyvek, többek között Muchnick [256], Grune, Bal, Jacobs és Langendoen [148] művei, vagy a legújabbak, Cooper és Torczon [69] könyve és a Louden [231] által írt könyvfejezet.

Magyarul Csörnyei Zoltán [76, 77] kétkötetes egyetemi jegyzete dolgozza fel a fordítóprogramok elméletének és írásának témakörét. A jegyzetek kissé átdolgozott, rövidített anyaga egyetemi tankönyv formájában is megjelent [78].

A formális nyelvek és automaták elméletének témaköréből több magyar nyelvű könyv és jegyzet is található: ilyen például Bánkfalvi Judit, Bánkfalvi Zsolt és Bognár Gábor 1978-ban [45], valamint Kása Zoltán 2004-ben [213] megjelent műve. Másokban egy-egy fejezet foglalkozik az elméletnek a fordítóprogramokban való alkalmazásával: ilyen például Bach Iván [30], Fülöp Zoltán [116], Révész György [301] és Varga László [347] könyve, valamint Dömösi Pál, Fazekas Attila, Horváth Géza és Mecsei Zoltán [90] digitális kézirata. [30, 90, 301] a CYK- és Earley-algoritmust tárgyalják, a precedencia elemzésekről a [30] és [347] könyvekben olvashatunk, [30, 45, 116] az *LR* elemzésekkel is foglalkozik.

21. Megbízható számolás

Egy tervezett számítás lefuttatásakor előre nem várható hatásoknak lesz kitéve. Néhány példa:

(1) Elveszett, vagy megváltozott adatok a végrehajtás során.

(2) Véletlen, fizikai hibák a gépben.

(3) Váratlan kölcsönhatások a rendszer különböző, egyidőben működő részei között, vagy elveszett hálózati összeköttetések.

(4) Hibák a programban.

(5) Rosszindulatú támadások.

Egyelőre nem ismertek algoritmusok, amelyek a programhibák problémáját megoldanák. A szoftver-mérnöki szakma a programok struktúrájának és előállítási folyamatának tanulmányozásával és javításával közelíti meg ezt a kérdést.

Rosszindulatú támadásokkal az informatikai biztonság szakmája foglalkozik. A javasolt megoldásoknak gyakran része a kriptográfia.

A (3) típusú problémák nagyon fontosak: az osztott számítások tudományát ezek vizsgálatára hozták létre.

Az adattárolási hibák problémája hasonló a megbízható kommunikáció problémájához, melyet az információelmélet tanulmányoz: felfoghatjuk úgy, mint a jelenből jövőbe való kommunikációt. A zaj ellen mindkét esetben ***hibajavító kódok*** segítségével védekezhetünk.

Ebben a fejezetben néhány példát tárgyalunk, főleg a (2) problémafajtából. Itt különbséget kell tenni állandó és átmeneti hibák között. Egy hiba ***állandó,*** ha a számítóberendezés egy része fizikai kárt szenved, és hosszú időre hibás marad, amíg csak külső beavatkozás nem történik (szerelő jön). A hiba ***átmeneti,*** ha csak egyetlen lépésben történik: a berendezésnek az a része, ahol történt, nem károsul, és a következő lépésekben megint helyesen működik. Például, ha a memória egy bitje 0-ról 1-re változik véletlenül, de a következő beírási művelet megint tud 0-t írni az érintett helyre, akkor átmeneti hiba történt. Ha a bit 1-re változott, és a gép nem tudja megint 0-ra változtatni, akkor ez állandó hiba.

Ezek közül a problémák közül egyesek, különösen az átmeneti hibák problémái, egyidősek a gépi számolással. Bármely fizikai számítási hiba részletei függenek a használt számítógéptől (és persze a kivitelezendő számítástól). De miután elvonatkoztatunk egy sereg zavaró részlettől, tiszta, de még mindig nehéz feladatokat adó elméleti megfogalmazásokra

juthatunk, melyekre szép megoldások is léteznek. Érdekes kapcsolatokra bukkanunk más tudományágakkal, mint például a statisztikus fizika és a biológia.

A számítógépipar az utóbbi öt évtizedben elképesztően sikeresen tette a számítógép-alkatrészeket kisebbé, gyorsabbá, és ugyanakkor megbízhatóbbá. A sajtóban naponta olvasható számítógépes rémtörténetek közül feltűnően hiányzik az, ahol a processzor egy 1-est írt 0 helyett, csak úgy szeszélyből. (Ilyen vitathatatlanul történik, de túl ritkán ahhoz, hogy látható működési rendellenesség azonosítható forrásává váljon.) Más oldalról viszont vannak olyan, az átmeneti hibák javítására vonatkozó eredmények, melyeknek általánossága több helyzetben is alkalmazhatóvá teszi őket. Még ha az egyes fizikai processzorok nagyon megbízhatóak is (hiba talán egyszer történik minden 10^{20} ciklusban), amikor egy egész hálózat működését tekintjük számolásnak, akkor a megbízhatatlan hálózati kapcsolatok vagy akár kártevő szándékú résztvevők által okozott problémák sokban hasonlítanak a megbízhatatlan processzorok által okozottakra.

A számítások megbízhatóvá tételének kulcsgondolata a *redundancia,* melyet a következő két módszerként fogalmazhatunk meg:

(i) Tároljuk információnkat olyan formában, hogy egyetlen kis részének elvesztése se végzetes: visszaállítható a megmaradó adatokból. Például, tároljunk mindent több példányban.

(ii) Hajtsuk végre a számítást többször, hogy az esetleges hibás eredményt a többi verzió „leszavazza".

Fejezetünk csak ezeket a módszereket fogja használni, de vannak más figyelemre méltó gondolatok, melyeket nincs itt alkalmunk továbbkövetni. Például, a (ii) módszer különösen költségesnek látszik; jó lenne a sok ismétlést elkerülni. A következő ötletek ezt a dilemmát veszik célba.

(A) Hajtsuk végre számításainkat közvetlenül az információ redundáns formáján: akkor talán elkerülhető a legtöbb ismétlés.

(B) Osszuk be a számítást „szakaszokra" úgy, hogy a továbbiakban is felhasználandó részeredmények olcsón ellenőrizhetők legyenek, minden „mérföldkőnél" a szakaszok között. Csak ha az ellenőrzés hibát talál, akkor ismételjük meg az utolsó szakaszt.

21.1. Valószínűségszámítás

Fejezetünk nem kíván túl magas felkészültséget valószínűségszámításból, de bizonyos tények ismételten felhasználásra kerülnek: ezeket itt átvesszük. Akinek az itt közölt információnál többre van szüksége, az azt bármely haladó valószínűségszámítási tankönyvben megtalálja.

21.1.1. Terminológia

Egy ***valószínűségi teret*** egy $(\Omega, \mathcal{A}, \Pr)$ hármas ír le, ahol Ω az ***elemi események*** halmaza, az $\mathcal{A}$ az Ω bizonyos részhalmazainak osztálya, melyeket ***eseményeknek*** nevezünk, és $\Pr$:

$\mathcal{A} \to [0,1]$ egy függvény. Ha $E \in \mathcal{A}$, akkor a $\Pr(E)$ értéket az E esemény ***valószínűségének*** nevezzük. Megköveteljük, hogy $\Omega \in \mathcal{A}$, és hogy ha $E \in \mathcal{A}$, akkor $\Omega \setminus E \in \mathcal{A}$. Továbbá, ha a halmazoknak egy (esetleg végtelen) sorozata $\mathcal{A}$-ban van, akkor az uniójuk is. Azt is megköveteljük, hogy $\Pr(\Omega) = 1$ és hogy ha $E_1, E_2, \ldots \in \mathcal{A}$ diszjunktak, akkor

$$\Pr\Big(\bigcup_i E_i\Big) = \sum_i \Pr(E_i)\ .$$

Ha $\Pr(F) > 0$, akkor az E esemény F-re vonatkoztatott ***feltételes valószínűségét***

$$\Pr(E \mid F) = \Pr(E \cap F)/\Pr(F)$$

definiálja. Az $E_1, \ldots, E_n$ események ***függetlenek,*** ha minden $1 \le i_1 < \cdots < i_k \le n$ sorozatra

$$\Pr(E_{i_1} \cap \cdots \cap E_{i_k}) = \Pr(E_{i_1}) \cdots \Pr(E_{i_k}).$$

21.1. példa. Legyen $\Omega = \{1, \ldots, n\}$, ahol $\mathcal{A}$ az Ω összes részhalmazából áll, és $\Pr(E) = |E|/n$.

Általánosabban, egy ***diszkrét valószínűségi tér*** meg van adva, ha adott egy megszámlálható $\Omega = \{\omega_1, \omega_2, \ldots\}$ halmaz, és egy $p_1, p_2, \ldots$ sorozat, melyre $p_i \ge 0$, $\sum_i p_i = 1$. Az események $\mathcal{A}$ halmaza az Ω összes részhalmazainak halmaza, és egy $E \subset \Omega$ esemény valószínűsége így van definiálva: $\Pr(E) = \sum_{\omega_i \in E} p_i$.

Egy ***valószínűségi változó*** egy valamilyen Ω valószínűségi tér feletti f függvény valós szám értékekkel, továbbá azzal a tulajdonsággal, hogy minden $\{\omega : f(\omega) < c\}$ formájú halmaz esemény, tehát $\mathcal{A}$-ban van. Valószínűségi változókat gyakran X, Y, Z nagybetűkkel jelölünk, esetleg indexekkel, és az ω független változót elhagyjuk az $X(\omega)$ teljes formából. Az $\{\omega : X(\omega) < c\}$ eseményt így is írjuk: $[X < c]$. Ezt a jelölést analóg módon bonyolultabb eseményekre is ki fogjuk terjeszteni. Egy X valószínűségi változó ***eloszlása*** az $F(c) = \Pr[X < c]$ függvény. Sokszor csak változóink eloszlását adjuk meg, és nem is említjük a hozzájuk tartozó valószínűségi teret, ha az összefüggésből világos, hogy így-vagy-úgy megadható. Beszélhetünk két vagy több valószínűségi változó ***közös eloszlásáról,*** de csak akkor, ha feltesszük, hogy mint függvények, közös valószínűségi téren definiálhatók. Az $X_1, \ldots, X_n$ közös eloszlással rendelkező valószínűségi változókat ***függetleneknek*** nevezzük, ha az összes ilyen típusú esemény n-es független: $[X_1 < c_1], \ldots, [X_n < c_n]$.

Ha egy X valószínűségi változó az $x_1, x_2, \ldots$ értékeket $p_1, p_2, \ldots$ valószínűséggel veszi fel, akkor a ***várható értékét*** az

$$\mathbf{E}X = p_1 x_1 + p_2 x_2 + \cdots$$

képlet definiálja. Könnyű látni, hogy a várható érték lineárisan függ a valószínűségi változótól:

$$\mathbf{E}(\alpha X + \beta Y) = \alpha \mathbf{E}X + \beta \mathbf{E}Y\ ,$$

még akkor is, ha X, Y nem független. Ha az X, Y változók függetlenek, akkor a várható értékeket össze is lehet szorozni:

$$\mathbf{E}XY = \mathbf{E}X \cdot \mathbf{E}Y\ . \tag{21.1}$$

Van egy fontos, egyszerű egyenlőtlenség, a ***Markov egyenlőtlenség,*** mely azt mondja, hogy egy tetszőleges nemnegatív X valószínűségi változóra és bármely $\lambda > 0$ értékre

$$\Pr[X \ge \lambda] \le \mathbf{E}X/\lambda\ . \tag{21.2}$$

21.1.2. A nagy számok törvénye (nagy eltérésekkel)

Az itt megadott egyenlőtlenségek a későbbiekben hasznosak lesznek:

$$\frac{x}{1+x} \leq \ln(1+x) \leq x, \qquad \text{ha } x > -1\ . \tag{21.3}$$

Itt a jól ismert $\ln(1+x) \leq x$ felső korlát abból következik, hogy az $\ln(1+x)$ függvény görbéje az $x = 0$ pontban húzott tangens alatt fekszik. Az alsó korlátot az $\frac{1}{1+x} = 1 - \frac{x}{1+x}$ azonosságból és a következőkből kapjuk:

$$-\ln(1+x) = \ln\frac{1}{1+x} = \ln\left(1 - \frac{x}{1+x}\right) \leq -\frac{x}{1+x}\ .$$

Legyenek $X_1, \ldots, X_n$ független, egyforma eloszlású valószínűségi változók, melyekre

$$\Pr[\, X_i = 1\,] = p, \quad \Pr[\, X_i = 0\,] = 1 - p\ .$$

Legyen

$$S_n = X_1 + \cdots + X_n.$$

Meg akarjuk becsülni a $\Pr[\, S_n \geq fn\,]$ valószínűséget minden $0 < f < 1$ konstansra. A „nagy számok törvénye" azt állítja, hogy ha $f > p$, akkor ez a valószínűség gyorsan 0-hoz tart $n \to \infty$ esetén, míg ha $f < p$, akkor gyorsan 1-hez tart. Legyen

$$D(f,p) = f\ln\frac{f}{p} + (1-f)\ln\frac{1-f}{1-p} \tag{21.4}$$

$$> f\ln\frac{f}{p} - f = f\ln\frac{f}{ep}\ , \tag{21.5}$$

ahol az egyenlőtlenség (hasznos, ha f kicsi és $ep < f$) következik $1 > 1-p > 1-f$-ből és $\ln(1-f) \geq -\frac{f}{1-f}$-ből (lásd (21.3)-t). A logaritmus konkáv tulajdonsága segítségével megmutatható, hogy $D(f,p)$ mindig nemnegatív, és csak akkor 0, ha $f = p$ (lásd a 21.1-1.. gyakorlatot).

21.1. tétel (Nagy eltérések pénzfeldobásokra). *Ha $f > p$, akkor*

$$\Pr[\, S_n \geq fn\,] \leq e^{-nD(f,p)}\ .$$

Eszerint a tétel szerint ha $f > p$, akkor $\Pr[\, S_n > fn\,]$ exponenciális sebességgel tart 0-hoz. A (21.5) egyenlőtlenséggel tovább egyszerűsítve a

$$\Pr[\, S_n \geq fn\,] \leq e^{-nf\ln\frac{f}{ep}} = \left(\frac{ep}{f}\right)^{nf} \tag{21.6}$$

formát kapjuk, amely hasznos akkor, ha f kicsi, és $ep < f$.

Bizonyítás. Egy később megválasztandó $\alpha > 1$ valós számra legyen Y_n az a valószínűségi változó, amely α, ha $X_n = 1$ és 1 ha $X_n = 0$, és legyen $P_n = Y_1 \cdots Y_n = \alpha^{S_n}$: akkor

$$\Pr[\, S_n \geq fn\,] = \Pr[\, P_n \geq \alpha^{fn}\,]\ .$$

A Markov-egyenlőtlenség (lásd (21.2)-t) alkalmazásával

$$\Pr[\, P_n \geq \alpha^{fn}\,] \leq \mathbf{E}\, P_n/\alpha^{fn} = (\mathbf{E}\, Y_1/\alpha^f)^n\ ,$$

ahol $\mathbf{E}Y_1 = p\alpha + (1-p)$. Válasszunk így: $\alpha = \frac{f(1-p)}{p(1-f)}$, ez > 1, ha $p < f$. Ekkor $\mathbf{E}Y_1 = \frac{1-p}{1-f}$, és így

$$\mathbf{E}Y_1/\alpha^f = \frac{p^f(1-p)^{1-f}}{f^f(1-f)^{1-f}} = e^{-D(f,p)} .$$

■

Ez a tétel binomiális együtthatókra is jól használható becsléseket ad. Legyen

$$h(f) = -f\ln f - (1-f)\ln(1-f) .$$

Ezt néha az $(f, 1-f)$ valószínűségeloszlás ***entrópiájának*** nevezzük (a 2 alapú logaritmus helyett e alapú logaritmusban mérve). A (21.3) egyenlőtlenségből a

$$-f\ln f \le h(f) \le f\ln\frac{e}{f} \tag{21.7}$$

becslést kapjuk, ami hasznos kis f-re.

21.2. következmény. *Az $f \le 1/2$ esetben*

$$\sum_{i\le fn}\binom{n}{i} \le e^{nh(f)} \le \left(\frac{e}{f}\right)^{fn} . \tag{21.8}$$

Ha például $f = k/n$, ahol $k \le n/2$, akkor

$$\binom{n}{k} = \binom{n}{fn} \le \left(\frac{e}{f}\right)^{fn} = \left(\frac{ne}{k}\right)^{k} . \tag{21.9}$$

Bizonyítás. A 21.1. tétel azt adja az $f > p = 1/2$ esetre, hogy

$$2^{-n}\sum_{i\ge fn}^{n}\binom{n}{i} = \Pr[\,S_n \ge fn\,] \le e^{-nD(f,p)} = 2^{-n}e^{nh(f)} ,$$

$$\sum_{i\ge fn}^{n}\binom{n}{i} \le e^{nh(f)} .$$

A $g = 1-f$ helyettesítéssel, észre véve az $\binom{n}{f} = \binom{n}{g}$, $h(f) = h(g)$ szimmetriákat és (21.7)-et kapjuk a (21.8) eredményt. ■

21.3. megjegyzés. *A (21.6) egyenlőtlenség a $\Pr[\,S_n \ge fn\,] \le \binom{n}{fn}p^{fn}$ triviális becslésből is következik, ha azt (21.9)-cel kombináljuk.*

Gyakorlatok

21.1-1. Bizonyítsuk be a főszöveg állítását, hogy $D(f,p)$ mindig nemnegatív, és csak akkor 0, ha $f = p$.

21.1-2. Az $f = p + \delta$ értékkel, vezessük le a 21.1. tételből a következő hasznos korlátot:

$$\Pr[\,S_n \ge fn\,] \le e^{-2\delta^2 n} .$$

Útmutatás. Legyen $F(x) = D(x,p)$, és használjuk a Taylor-formulát: $F(p+\delta) = F(p) + F'(p)\delta + F''(p+\delta')\delta^2/2$, ahol $0 \le \delta' \le \delta$.

21.2. Logikai hálózatok

Olyan számítási modellben, mely hibákat is figyelembe vesz, természetes az a feltevés, hogy hibák *mindenütt* megjelenhetnek. A számítógép legismerősebb formája – melyben a processzor és tár elválik egymástól – ilyen körülmények között rendkívül sebezhetőnek tűnik: amíg a processzor nem „néz oda”, a zaj kijavíthatatlan kárt okozhat a tárban. Dolgozzunk ezért inkább olyan modellekkel, melyek *párhuzamosak:* a rendszer minden része dolgoz fel információt, nem csak egyes kitüntetett helyei. Ekkor a hibajavítást a rendszer minden részébe be lehet építeni. A legjobban ismert párhuzamos számítási modellre, a logikai hálózatokra szorítkozunk.

21.2.1. Boole-függvények és kifejezések

Vessünk egy pillantást a számítógép belsejébe (vagy inkább az integrált áramkör belsejébe, egy mikroszkóppal). A rengeteg irreleváns fizikai részlettől megzavarodva, inkább a hálózattervező rajzai felé fordulunk; mégpedig a tervezés olyan szakaszában, amikor ezek az áramkör legkisebb elemeit számítási funkcióik megjelölésével együtt mutatják. Olyan vonalak hálózatát fogjuk látni, melyek két állapotot vehetnek fel (elektromos potenciáljuk szerint): „magas” vagy „alacsony”, „igaz” vagy „hamis”, vagy, ahogy mi fogjuk jelölni, 1 vagy 0. A pontok, melyeket ezek a vonalak összekötnek, az ismerős ***logikai összetevők:*** a számítás legalacsonyabb szintjén a tipikus számítógép ***biteket*** dolgoz fel. Egész számok, lebegőpontos számok, betűk mind megadhatók bitfüzérekkel, és a szokásos elemi aritmetikai műveletek is összeállíthatók bit-műveletek összekapcsolásával.

21.4. definíció. *Egy* ***Boole-vektorfüggvény*** *egy* $f : \{0,1\}^n \to \{0,1\}^m$ *leképezés. Többnyire az* $m = 1$ *esettel fogunk foglalkozni, és ekkor* ***Boole-függvényről*** *fogunk beszélni.*

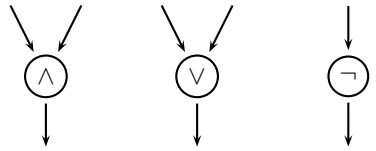
Az $f(x_1, \ldots, x_n)$ kifejezés változóit néha ***logikai változóknak, Boole-változóknak,*** vagy ***biteknek*** nevezzük.

21.2. példa. Egy irányítatlan, N pontú G gráfra érdekelhet minket az a kérdés, van-e Hamilton-köre (a G összes csúcsainak egy olyan $(u_1, \ldots, u_n)$ felsorolása, hogy minden $i < n$-re (u_i, u_{i+1}) egy él, és (u_n, u_1) is egy él). Ezt a kérdést egy f Boole-függvény a következőképpen írja le. A gráfot $\binom{N}{2}$ logikai változóval adhatjuk meg: x_{ij} $(1 \le i < j \le N)$: $x_{ij} = 1$, ha él fut az i és j csúcsok között. Az $f(x_{12}, x_{13}, \ldots, x_{N-1,N})$ érték 1, ha G-ben van egy Hamilton-kör, és 0 egyébként.

21.3. példa. *Boole-vektorfüggvény.* Legyen $n = m = 2k$, legyen bemenetünk két k-bit hosszúságú bitfüzérrel felírt egész szám: u és v: $x = (u_1, \ldots, u_k, v_1, \ldots, v_k)$. A kimenet az $y = u \cdot v$ szorzat (bináris formában írva): ha $u = 5 = (101)_2$, $v = 6 = (110)_2$, akkor $y = u \cdot v = 30 = (11110)_2$.

Csak négy egyváltozós Boole-függvény van: 0 és 1 konstansok, az azonosság, és a ***tagadás***: $x \to \neg x = 1 - x$. A következő további kétváltozós Boole-függvényeket említjük meg: a ***konjunkció*** (logikai ÉS) művelete:

$$x \wedge y = \begin{cases} 1, & \text{ha } x = y = 1\ , \\ 0 & \text{különben}\ , \end{cases}$$



21.1. ábra. ÉS, VAGY és NEM kapu

ami azonos a szorzás műveletével. A ***diszjunkció,*** azaz VAGY művelete:

$$x \vee y = \begin{cases} 0, & \text{ha } x = y = 0\,, \\ 1 & \text{különben}\quad . \end{cases}$$

Könnyű látni, hogy $x \vee y = \neg(\neg x \wedge \neg y)$: más szóval az $x \vee y$ diszjunkció a $\neg$, $\wedge$ függvényekből kifejezhető az ***összetétel*** műveletének segítségével. A következő kétváltozós logikai függvényeket ugyancsak gyakran használják:

$$\begin{aligned} x \to y &= \neg x \vee y && \text{(implikáció)}\quad, \\ x \leftrightarrow y &= (x \to y) \wedge (y \to x) && \text{(ekvivalencia)}\quad, \\ x \oplus y &= x + y \bmod 2 = \neg(x \leftrightarrow y) && \text{(bináris összeadás)}\quad . \end{aligned}$$

Véges sok Boole-függvény elégséges ahhoz, hogy az *összes többit* kifejezzük: tehát tetszőlegesen bonyolult Boole-függvényeket ki lehet „számolni” „elemi” műveletekkel. Bizonyos értelemben ez történik minden számítógépben.

21.5. definíció. *A Boole-függvények egy Q halmaza **teljes bázis,** ha minden Boole-függvény kifejezhető a Q elemeiből képezett összetétellel.*

21.6. állítás. *Az $\{\wedge, \vee, \neg\}$ halmaz teljes bázis. Más szóval, minden Boole-függvény megadható egy logikai kifejezéssel, mely csak ezt a két műveletet használja.*

A bizonyítás a kijelentéslogika bármely tankönyvében megtalálható. Mivel $\vee$ kifejezhető $\{\wedge, \neg\}$ segítségével, ez utóbbi halmaz is teljes bázis (és $\{\vee, \neg\}$ is).

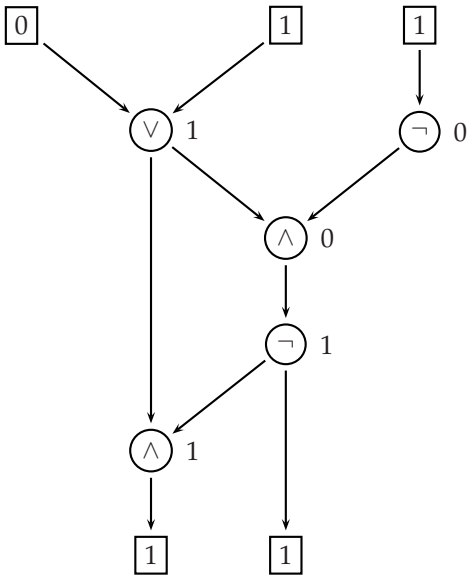
Mostantól egy ***logikai kifejezésen*** (képleten) olyan kifejezést értünk, amely valamilyen adott teljes bázis elemeiből van felépítve. Ha a teljes bázist nem említjük, akkor $\{\wedge, \vee, \neg\}$-ra gondolunk.

Általában egy és ugyanaz a Boole-függvény többféleképpen is megadható logikai kifejezésekkel. Az adott kifejezésből már könnyű a függvény kiszámítása. Csakhogy a legtöbb Boole-függvényt csak nagyon nagy logikai kifejezésekkel lehet leírni (lásd a 21.2-4. gyakorlatot).

21.2.2. Logikai hálózatok

Egy logikai kifejezés néha azért nagy, mert felírása nem ad lehetőséget a részeredmények újrafelhasználására. (Például az

$$((x \vee y \vee z) \wedge u) \vee (\neg(x \vee y \vee z) \wedge v)$$



21.2. ábra. Egy értékadás (értékek a csúcsokon, konfiguráció) a kapukon át terjed tovább. Ez a "számolás".

kifejezésben az $x \vee y \vee z$ rész kétszer jelenik meg.) Ezt a hiányt pótolja a következő, általánosabb formalizmus.

Egy logikai hálózat lényegében egy ciklusmentes irányított gráf, melynek minden csúcsa egy (valamely teljes bázisból vett) Boole-függvényt számol ki a bemenő élein érkező biteken, és az eredményt kiküldi a kimenő élein (lásd a 21.2. ábrát). Lássuk a szabatos definíciót.

21.7. definíció. *Legyen Q a Boole-függvények egy teljes bázisa. Egy N számra legyen $V = \{1, \ldots, N\}$ a* ***csúcsok*** *egy halmaza. Egy* ***logikai hálózat*** *a Q bázis fölött a következőkkel van megadva:*

$$\mathcal{N} = (V, \{k_v : v \in V\}, \{\arg_j(v) : v \in V;\ j = 1, \ldots, k_v\}, \{b_v : v \in V\}) . \tag{21.10}$$

Minden v csúcshoz egy k_v természetes szám mutatja ***bemeneteinek*** *számát. A* ***forrásokat,*** *azaz olyan v csúcsokat, melyekre $k_v = 0$,* ***bemenet-csúcsoknak*** *nevezzük: jelöljük őket, növekedő sorrendben, így:*

$$\mathrm{be}_i \quad (i = 1, \ldots, n) .$$

Minden nem-bemeneti v csúcshoz tartozik egy

$$b_v(y_1, \ldots, y_{k_v})$$

Boole-függvény a Q teljes bázisból: ezt a v csúcs ***kapujának*** *nevezzük. A változók száma a csúcsba bejövő élek számával egyenlő. A gráf* ***nyelőit,*** *a kimenő élek nélküli csúcsokat,*

kimeneti csúcsoknak *nevezzük. Őket így jelölhetjük:*

$$\mathrm{ki}_i \quad (i = 1, \ldots, m) \; .$$

(A mi logikai hálózatainknak többnyire csak egy kimeneti csúcsa lesz.) Minden nembemeneti v csúcshoz és minden $j = 1, \ldots, k_v$ számhoz egy $\arg_j(v) \in V$ csúcs tartozik (a csúcs, mely a v csúcs kapujába a j-edik változó értékét küldi). A hálózat egy $G = (V, E)$ gráfot definiál, melynek élhalmaza

$$E = \{ \, (\arg_j(v), v) : v \in V, \;\; j = 1, \ldots, k_v \, \} \; .$$

Megköveteljük, hogy $\arg_j(v) < v$ teljesüljön minden j, v-re (a csúcsokat az $1, \ldots, N$ természetes számokkal azonosítottuk): ebből következik, hogy a G gráfban nincsenek irányított ciklusok. A hálózat

$$|\mathcal{N}|$$

nagysága *a csúcsok száma. Egy v csúcs* ***mélysége*** *a bemeneti csúcsokból v-be vezető leghosszabb irányított út hossza. A hálózat mélysége a legmélyebb kimeneti csúcs mélysége.*

21.8. definíció. *Egy logikai hálózat egy* ***bemeneti értékadása,*** *vagy* ***bemeneti konfigurációja*** *egy $\boldsymbol{x} = (x_1, \ldots, x_n)$ Boole-vektor, amely az x_i értéket adja a be_i csúcsnak:*

$$\mathrm{ért}_{\boldsymbol{x}}(v) = y_v(\boldsymbol{x}) = x_i \; ,$$

ha $v = \mathrm{be}_i$, $i = 1, \ldots, n$. Az $y_v(\boldsymbol{x})$ függvény egyértelműen kiterjeszthető a hálózat összes többi csúcsára egy $v \mapsto y_v(\boldsymbol{x})$ ***konfigurációvá****, a következőképpen. Ha a b_v kapunak k változója van, akkor*

$$y_v = b_v(y_{\arg_1(v)}, \ldots, y_{\arg_k(v)}) \; . \tag{21.11}$$

Például, ha $b_v(x, y) = x \wedge y$, és $u_j = \arg_j(v)$ $(j = 1, 2)$ a v csúcshoz tartozó bemenő csúcsok, akkor $y_v = y_{u_1} \wedge y_{u_2}$. Ezt a folyamatot, melyben a fenti egyenletet követve fokozatosan kiterjesztjük a konfigurációt, a hálózat ***számolásának*** *is nevezzük. Az $y_{\mathrm{ki}_i}(\boldsymbol{x})$ $(i = 1, \ldots, m)$ értékvektor a számolás* ***eredménye****. Azt mondjuk, hogy a logikai hálózat* ***kiszámítja*** *az*

$$\boldsymbol{x} \mapsto (y_{\mathrm{ki}_1}(\boldsymbol{x}), \ldots, y_{\mathrm{ki}_m}(\boldsymbol{x})) \; .$$

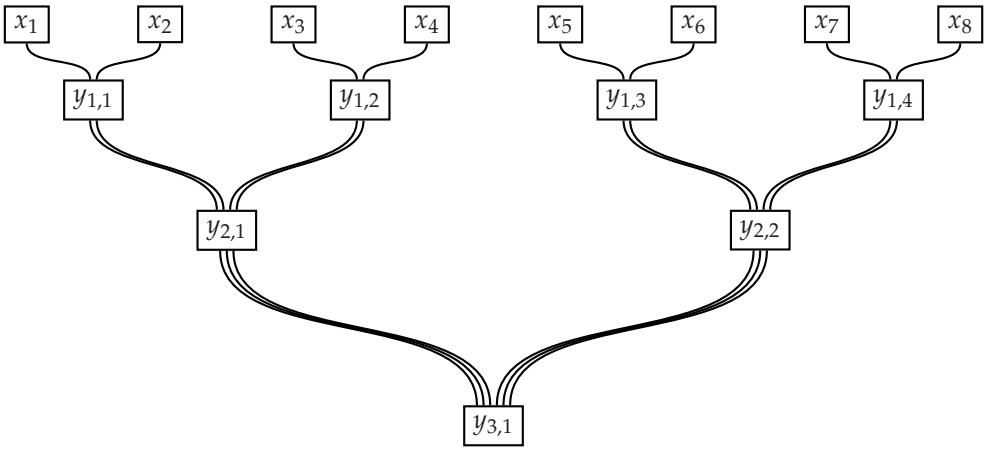
vektorfüggvényt. Az értékadási eljárást ***szakaszokban*** *is lehet végezni: a t-edik szakaszban minden t mélységű csúcs értéket kap.*

Az élekhez is rendelünk értékeket: egy élhez rendelt érték ugyanaz, mint amit a kiindulópontjában találunk.

21.2.3. Gyors összeadás logikai hálózattal

Egy logikai hálózat mélysége annak a legrövidebb időnek tekinthető, amennyi alatt ez a hálózat a bemenetvektorból a kimenetvektort ki tudja számolni. Logikai hálózatok egy alkalmazásaként, dolgozzunk ki egy hálózatot, mely bemenő bitjeinek összegét nagyon gyorsan számolja ki. Az eredményre e fejezetben később szükség lesz hibajavító célokra.

21.9. definíció. *Egy logikai hálózat* ***közel-többséget*** *számol ki, ha kimenete egy y bit, a következő tulajdonsággal: ha a bemeneti biteknek legalább $3/4$-e b-vel egyenlő, akkor $y = b$.*



21.3. ábra. Naiv párhuzamos összeadás.

Hálózatunk mélysége nyilván $\Omega(\lg n)$, mert a kimenetből utaknak kell vezetni a bemenetek többségébe. A többség kiszámítása érdekében a bemeneti bitek összeadásának feladatát is megoldjuk.

21.10. tétel.

(a) *Egy teljes bázis felett, mely az összes 3-változós Boole-függvényeket tartalmazza, minden n-re létezik egy n bemenetnagyságú és legfeljebb* $3\lg(n+1)$ *mélységű logikai hálózat, melynek kimenetvektora a bemenő bitek összegét adja meg, binárisan ábrázolva.*

(b) *Ugyanezen teljes bázis felett, minden n-re létezik egy n bemenetnagyságú és* $\leq 2\log(n+1)$ *mélységű logikai hálózat, mely közel-többséget számol ki.*

Bizonyítás. Először a (a) részt bizonyítjuk. Az egyszerűség kedvéért tegyük fel, hogy $n = 2^k - 1$: ha n nem ilyen formájú, akkor néhány ál-bemenetet vezethetünk be. A naiv megközelítés a 21.3. ábra szerint járna el: először legyen $y_{1,1} = x_1 + x_2$, $y_{1,2} = x_3 + x_4$, …, $y_{1,2^{k-1}} = x_{2^k-1} + x_{2^k}$, majd számoljuk ki a következőket: $y_{2,1} = y_{1,1} + y_{1,2}$, $y_{2,2} = y_{1,3} + y_{1,4}$, és így tovább. Ekkor a $y_{k,1} = x_1 + \cdots + x_{2^k}$ értéket k szakaszban megkapjuk.

Némi nehézséget okoz, hogy $y_{i,j}$ szám, nem bit, ezért egy *bitvektor* adja meg, azaz a hálózat csúcsainak egy csoportja, nem csak egyetlen csúcs. Csakhogy az általános

$$y_{i+1,j} = y_{i,2j-1} + y_{i,2j}$$

összeadási művelet, naiv módon végrehajtva a tipikus esetben több, mint konstans mennyiségű „lépésbe” kerül: az $y_{i,j}$ számok hosszúsága $i+1$, ezért az összeadás további i-vel növelheti a mélységet, az $1 + 2 + \cdots + k = \Omega(k^2)$ értéket adva.

A következő észrevétellel csökkenthető a mélység. Legyen a, b, c három szám bináris jelölésben: például $a = \sum_{i=0}^{k} a_i 2^i$. Egyszerű párhuzamos képletek segítségével e három szám összegét két másik szám összegével fejezhetjük ki: $a + b + c = d + e$, ahol d, e is binárisan ábrázolt számok:

$$\begin{aligned} d_i &= a_i + b_i + c_i \bmod 2 \,, \\ e_{i+1} &= \lfloor (a_i + b_i + c_i)/2 \rfloor \,. \end{aligned} \tag{21.12}$$

Miután mindkét képlet egyetlen 3-változós kapuval kiszámolható, 3 számot 2-vel lehet helyettesíteni egyetlen párhuzamos számolási lépésben (az összeg megtartásával). Két ilyen lépés 4 számot 2-re csökkent. Tehát $2(k-1)$ lépésben 2^k tag összegét 2 tag összegére redukálhatjuk. E két szám szokásos módon való összeadása a mélységet még k-val növeli: azaz 2^k bitet $3k-2$ lépésben tudunk összeadni.

A (b) rész bizonyításához építsük meg a (a) rész hálózatát, de az utolsó összeadás nélkül: a kimenet két k-bit szám, melynek összege érdekel minket. Ezen számok legmagasabb helyi értékű nemnulla bitje valamely $< k$ helyen található. Ha az összeg több, mint 2^{k-1}, akkor e számok egyikében nem-0 bit van a $(k-1)$ vagy $(k-2)$ helyeken. Ez megfelelő 3-bemenetű kapuk két további alkalmazásával felismerhető. ■

Gyakorlatok

21.2-1. Mutassuk meg, hogy $\{1, \oplus, \wedge\}$ teljes bázis.

21.2-2. Mutassuk meg, hogy az x NOR $y = \neg(x \vee y)$ függvény önmagában is teljes bázist alkot.

21.2-3. Rögzítsük az $\{\wedge, \neg\}$ teljes bázist. Bizonyítsuk a 21.6. kijelentést (vagy keressük meg egy tankönyvben). Adjunk segítségével felső korlátot, általános n-változós f Boole-függvény esetén, a következőkre:

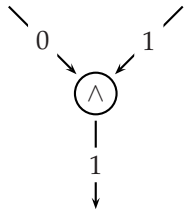
(a) az f-et leíró legkisebb logikai kifejezés nagysága;

(b) az f-et kiszámító legkisebb logikai hálózat nagysága;

(c) egy f-et kiszámító logikai hálózat legkisebb lehetséges mélysége.

21.2-4. Mutassuk meg, hogy minden n-re létezik egy n-változós f Boole-függvény, melyre minden, az $\{\wedge, \neg\}$ teljes bázisban őt kiszámoló logikai hálózat $\Omega(2^n/n)$ csúcsot tartalmaz. *Útmutatás.* Egy $c > 0$ konstansra, becsüljük felülről a legfeljebb $c2^n/n$ csúcsú logikai hálózatok számát, és hasonlítsuk ezt össze az n-változós Boole-függvények számával.

21.2-5. Tekintsük azt az $\mathcal{M}_3^r$ hálózatot 3^r bemenettel, melynek egyetlen kimenő bitjét r-szer iterált 3-bemenetű többségi kapukkal kapjuk a bemenetekből. Mutassuk meg, hogy létezik egy olyan $\boldsymbol{x}$ bemenetvektor, amely csak $n^{1/\lg 3}$ helyen 1, de amellyel $\mathcal{M}_3^r$ kimenete 1 lesz. Tehát a bemeneteknek elenyésző kisebbségét is el lehet ügyesen úgy rendezni, hogy a hálózat kimenetét meghatározza.



21.4. ábra. Hiba egy kapunál.

21.3. Költséges hibatűrés logikai hálózatokban

Vegyünk egy $\mathcal{N}$ logikai hálózatot, a 21.7. definíció szerint. Ha zaj is felléphet, akkor az

$$y_v = \text{ért}_{\boldsymbol{x}}(v)$$

értékeket nem határozza meg már a (21.11) képlet. Helyükre az Y_v valószínűségi változók lépnek. Nevezzük az $(Y_v : v \in V)$ véletlen vektort ***véletlen konfigurációnak***.

21.11. definíció. *A v csúcsnál legyen*

$$Z_v = b_v(Y_{\text{arg}_1(v)}, \ldots, Y_{\text{arg}_k(v)}) \oplus Y_v \ , \tag{21.13}$$

*azaz $Z_v = 1$, ha Y_v nem azonos a zaj nélküli b_v kapu által az $Y_{\text{arg}_j(v)}$ bemenetekből kiszámolt értékkel. (Lásd a 21.4 ábrát.) A csúcsok halmazát, ahol Z_v nem nulla, a **tévedések halmazának** nevezzük (ezzel elkerüljük a többféleképpen érthető "hiba" szót).*

*Nevezzük az $\text{ért}_{\boldsymbol{x}}(v) \oplus Y_v$ különbséget a v csúcs **eltérésének**.*

Szorítsuk meg, milyen fajta zajt engedünk meg. Minden tévedés legfeljebb ε valószínűséggel léphet fel. Két megadott helyen egyszerre legfeljebb ε^2 valószínűséggel léphet fel tévedés, és így tovább.

21.12. definíció. *Adott $\varepsilon > 0$-ra azt mondjuk, hogy az $(Y_v : v \in V)$ véletlen konfiguráció **ε-megengedett**, ha*

(a) *$Y_{\text{be}(i)} = x_i$ minden $i = 1, \ldots, n$-re.*

(b) *A nem bemeneti csúcsok minden C halmazára*

$$\Pr[\, Z_v = 1 \text{ minden } v \in C\text{-re }] \le \varepsilon^{|C|} \ . \tag{21.14}$$

Más szóval, egy ε-megengedett véletlen konfigurációban, k különböző megadott kapunál legfeljebb ε^k valószínűséggel léphet fel egyszerre tévedés. Így követeljük meg, hogy ne csak kicsi legyen a tévedések valószínűsége, de ráadásul a tévedések ne „esküdhessenek össze". A megengedettségi feltétel teljesül, ha a tévedések egymástól függetlenül, $\le \varepsilon$ valószínűséggel következnek be.

Célunk olyan hálózatot készíteni, amely nagy valószínűséggel helyesen működik, a mindig jelenlévő zaj ellenére: más szóval, a hibák *nem halmozódnak*. Ezt a fogalmat formalizáljuk a következőkben.

21.13. definíció. *Egy $\mathcal{N}$ hálózatot, melynek kimenő csúcsa w, akkor nevezünk (ε,δ)-**ellenállónak,** ha minden $\boldsymbol{x}$ bemenetvektorhoz, minden ε-megengedett $\boldsymbol{Y}$ konfigurációban* $\Pr[\, Y_w \neq \text{ért}_{\boldsymbol{x}}(w)\,] \leq \delta$.

Járjuk körül egy kicsit ezt a fogalmat. Nincs (ε,δ)-ellenálló hálózat, ha $\delta < \varepsilon$, mert még az utolsó kapu is ε valószínűséggel tévedhet. Engedjük meg ezért, egy kicsit nagylelkűen, a $\delta > 2\varepsilon$ lehetőséget. Nyilván minden $\mathcal{N}$ hálózathoz és minden $\delta > 0$ értékhez lehet olyan kicsi ε-t választani, amely biztosítja, hogy (ε,δ)-ellenálló legyen $\mathcal{N}$. De hát nem ezt akarjuk elérni: remélhetőleg nem lesz szükség egyre megbízhatóbb kapukra, ahányszor csak nagyobb hálózatokat akarunk építeni. Egy olyan

$$F(N,\delta)$$

függvényt keresünk tehát, és egy olyan $\varepsilon_0 > 0$ tévedés-korlátot, hogy minden $\varepsilon < \varepsilon_0$ és $\delta \geq 2\varepsilon$ esetén, minden N nagyságú $\mathcal{N}$ Boole-hálózatra, legyen egy $F(N,\delta)$ nagyságú (ε,δ)-ellenálló $\mathcal{N}'$ hálózat, amely ugyanazt a függvényt számolja ki, mint $\mathcal{N}$. Ha ezt elérjük, akkor elmondhatjuk, hogy megakadályoztuk a hibák halmozódását. Persze azt akarjuk, hogy $F(N,\delta)$ viszonylag kicsi legyen, és ε_0 nagy (nagyobb zajt megengedve). Az $F(N,\delta)/N$ függvényt ***redundanciának*** nevezhetjük: ezzel szorzóval kell növelni a hálózat nagyságát, hogy ellenállóvá tegyük. Érdemes megjegyezni, hogy a probléma még akár $\delta = 1/3$ esetén sem triviális. Ha a hibák halmozódása előtt nincs akadály, akkor fokozatosan minden információ elvész a kívánt kimenő értékről, és semmilyen $\delta < 1/2$ nem garantálható.

Hogy javítsuk ki a hibákat? Egy egyszerű gondolat: számoljunk ki "mindent" 3-szor, aztán folytassuk a többségi szavazás eredményével.

21.14. definíció. *Egy d páratlan természetes számra, a d bemenetű többségi kapu az a Boole-függvény, amelynek kimenő értéke egyenlő a bemenő értékek többségével.*

A d-bemenetű többségi érték kiszámítható, $O(d)$ ÉS és VAGY kapu segítségével.

Miért várható, hogy a többségi szavazás segít? A következő, *informális diszkusszió* segít megérteni az előnyöket és buktatókat. Tegyük fel egy pillanatra, hogy az egész számítás kimenete egyetlen bit. Ha bármelyik, függetlenül kiszámolt eredmény tévedési valószínűsége δ, akkor annak a valószínűsége, hogy legalább 2 téves közülük, $3\delta^2$-el korlátozható. Mivel maga a többségi szavazás is hibázhat $\leq \varepsilon$ valószínűséggel, a kudarc teljes valószínűségét $3\delta^2 + \varepsilon$ korlátozza. Tehát a hiba δ valószínűsége csökken, a $3\delta^2 + \varepsilon < \delta$ feltétel mellett.

Úgy látszik tehát, hogy ha δ kicsi, akkor ismétlés és többségi szavazás tovább csökkentheti. Persze, ha a hibavalószínűség növekedését meg akarjuk akadályozni, a többségi szavazást újra és újra végre kell hajtani. Tegyük fel például, hogy számításunk t egymást követő szakaszból áll. Az i-edik szakasz után korlátunk a hibás kimenet valószínűségére δ_i. Minden szakasz után többségi szavazást akarunk végrehajtani. Hajtsuk végre az i szakaszt háromszor. A hiba valószínűségére most a

$$\delta_{i+1} = \delta_i + 3\delta^2 + \varepsilon \tag{21.15}$$

korlátot kapjuk. Tehát a különböző szakaszok hibavalószínűségei halmozódnak, és még a $3\delta^2 + \varepsilon < \delta$ egyenlőtlenség esetén is csak a $\delta_t < (t-1)\delta$ korlátot kapjuk. Ez a stratégia tehát nem működik tetszőlegesen nagy számításokra.

Egy vad ötlet a halmozódás elkerülésére: ismételjünk meg *mindent* háromszor, ami az i-edik szakasz előtt történt, ne csak magát az i-edik szakaszt! Ekkor az egyre növekvő (21.15) korlátot a

$$\delta_{i+1} = 3(\delta_i + \delta)^2 + \varepsilon$$

korlát helyettesíti. Most, ha $\delta_i < \delta$ és $12\delta^2 + \varepsilon < \delta$, akkor megint csak $\delta_{i+1} < \delta$, tehát a hibák nem halmozódnak. De irdatlan árat fizettünk: mire a számítás $(i+1)$-edik szakaszába értünk, a hibatűrő verzió nagysága 3-szorosa annak, ami az i-edik szakaszig volt. Ha t szakaszt akarunk ilyen módon hibatűrővé tenni, ez egy 3^t szorzóba kerül. Így a fent bevezetett $F(N, \delta)$ függvény N-ben exponenciálissá válhat.

Az alábbi tétel egy lehetséges szabatos verziója az itt elhangzott megfontolásoknak.

21.15. tétel. *Legyen R a Boole-függvények egy teljes, véges bázisa. Ha $2\varepsilon \le \delta \le 0.01$, akkor minden függvényt ki lehet számolni egy (ε, δ)-ellenálló hálózattal R fölött.*

Bizonyítás. Az egyszerűség kedvéért az eredményt csak olyan teljes bázisban bizonyítjuk, amely tartalmazza a háromváltozós többségi szavazást, és nem tartalmaz háromnál több változós függvényt. Azt is föltesszük, hogy a hibák függetlenek egymástól.

Legyen $\mathcal{N}$ egy t mélységű zaj nélküli hálózat, mely az f függvényt számolja ki. Bebizonyítjuk, f-et ki lehet számolni egy $2t$ mélységű (ε, δ)-ellenálló $\mathcal{N}'$ hálózattal. A bizonyítás t szerinti indukció. Az ε-ra és δ-ra vonatkozó elégséges feltételek menet közben fognak kiderülni.

Az állítás természetesen igaz $t = 1$-re, tegyük fel tehát, hogy $t > 1$. Legyen g az $\mathcal{N}$ hálózat kimeneti kapuja, akkor $f(\boldsymbol{x}) = g(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), f_3(\boldsymbol{x}))$. Az f_i függvényeket $\le t-1$ mélységű $\mathcal{N}_i$ részhálózatok számolják ki. Az induktív feltevés szerint az f_i függvényeket ki lehet számolni (ε, δ)-ellenálló, $\le 2t-2$ mélységű $\mathcal{N}_i'$ hálózatokkal. Legyen $\mathcal{M}$ az új hálózat, amely az $\mathcal{N}_i'$ hálózatok példányait tartalmazza (a megfelelő beviteli csúcsok összeragasztásával), és egy új csúcsot, amelyben a g kapu bemenetként kapja az $\mathcal{N}_i'$ hálózatok kimeneteit, és kiszámolja $f(\boldsymbol{x})$-et. Ekkor $\mathcal{M}$ hibavalószínűsége legfeljebb $3\delta + \varepsilon < 4\delta$, ha $\varepsilon < \delta$, mert mindhárom hálózat δ valószínűséggel hibázhat, és a g kaput tartalmazó csúcs $\le \varepsilon$ valószínűséggel tévedhet.

Végül állítsuk össze az $\mathcal{N}'$ hálózatot az $\mathcal{M}$ hálózat három egyforma példányából (összeragasztott bemenetekkel), és még egy csúcsból, amely a három kimenet között többségi szavazással dönt. Az $\mathcal{N}'$ hálózat hibavalószínűsége legfeljebb $3(4\delta)^2 + \varepsilon = 48\delta^2 + \varepsilon$. Valóban, a hibát vagy a többségi kapu tévedése okozza, vagy legalább ketten hibáztak az $\mathcal{M}$ hálózat három független példánya közül. Tehát a

$$48\delta^2 + \varepsilon \le \delta \tag{21.16}$$

feltétel mellett az $\mathcal{N}'$ hálózat (ε, δ)-ellenálló. A feltétel teljesül, ha $2\varepsilon \le \delta \le 0.01$. ■

A bizonyításban konstruált $\mathcal{N}'$ hálózat legalább 3^t-szer nagyobb, mint $\mathcal{N}$, a redundancia tehát irdatlanul nagy. Szerencsére, sokkal gazdaságosabb megoldásokat is fogunk látni. De vannak érdekes, kis mélységű hálózatok, amelyekre a 3^t szorzó nem extravagáns.

21.16. tétel. *Álljon teljes bázisunk az összes 3-változós Boole-függvényből. Ekkor minden elég kicsi $\varepsilon > 0$-ra, ha $2\varepsilon \le \delta \le 0.01$, akkor minden n-re van egy n-bemenetű, (ε, δ)-ellenálló logikai hálózat, melynek mélysége $\le 4\log(n+1)$ és nagysága $(n+1)^7$, és mely a bemenetek közelítő többségét számolja ki (a 21.9. definíció szerint).*

Bizonyítás. Alkalmazzuk a 21.15. tételt arra a hálózatra, melyet a 21.10. tétel (a) részéből kapunk. Ez egy új, $4\log(n+1)$-mély (ε,δ)-ellenálló hálózatot ad, mely közelítő többséget számol ki. Bármilyen ilyen hálózat, mely 3-bemenetű kapukból áll, legfeljebb $3^{4\log(n+1)} = (n+1)^{4\log 3} < (n+1)^7$ nagyságú. ■

Gyakorlatok

21.3-1. A 21.2-5. gyakorlat azt sugallja, hogy az $\mathcal{M}_3^r$ iterált többségi szavazás manipulálható. Bizonyos körülmények között azonban nagyon jól működik. Legyen az $\mathcal{M}_3^r$ bemenete a független Boole-értékű valószínűségi változók $\boldsymbol{X} = (X_1,\ldots,X_n)$ vektora, melyre $\Pr[\,X_i = 1\,] = p < 1/6$. Legyen Z a hálózat (véletlen) kimenet-bitje. Feltéve, hogy többségi kapuink egymástól függetlenül $\le \varepsilon \le p/2$ valószínűséggel tévedhetnek, bizonyítsuk be az alábbi becslést:

$$\Pr[\,Z = 1\,] \le \max\{10\varepsilon,\ 0.3(p/0.3)^{2^k}\}.$$

Útmutatás. Legyen $g(p) = \varepsilon + 3p^2$, $g_0(p) = p$, $g_{i+1}(p) = g(g_i(p))$, és bizonyítsuk a következőt: $\Pr[\,Z = 1\,] \le g_r(p)$.

21.3-2. Azt mondjuk, hogy az $\mathcal{N}$ hálózat az $f(x_1,\ldots,x_n)$ függvényt (ε,δ)-***bemenet-biztos*** módon számolja ki, ha a következő teljesül. Bármilyen $\boldsymbol{x} = (x_1,\ldots,x_n)$ bemenő vektorra, bármilyen ezt „perturbáló” független $\boldsymbol{X} = (X_1,\ldots,X_n)$ Boole-változó sorozatra, amely a $\Pr[\,X_i \ne x_i\,] \le \varepsilon$ követelménynek tesz eleget, az $\boldsymbol{X}$ bemenetű $\mathcal{N}$ hálózat Y kimenete keveset változik: $\Pr[\,Y = f(\boldsymbol{x})\,] \ge 1-\delta$. Mutassuk meg, hogy ha létezik olyan logikai hálózat, amely az $x_1 \oplus \cdots \oplus x_n$ függvényt $(\varepsilon, 1/4)$-bemenet-biztosan kiszámítja, akkor $\varepsilon \le 1/n$.

21.4. A részeredmények védelme

Ebben a fejezetben olyan hibatűrési módszereket ismerünk meg, amelyeknek viselkedése a rendszernagyság növelésekor kedvezőbb. Meg fogjuk mutatni a következőt:

21.17. tétel. *Léteznek olyan R_0, ε_0 konstansok, hogy az*

$$F(n,\delta) = N\log(n/\delta)$$

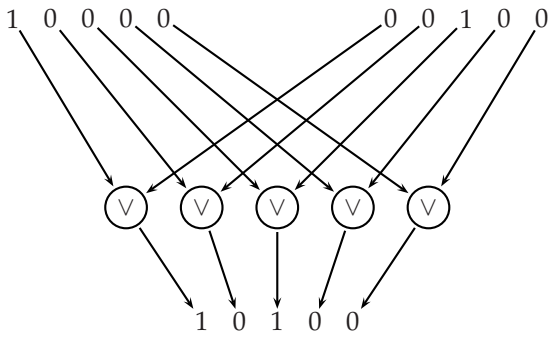
definícióval, minden $\varepsilon < \varepsilon_0$, $\delta \ge 3\varepsilon$ esetére, minden N nagyságú determinisztikus számításhoz van egy (ε,δ)-ellenálló, $R_0F(N,\delta)$ nagyságú számítás, amely ugyanazt az eredményt adja.

Bevezetünk egy fogalmat, amely egyszerűsíti hálózataink hiba-elemzését, függetlenítve azt a bemeneti $\boldsymbol{x}$ vektortól.

21.18. definíció. *Egy $\mathcal{N}$ logikai hálózat egy v csúcsában nevezzünk egy többségi kaput* ***javító többségi kapunak****, ha $\mathcal{N}$ minden $\boldsymbol{x}$ bemeneti vektorára, a v csúcs minden bemeneti élén ugyanaz az érték érkezik. Tekintsük az $\mathcal{N}$ hálózat egy számítását: ez a számítás egyes csúcsokat és vezetékeket* ***megfertőz.*** *A következő szabályok szerint terjed a fertőzés:*

– *A bemeneti csúcsok nem fertőzöttek.*

– *Ha egy csúcs fertőzött, akkor minden kimeneti vezetéke fertőzött.*



21.5. ábra. Végrehajtó szerv.

– *Egy javító többségi kaput tartalmazó csúcs akkor fertőzött, ha vagy téved, vagy bemeneteinek többsége fertőzött.*

– *Minden más csúcs akkor fertőzött, ha vagy téved, vagy valamelyik bemenete fertőzött.*

Nyilván, ha minden ε-megengedett véletlen konfigurációnál a hálózat kimenete $\leq \delta$ valószínűséggel fertőzött, akkor a hálózat (ε, δ)-ellenálló.

21.4.1. Kábelek

Egyelőre még csak a jelen fejezet bevezetésének (ii) ötletét használtuk: a számítási lépések ismétlését. Próbáljuk a (i) ötletet is használni logikai hálózatokban (az információt redundáns formában tartani).

Hogy kaputól kapuig védjük az áramló információt, a zaj nélküli hálózat minden vezetékét egy k vezetéket tartalmazó „kábellel” helyettesítjük (ahol k-t alkalmasan fogjuk választani). Egy kábel minden vezetékének ugyanazt a bit információt kellene szállítani, és azt reméljük, hogy többségük ezt a bitet viszi majd, még ha egyes vezetékek tévednek is.

21.19. definíció. *Egy $\mathcal{N}'$ logikai hálózatban, az élek egy bizonyos halmazát **kábelnek** hívhatjuk, ha a hálózat minden zajmentes számolásában, mindegyik él ugyanazt a Boole-értéket viszi. A halmaz elemszámát a kábel **vastagságának** nevezzük. Rögzítsünk egy megfelelő konstans ϑ küszöböt. Tekintsük az $\mathcal{N}'$ hálózat egy zajos verziójának bármilyen lehetséges számolását, és ebben egy k vastagságú kábelt. Ezt a kábelt **ϑ-biztonságosnak** nevezzük, ha legfeljebb ϑk él fertőzött benne.*

Vegyünk egy $\mathcal{N}$ hálózatot, amelyet ellenállóvá akarunk tenni. Amint az $\mathcal{N}$ vezetékeit az $\mathcal{N}'$ (egyenként k vezetéket tartalmazó) kábeleivel helyettesítünk, egy-egy v csúcsnál minden 2-bemenetelű zajnélküli kaput egy úgynevezett ***végrehajtó szerv*** nevű, k kapuból álló egységgel helyettesítjük (lásd a 21.5. ábrát). Ez, minden $i = 1, \ldots, k$-ra, az első és a második kábel i-edik vezetékét a végrehajtó szerv i-edik csúcsába vezeti. Mindezek a csúcsok ugyanazt a b_v típusú kaput tartalmazzák. Az ezekből a csúcsokból előbukkanó vezetékek adják a végrehajtó szerv ***kimeneti kábelét***.

21.6. ábra. Felújító szerv.

A kimeneti kábelben túl magasra nőhet a fertőzött vezetékek száma: valóban, ha az x vezetékben ϑk fertőzött vezeték volt, és az y vezetékben ugyancsak, akkor a $g(x,y)$ vezetékben már akár $2\vartheta k$ fertőzött vezeték is lehet (nem is számolva a végrehajtó szerv tévedései által hozzáadott új fertőzéseket). A konstrukció döntő része az, hogy a végrehajtó szervhez még egy úgynevezett ***felújító szervet*** is illesztünk: ennek az egységnek az a feladata, hogy a kábel fertőzöttségét csökkentse (lásd a 21.6. ábrát).

21.4.2. Sűrítők

Hogy készítsünk felújító szervet? Szem előtt tartva, hogy ennek a szervnek is zajban kell működni, építhetnénk (megfelelő δ'-re) egy speciális (ε,δ')-ellenálló hálózatot, amely k bemenetének közel-többségét számolja ki, k független példányban. A 21.16 tétel egy $k(k+1)^7$ nagyságú hálózatot szolgáltat erre.

Szerencsére jobb megoldás is van, legalábbis aszimptotikusan. *Nagyon egyszerű* felújító szervet fogunk keresni, olyat, amelynek a saját zaját már könnyű lesz elemezni. Mi egyszerűbb, mint egy olyan hálózat, melyben csak *egy lépés* van a bemenetek és kimenetek között? Rögzítsünk egy páratlan d egész számot (például, $d=3$). Szervünk minden kapuja egy d-bemenetű többségi kapu lesz.

21.20. definíció. *Nevezzünk* ***multigráfnak*** *minden olyan gráfot, amelyben minden pontpár között több él is futhat, nem csak 0 vagy 1. Egy páros multigráfot, melynek k bemenete és k kimenete van, nevezzünk d-**félregulárisnak,** ha minden kimeneti pont d fokú. Egy ilyen gráfot (d,α,γ,k)-**sűrítőnek** nevezünk, ha a következő tulajdonsággal bír: a bemenetek minden legfeljebb αk pontot tartalmazó E halmazához, legfeljebb $\leq \gamma\alpha k$ olyan kimenet van, amely az E legalább $d/2$ elemével van összekötve (multiplicitást is számolva).*

A sűrítő tulajdonság általában a $\gamma < 1$ esetben érdekes. Például egy $(5, 0.1, 0.5, k)$-sűrítőben a kimenetek foka 5, és a többségi szavazás ezekben a pontokban minden olyan hibahalmazt, amely legfeljebb a bemenetek 10%-át foglalja el, a kimenetek 5%-ára csökkent.

Egy megfelelő paraméterekkel bíró sűrítő működhetne felújító szervként: csökkentve a kisebbségben lévő eltéréseket, a kábel biztonságát helyreállíthatja. De vannak-e sűrítők?

21.21. tétel. *Minden* $0 < \gamma < 1$, *és páratlan egész* d*-re, ha*

$$1 < \gamma(d-1)/2, \tag{21.17}$$

akkor létezik egy $\alpha > 0$ *korlát, mellyel minden egész* $k > 0$*-ra vannak* (d, α, γ, k)*-sűrítők.*

Amint látjuk, a $d = 3$ esetre a tétel nem garantál sűrítőt $\gamma < 1$ értékkel.

Bizonyítás. Nem adunk explicit konstrukciót a keresett multigráfra, csak megmutatjuk, hogy létezik. Választunk egy véletlen d-félreguláris multigráfot (minden ilyen multigráfot ugyanolyan valószínűséggel), és megmutatjuk, hogy ez pozitív valószínűséggel (d, α, γ, k)-sűrítő lesz. Ezt a bizonyítási módszert ***valószínűségi módszernek*** nevezik. Legyen

$$s = \lfloor d/2 \rfloor .$$

Konstrukciónk kicsit általánosabb lesz, $k' \neq k$ számú kimenetet is megengedve. Képezzünk egy véletlen páros multigráfot k bemenettel és k' kimenettel, a következőképpen: minden kimenethez d élt húzunk véletlen bemeneti csúcsokból, amelyeket függetlenül és egyenletes eloszlással választunk az összes bemeneti csúcsok közül.

Legyen A egy αk nagyságú bemenethalmaz, legyen v egy kimeneti csúcs, és legyen E_v az esemény, hogy v-be legalább $s+1$ él vezet A-ból. Ekkor

$$\Pr(E_v) \leq \binom{d}{s+1}\alpha^{s+1} = \binom{d}{s}\alpha^{s+1} .$$

Jelöljük a jobb oldalt p-vel. Átlagban (várható értékben), az E_v esemény pk' különböző kimenetben következik be. Egy A bemenethalmazhoz legyen F_A az az esemény, hogy a v kimenetek száma, melyekben az E_v esemény bekövetkezik, több, mint $\gamma\alpha k'$. A (21.6) egyenlőtlenség alapján

$$\Pr(F_A) \leq \left(\frac{ep}{\gamma\alpha}\right)^{k'\gamma\alpha} .$$

Az összes lehetséges legfeljebb αk elemű A bemenethalmazok M számára a (21.7) egyenlőtlenség a következő becslést adja:

$$M \leq \sum_{i \leq \alpha k}\binom{k}{i} \leq \left(\frac{e}{\alpha}\right)^{\alpha k} .$$

Annak valószínűsége, hogy véletlen gráfunk nem sűrítő, legfeljebb akkora, mint annak valószínűsége, hogy az F_A esemény legalább egy A bemenethalmazra bekövetkezik. Ezt most így becsülhetjük:

$$M \cdot \Pr(F_A) \leq e^{-\alpha D k'} ,$$

ahol

$$D = -(\gamma s - k/k')\ln\alpha - \gamma(\ln\tbinom{d}{s} - \ln\gamma + 1) - k/k' .$$

Az α konstans csökkentésével ebben a kifejezésben az első tag dominál. Együtthatója pozitív, a (21.17) feltétel miatt. Tehát $D > 0$, ha teljesül az

$$\alpha < \exp\left(-\frac{\gamma(\ln\tbinom{d}{s} - \ln\gamma + 1) + k/k'}{\gamma s - k/k'}\right)$$

egyenlőtlenség. ■

21.4. példa. A $\gamma = 0.4$, $d = 7$, választással $\alpha = 10^{-7}$ megfelel.

Egy (d, α, γ, k)-sűrítőből úgy csinálunk egy $\mathcal{R}$ felújító szervet, hogy a kimenő csúcsaiba d-bemenetű többségi kapukat teszünk. Ha a kapuk néha tévednek, akkor $\mathcal{R}$ kimenete véletlen. Tegyük fel, hogy $\mathcal{R}$-nek legfeljebb αk bemenete fertőzött. Ekkor csak úgy lehet $(\gamma + \rho)\alpha k$ kimenet fertőzött, ha $\alpha\rho k$ többségi kapu hibázik. Legyen

$$p_{\mathcal{R}}$$

ennek az eseménynek a valószínűsége. Feltéve, hogy a kapuk $\mathcal{R}$-ben egymástól függetlenül $\leq \varepsilon$ valószínűséggel hibáznak, a (21.6) egyenlőtlenségről

$$p_{\mathcal{R}} \leq \left(\frac{e\varepsilon}{\alpha\rho}\right)^{\alpha\rho k} \tag{21.18}$$

következik.

21.5. példa. Válasszuk a következő értékeket: $\gamma = 0.4$, $d = 7$, $\alpha = 10^{-7}$, akárcsak a 21.4. példában, továbbá $\rho = 0.14$ (ez teljesíteni fogja a későbbiekben szükséges (21.19) egyenlőtlenséget). Ekkor az $\varepsilon = 10^{-9}$ tévedéskorláttal a $p_{\mathcal{R}} \leq e^{-10^{-8}k}$ korlátot kapjuk.

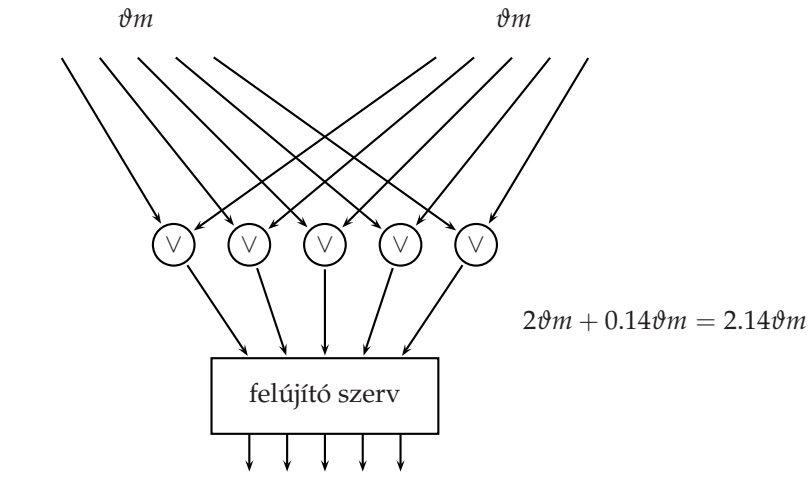
A vonzóan kicsi $d = 7$ fokszám sajnos kiábrándítóan gyenge korlátot ad csak annak valószínűségére, hogy a sűrítő kudarcot vall. Ez a korlát ugyan exponenciális sebességgel csökken a k kábelvastagság függvényében, de csak szélsőségesen nagy k esetén lesz tényleg kicsi.

21.6. példa. Megint $\gamma = 0.4$-et választva, de hozzá $d = 41$-et (tehát minden sűrítő kapu 41 vezeték többségét veszi 7 helyett), valamivel reálisabb eredményeket kapunk. Ez a választás lehetővé teszi az $\alpha = 0.15$ értéket. Legyen megint $\rho = 0.14$, $\varepsilon = 10^{-9}$, akkor $p_{\mathcal{R}} \leq e^{-0.32k}$ következik.

Ezek a számok kevésbé ijesztőek, de még mindig közel száz vezeték kell ahhoz, hogy a felújítási hiba valószínűsége kicsi legyen. És bár a gyakorlatban a számítógép-elemek sokkal kisebb, mint 10^{-9} gyakorisággal tévednek, az a kérdés is érdekelhet minket, mi a legnagyobb megtűrhető ε.

21.4.3. A biztonság terjesztése

Sűrítők segítségével olyan logikai hálózatot építhetünk, melynek minden kábele nagy valószínűséggel biztonságos.



21.7. ábra. Végrehajtó szervet felújító szerv követ.

21.22. definíció. *Egy adott $\mathcal{N}$ logikai hálózathoz, melynek (egyszerűség kedvéért) csak egyetlen bit a kimenete, egy k kábelnagysághoz és egy $\mathcal{R}$ logikai hálózathoz, melynek k bemenete és k kimenete van, legyen*

$$\mathcal{N}' = \mathrm{Cab}(\mathcal{N}, \mathcal{R})$$

a logikai hálózat, melyet a következőképpen kapunk. A bemenetek ugyanazok, mint $\mathcal{N}$-nek. Az $\mathcal{N}$ minden vezetékét egy k vastagságú kábellel helyettesítjük, és $\mathcal{N}$ minden kapuját helyettesítjük egy végrehajtó szervvel, amit egy olyan felújító szerv követ, mely az $\mathcal{R}$ hálózat másolata. Az új hálózatnak k kimenete van: ezek az $\mathcal{N}'$ utolsó felújító szervének kimeneteivel azonosak.

Zaj nélküli számításokban, minden bemenő Boole-vektorhoz, $\mathcal{N}'$ kimenete ugyanaz, mint $\mathcal{N}$-é, de k azonos példányban.

21.23. lemma. *Léteznek olyan $d, \varepsilon_0, \vartheta, \rho > 0$ konstansok, és minden k kábelvastagságra létezik egy $2k$ nagyságú $\mathcal{R}$ hálózat $\leq d$ bemenetszámú kapukkal, a következő tulajdonsággal. Minden $\mathcal{N}$ logikai hálózathoz, melynek kapunagysága 2 és nagysága N, minden $\varepsilon < \varepsilon_0$-ra, az $\mathcal{N}' = \mathrm{Cab}(\mathcal{N}, \mathcal{R})$ hálózat minden ε-megengedett konfigurációjára, annak a valószínűsége, hogy $\mathcal{N}'$-nek nem minden kábele ϑ-biztonságos, kisebb, mint $2N(\frac{e\varepsilon}{\vartheta\rho})^{\vartheta\rho k}$.*

Bizonyítás. Tudjuk, hogy található olyan d, α és $\gamma < 1/2$, melyre minden k-hoz létezik egy (d, α, γ, k)-sűrítő. Válasszuk ρ-t úgy, hogy a következő egyenlőtlenség teljesüljön:

$$\gamma(2 + \rho) + \rho \leq 1, \tag{21.19}$$

és legyen

$$\vartheta = \alpha/(2+\rho). \tag{21.20}$$

Készítsünk egy $\mathcal{R}$ felújító szervet egy (d, α, γ, k)-sűrítőből. Tekintsük az $\mathcal{N}$ hálózat egy v kapuját, és az $\mathcal{N}' = \mathrm{Cab}(\mathcal{N}, \mathcal{R})$ hálózat megfelelő végrehajtó és felújító szervét. Becsüljük meg annak az E_v eseménynek a valószínűségét, hogy ennek a kombinált szervnek bemenő kábelei ϑ-biztonságosak, de kimenő kábele nem. Tegyük fel, hogy a két bemenő kábel biztonságos: akkor a végrehajtó szervnek legfeljebb $2\vartheta k$ kimenete fertőzött a bemenő kábelek miatt: új fertőzés ezenkívül még új tévedések miatt is megjelenhet. Legyen E_{v1} az esemény, hogy a végrehajtó szerv legalább további $\rho\vartheta k$ kimenetet fertőz meg. Ekkor $\Pr(E_{v1}) \le (\frac{e\varepsilon}{\rho\vartheta})^{\rho\vartheta k}$, a (21.18) becslést használva. A végrehajtó szerv kimenetei a felújító szerv bemenetei. Ha ezekből legfeljebb $(2+\rho)\vartheta k = \alpha k$ fertőzött, akkor, amennyiben a felújító szerv tökéletesen működik, a fertőzött vezetékek mennyisége $\gamma(2+\rho)\vartheta k$-ra csökkenne. Legyen E_{v2} az esemény, hogy a felújító szerv legalább további $\rho\vartheta k$ vezetéket fertőz meg. Ekkor ugyanazt a becslést használva, $\Pr(E_{v2}) \le (\frac{e\varepsilon}{\rho\vartheta})^{\rho\vartheta k}$. Ha se E_{v1}, se E_{v2} nem következik be, akkor legfeljebb $\gamma(2+\rho)\vartheta k + \rho\vartheta k \le \vartheta k$ fertőzött vezeték bukkan elő a felújító szervből (lásd (21.19)-t), tehát a kimenő kábel biztonságos. Tehát $E_v \subset E_{v1} \cup E_{v2}$, és így $\Pr(E_v) \le 2(\frac{e\varepsilon}{\rho\vartheta})^{\rho\vartheta k}$.

Legyenek $V = \{1, \dots, N\}$ az $\mathcal{N}$ hálózat csúcsai. Mivel az egész $\mathcal{N}'$ hálózat bemenő kábelei biztonságosak, azt az eseményt, hogy található egy nem biztonságos kábel, az $E_1 \cup E_2 \cup \cdots \cup E_N$ esemény tartalmazza: tehát valószínűsége legfeljebb $2N(\frac{e\varepsilon}{\rho\vartheta})^{\rho\vartheta k}$. ■

21.4.4. Végjáték

A 21.17. tétel bizonyítása. Csak arra az esetre bizonyítjuk a tételt, amikor a számítás egy egyetlen bit kimenetű logikai hálózat. Az általánosítás több bitre egyszerű. A 21.23. lemma olyan $\mathcal{N}'$ hálózatot ad, melynek kimeneti kábele biztonságos, kivéve egy legfeljebb $2N(\frac{e\varepsilon}{\rho\vartheta})^{\rho\vartheta k}$ valószínűségű eseményt. Válasszuk k-t úgy, hogy ez $\le \delta/3$ legyen:

$$k \ge \frac{\lg(6N/\delta)}{\rho\vartheta \lg \frac{\rho\vartheta}{e\varepsilon_0}} \,. \tag{21.21}$$

Már csak az van hátra, hogy ehhez a kimeneti kábelhez egy kis hálózatot illesszünk, mely a többségi értéket megbízhatóan előhozza belőle. Ez megtehető a 21.16. tétel segítségével, amely egy $(k+1)^7$ nagyságú úgynevezett „kóda” hálózatot ad $\mathcal{N}'$-hez. Nevezzük a kapott hálózatot $\mathcal{N}''$-nek.

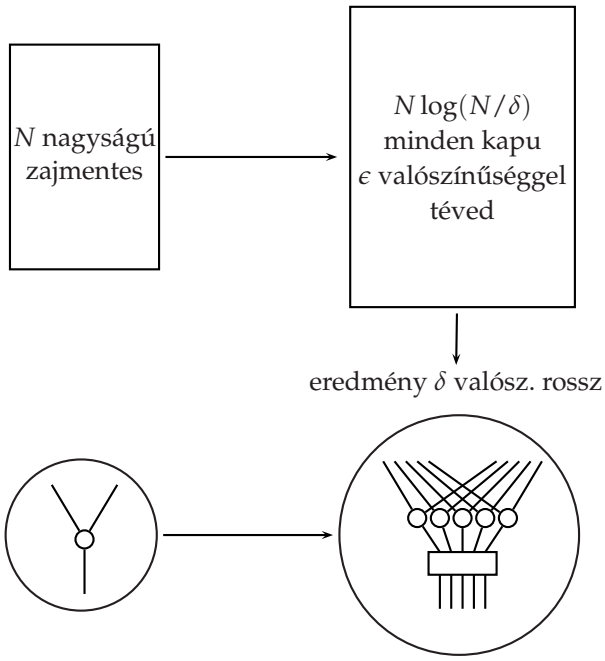
Annak valószínűsége, hogy a kimeneti kábel nem biztonságos, $< \delta/3$. Annak valószínűsége, hogy a kimeneti kábel biztonságos, de a „kóda” hálózat téved, kisebb, mint 2ε. Tehát annak valószínűsége, hogy $\mathcal{N}''$ hibázik, legfeljebb $2\varepsilon + \delta/3 \le \delta$, használva a $\delta \ge 3\varepsilon$ feltételt.

Becsüljük meg $\mathcal{N}''$ nagyságát. A (21.21) egyenlőtlenség szerint választhatunk egy $k = O(\lg(N/\delta))$ kábelvastagságot. Mivel $|\mathcal{N}'| \le 2kN$, ezért az

$$|\mathcal{N}''| \le 2kN + (k+1)^7 = O(N \lg(N/\delta))$$

felső korlátot kapjuk. ■

21.7. példa. Vegyük a 21.6. példa konstansait, és ϑ-t (21.20)-ból: akkor $\varepsilon_0 = 10^{-9}$, $d = 41$, $\gamma = 0.4$,



21.8. ábra. Megbízható hálózat, egy zaj nélküli hálózatból.

$\rho = 0.14$, $\alpha = 0.15$, $\vartheta = 0.07$, tehát

$$\frac{1}{\rho\vartheta \ln \frac{\rho\vartheta}{e\varepsilon_0}} \approx 6.75.$$

Ha most k-t a lehető legkisebbre választjuk, akkor $k \approx 6.75 \ln(N/\delta)$. Ha $\delta = 10^{-8}$, $N = 10^{12}$, akkor ez a $k = 323$ kábelvastagságot engedi meg. Ráadásul ehhez az igazán kellemetlen kábelvastagsághoz, a „kóda” hálózat nagysága $(k+1)^7 \approx 4 \cdot 10^{17}$, ami az $\mathcal{N}''$ hálózat egészét dominálja (noha $N \to \infty$ esetén aszimptotikusan elhanyagolható).

Amint a 21.7. példa mutatja, a redundanciának a fenti bizonyításból kiszámolható ára a gyakorlatban elfogadhatatlan. Az $O(\lg(N/\delta))$ faktor jól hangzik, mert csak logaritmikus a számítás nagyságához képest, és egy elég nagy többségi kaput választva (41 bemenet), a 6.75 szorzó az $O(\cdot)$-ban ugyancsak nem mutat rosszul; de mégse várnánk, hogy a megbízhatóság ára ilyen nagy legyen.

Mennyire javítható ez a redundancia optimalizálással, vagy más módszerekkel? A 21-6. gyakorlat azt mutatja, hogy egy valamivel szigorúbb hibamodellben (a hibák függetlenek és azonos valószínűségűek), több véletlenítéssel, valamivel jobb konstansok érhetők el. A 21.4-1., 21.4-2. és 21.4-6. gyakorlatok a „kóda” hálózatot javítgatták. De ezeknek a javításoknak egyike se hozza a redundanciát elfogadható szintre. Még ha eltekintünk is a véletlen választással járó kellemetlenségektől (ezen valamennyire lehet segíteni), maguk a koncentrátorok nagyok és ügyetlenek. A baj valószínűleg azzal van, hogy kiinduló modellnek logikai hálózatokat választottunk. Egy általános logikai hálózatot nem lehet természetes módon nem-konstants nagyságú részegységekre bontani, és így a megbízhatósági problémát

modulárisan kezelni.

21.4.5. Sűrítők konstrukciója

Ez az alfejezet az előzőknél vázlatosabb, és némi lineáris algebrai tudást feltételez.

Megmutattuk, hogy sűrítők léteznek. Milyen költséges egy (d, α, γ, k)-sűrítőt találni, mondjuk a $d = 41$, $\alpha = 0.15$, $\gamma = 0.4$ paraméterekkel, mint a 21.6. példában? Deterministikus algoritmust használva, végigkereshetnénk a körülbelül d^k páros d-félreguláris gráfot. Ezek mindegyikében végigpróbálhatnánk az összes $\leq \alpha k$ nagyságú bemeneti halmazt: mint tudjuk, ezek száma $\leq (e/\alpha)^{\alpha k} < 2^k$. Minden részhalmaz ellenőrzésének költsége $O(k)$, tehát a műveletek teljes száma $O(k(2d)^k)$. Bár ez a szám exponenciális k-ban, emlékezzünk rá, hogy hibajavító konstrukciónkban $k = O(\log(N/\delta))$, ahol N a zajmentes hálózat nagysága: a sűrítőkeresés teljes műveletszáma tehát N-ben polinomiális.

A 21.21. tétel bizonyítása mutatja, hogy egy véletlenül választott d-félreguláris páros gráf nagy valószínűséggel sűrítő. Van tehát egy gyorsabb, randomizált algoritmus sűrítő generálására. Válassz egy véletlen páros gráfot, ellenőrizd, sűrítő-e: ha nem, kezdd elölről. Átlagban konstans sok ismétlés után megállhatunk. Ez az algoritmus gyorsabb, de még mindig exponenciális k-ban, mert minden ellenőrzés $\Omega(k(e/\alpha)^{\alpha k})$ műveletbe kerül.

Van-e explicit konstrukció sűrítőre, k-ban exponenciális keresés elkerülésével? A válasz igenlő. De ebben a fejezetben csak azt mutatjuk meg, hogy a sűrítő tulajdonság egy bizonyos lineáris algebrai tulajdonságból következik, amit polinomiális időben ellenőrizni lehet. Ismeretesek explicit módon megadott gráfok, melyek ezzel a tulajdonsággal rendelkeznek. Leginkább ezeket nem sűrítő, hanem *tágító* tulajdonságuk miatt keresik (lásd a 21.4-3. gyakorlatot).

Ha $\boldsymbol{v}, \boldsymbol{w}$ vektorok, akkor legyen $(\boldsymbol{v}, \boldsymbol{w})$ a skaláris szorzatuk. Egy $2k$ csúcsú d-félreguláris páros multigráf egy $\boldsymbol{M} = (m_{ij})$, ***incidencia mátrixszal*** definiálható, melyben m_{ij} azon élek száma, melyek a j bemenetet az i kimenethez kötik. Legyen $\boldsymbol{e}$ a csupa egyes $(1, 1, \ldots, 1)^T$ vektor. Ekkor $\boldsymbol{Me} = d\boldsymbol{e}$, tehát $\boldsymbol{e}$ ***sajátvektora*** az $\boldsymbol{M}$ mátrixnak, a d ***sajátértékkel***. Sőt, d az $\boldsymbol{M}$ legnagyobb sajátértéke. Valóban, a $|\boldsymbol{x}|_1 = \sum_i |x_i|$ jelöléssel, minden $\boldsymbol{x} = (x_1, \ldots, x_k)$ sorvektorra $|\boldsymbol{xM}|_1 \leq |\boldsymbol{x}|_1$.

21.24. tétel. *Legyen G az $\boldsymbol{M}$ mátrix által definiált multigráf. Minden $\gamma > 0$ és*

$$\mu < d\sqrt{\gamma}/2 \tag{21.22}$$

értékre létezik olyan $\alpha > 0$, hogy ha az $\boldsymbol{M}^T\boldsymbol{M}$ mátrix második legnagyobb sajátértéke μ^2, akkor G egy (d, α, γ, k)-sűrítő.

Bizonyítás. Az $\boldsymbol{M}^T\boldsymbol{M}$ mátrix legnagyobb sajátértéke d^2. Mivel szimmetrikus, van ortogonális egység hosszúságú $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k$ sajátvektorokból álló bázisa, a

$$\lambda_1^2 \geq \cdots \geq \lambda_k^2$$

sajátértékekkel, ahol $\lambda_1 = d$, $\boldsymbol{e}_1 = \boldsymbol{e}/\sqrt{k}$.

Emlékezzünk, hogy az $\{\boldsymbol{e}_i\}$ ortonormális bázisban minden $\boldsymbol{f}$ vektort az $\boldsymbol{f} = \sum_i (\boldsymbol{f}, \boldsymbol{e}_i)\boldsymbol{e}_i$ módon fejezhetünk ki. Tetszőleges $\boldsymbol{f}$ vektorra, az $|\boldsymbol{Mf}|^2$ értéket a következőképpen becsül-

hetjük.

$$|\boldsymbol{M}\boldsymbol{f}|^2 = (\boldsymbol{M}\boldsymbol{f}, \boldsymbol{M}\boldsymbol{f}) = (\boldsymbol{f}, \boldsymbol{M}^T\boldsymbol{M}\boldsymbol{f}) = \sum_i \lambda_i^2 (\boldsymbol{f}, \boldsymbol{e}_i)^2$$
$$\leq d^2(\boldsymbol{f}, \boldsymbol{e}_1)^2 + \mu^2 \sum_{i>1} (\boldsymbol{f}, \boldsymbol{e}_i)^2 \leq d^2(\boldsymbol{f}, \boldsymbol{e}_1)^2 + \mu^2(\boldsymbol{f}, \boldsymbol{f})$$
$$= d^2(\boldsymbol{f}, \boldsymbol{e})^2/k + \mu^2(\boldsymbol{f}, \boldsymbol{f}).$$

Legyen most $A \subset \{1, \ldots, k\}$ egy αk nagyságú halmaz, és legyen $\boldsymbol{f} = (f_1, \ldots, f_k)^T$, ahol $f_j = 1$, ha $j \in A$, és 0 különben. Ekkor a $\boldsymbol{M}\boldsymbol{f}$ vektor i-edik koordinátája azon élek d_i számát adja, melyek az A halmazból az i csúcsba érkeznek. Továbbá, $(\boldsymbol{f}, \boldsymbol{e}) = (\boldsymbol{f}, \boldsymbol{f}) = |A|$, az A elemszáma. Azt kapjuk, hogy

$$\sum_i d_i^2 = |\boldsymbol{M}\boldsymbol{f}|^2 \leq d^2(\boldsymbol{f}, \boldsymbol{e})^2/k + \mu^2(\boldsymbol{f}, \boldsymbol{f}) = d^2\alpha^2 k + \mu^2 \alpha k \ ,$$
$$k^{-1} \sum_i (d_i/d)^2 \leq \alpha^2 + (\mu/d)^2 \alpha \ .$$

Tegyük fel, hogy $c\alpha k$ olyan i csúcs van, melyre $d_i > d/2$, akkor ebből

$$c\alpha \leq 4(\mu/d)^2\alpha + 4\alpha^2$$

következik. Ilyen módon, mivel (21.22) miatt $4(\mu/d)^2 < \gamma$, ha α elég kicsi, akkor $\mathcal{M}$ egy (d, α, γ, k)-sűrítő. ■

A (21.22) feltétel enyhíthető. Valójában elegendő olyan gráfokat keresni, nagy k-ra, melyekben $\mu/d < c < 1$, ahol d, c konstansok. Ehhez definiáljuk két $2k$ elemű páros multigráf szorzatát a megfelelő mátrixok szorzatával.

Tegyük fel, $\boldsymbol{M}$ szimmetrikus: akkor második legnagyobb sajátértéke μ, és $\boldsymbol{M}^r$ két legnagyobb sajátértékének aránya $(\mu/d)^r$. Tehát elég nagy r-re a $\boldsymbol{M}^r$ mátrix ki fogja elégíteni a (21.22) feltételt. Sajnos a hatványozás a d fokszámot megnöveli, valószínűleg még inkább eltávolítva minket a gyakorlati megvalósíthatóságtól.

Azt találtuk, hogy létezik konstrukció egy kívánt paraméterekkel rendelkező sűrítőre, ha csak találunk tetszőleges nagy $2k$ nagyságú multigráfokat, szimmetrikus $\boldsymbol{M}_k$ mátrixszal, melyekben a két legnagyobb sajátérték aránya egy k-tól független $c < 1$ konstans alatt van. Ilyen multigráfokra különböző konstrukciók léteznek (a történeti visszatekintésben adunk néhány utalást ezekre). A sajátérték-arány becslése egyik esetben se nagyon egyszerű.

Gyakorlatok

21.4-1. A 21.17. tétel bizonyítása egy $(k+1)^7$ nagyságú „kóda” hálózatot használ. Miután bebizonyítottuk, természetesen ezt a tételt a befejező többségi érték kiszámítására is használhatjuk: ez a „kóda” hálózat nagyságát $O(k \lg k)$-ra csökkentené. Próbáljuk ki ezt a fent használt numerikus példákon, hogy lássuk, lényeges javuláshoz vezet-e.

21.4-2. A 21.21 tétel bizonyítása olyan k-bemenetű, sűrítő-tulajdonságú páros gráfokat is nyújt, melyeknek csak $k' < 0.8k$ kimenete van. A 21.17 tétel bizonyításában szereplő „kóda” hálózat talán csökkenthető több ilyen sűrítő egymás után kapcsolásával. Próbáljuk ezt ki, annak szem előtt tartásával, hogy k csökkenésekor a (21.18) egyenlőtlenség „exponenciális”

hibabecslése gyengül.

21.4-3. Egy d-félreguláris páros multigráfot, melyben a k bemenet halmaza A és a k kimenet halmaza B akkor nevezünk (d,α,λ,k)-***tágítónak (expandernek),*** ha a következő tulajdonsággal bír: minden $E \subset A$ halmazra, ha $|E| \le \alpha k$, akkor B-nek legalább $\lambda\alpha k$ eleme van E-vel összekötve. Bizonyítsuk a következő tételt, mely a 21.21 tétel analógja: Minden $\lambda < d$ esetén létezik egy olyan α, hogy minden $k > 0$-ra van (d,α,λ,k)-tágító. *Útmutatás.* A 21.21 tétel bizonyításához hasonlóan mutassuk meg, hogy egy véletlen d-félreguláris multigráf nagy valószínűséggel tágító.

21.4-4. Egy zajos logikai hálózatban legyen $F_v = 1$, ha a v csúcs kapuja téved, és 0 különben. Továbbá legyen $T_v = 1$, ha v fertőzött, és 0 különben. Tegyük fel, hogy az F_v valószínűségi változók eloszlása nem függ a bemeneti Boole-vektortól. Mutassuk meg, hogy akkor a T_v valószínűségi változók együttes eloszlása is független a bemenetvektortól.

21.4-5. Ez a gyakorlat a 21.3-1. gyakorlat eredményét terjeszti ki véletlen bemenetvektorokra: megmutatja, hogy ha egy véletlen bemenetvektorban csak kevés hiba van, akkor a 21.2-5. gyakorlat $\mathcal{M}_3^r$ iterált többségi szavazása még mindig működhet rajta, amennyiben a bemeneti vezetékeket véletlenül átrendezzük. Legyen $k = 3^r$, és legyen $\boldsymbol{j} = (j_1,\ldots,j_k)$ egy $j_i \in \{1,\ldots,k\}$ egész számokból álló vektor. A $C(\boldsymbol{j})$ logikai hálózatot a következőképpen definiáljuk. Ez a hálózat veszi az $\boldsymbol{x} = (x_1,\ldots,x_k)$ bemenetvektort, kiszámolja az $\boldsymbol{y} = (y_1,\ldots,y_k)$ vektort, ahol $y_i = x_{j_i}$ (más szóval, egyszerűen egy vezetéket visz a j_i bemenetcsúcstól a „közbülső” i csúcshoz), majd beadja $\boldsymbol{y}$-t az $\mathcal{M}_3^r$ hálózatba.

Jelöljük $C(\boldsymbol{j})$ (esetleg véletlen) kimenetbitjét Z-vel. Minden rögzített $\boldsymbol{x}$ bemenetvektorhoz, feltéve, hogy többségi kapuink egymástól függetlenül $\le \varepsilon \le \alpha/2$ valószínűséggel tévednek, legyen $q(\boldsymbol{j},\boldsymbol{x}) := \Pr[\,Z = 1\,]$. Tegyük fel, hogy a bemenet a (nem feltétlenül független) Boole valószínűségi változók egy $\boldsymbol{X} = (X_1,\ldots,X_k)$ vektora, melyre $p(\boldsymbol{x}) := \Pr[\,\boldsymbol{X} = \boldsymbol{x}\,]$. Az $|\boldsymbol{X}| = \sum_i X_i$ jelöléssel, tegyük fel, hogy $\Pr[\,|\boldsymbol{X}| > \alpha k\,] \le \rho < 1$. Bizonyítsuk, hogy a $\boldsymbol{j}$ vektornak létezik olyan választása, melyre

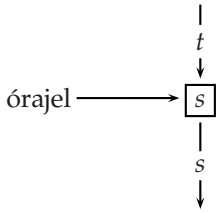
$$\sum_{\boldsymbol{x}} p(\boldsymbol{x})q(\boldsymbol{j},\boldsymbol{x}) \le \rho + \max\{10\varepsilon,\ 0.3(\alpha/0.3)^{2^k}\}.$$

Ez a választás függhet az $\boldsymbol{X}$ véletlen vektor eloszlásától. *Útmutatás.* Válasszuk a $\boldsymbol{j}$ vektort (és ezzel a $C(\boldsymbol{j})$ hálózatot) véletlenül, azaz mint egy $\boldsymbol{J} = (J_1,\ldots,J_k)$ véletlen vektort, ahol a J_i valószínűségi változók függetlenek, és egyenletes eloszlásúak $\{1,\ldots,k\}$ felett, és legyen $s(\boldsymbol{j}) := \Pr[\,\boldsymbol{J} = \boldsymbol{j}\,]$. Bizonyítsuk a következőt:

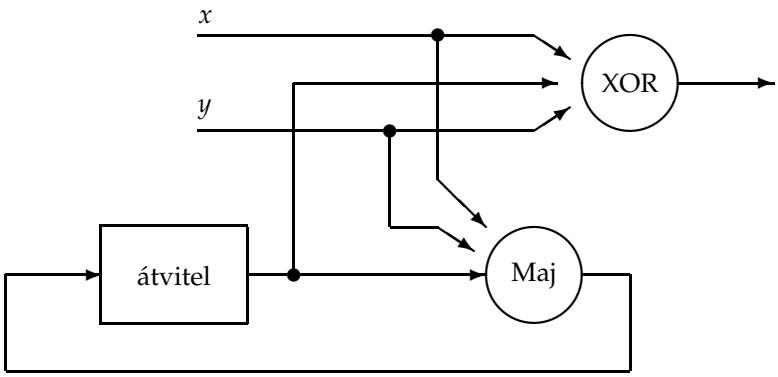
$$\sum_{\boldsymbol{j}} s(\boldsymbol{j}) \sum_{\boldsymbol{x}} p(\boldsymbol{x})q(\boldsymbol{j},\boldsymbol{x}) \le \rho + \max\{10\varepsilon,\ 0.3(\alpha/0.3)^{2^k}\}.$$

Ehhez, cseréljük fel a $\boldsymbol{x}$ és $\boldsymbol{j}$ szerinti átlagolást, majd vegyük észre, hogy $\sum_{\boldsymbol{j}} s(\boldsymbol{j})q(\boldsymbol{j},\boldsymbol{x})$ a $Z = 1$ valószínűsége, ha a J_i “drótokat” véletlenül, “menetközben” választjuk a hálózat számolása során.

21.4-6. A 21.4-4. gyakorlat jelölésével tegyük fel, mint ott, hogy az F_v valószínűségi változók eloszlása nem függ a bemeneti Boole-vektortól. Vegyük a 21.22. definícióban bevezetett $\mathrm{Cab}(\mathcal{N},\mathcal{R})$ logikai hálózatot, és definiáljuk a $\boldsymbol{T} = (T_1,\ldots,T_k)$ véletlen Boole-vektort, melyben $T_i = 1$, ha az i-edik kimeneti csúcs fertőzött. Alkalmazzuk a 21.4-5. gyakorlatot annak megmutatására, hogy létezik egy $C(\boldsymbol{j})$ hálózat, melyet a kimenő csúcsokhoz illeszthetünk, és amely a „kóda” hálózat szerepét játszhatja a 21.17 tétel bizonyításában. A $C(\boldsymbol{j})$ nagysága csak lineáris k-ban, nem pedig $(k+1)^7$, mint abban a bizonyításban. De a hibák eloszlásáról kicsit többet tettünk fel, ezenkívül a $\boldsymbol{j}$ „drótozás” függ a $\mathrm{Cab}(\mathcal{N},\mathcal{R})$ hálózattól.



21.9. ábra. Bit-tároló elem



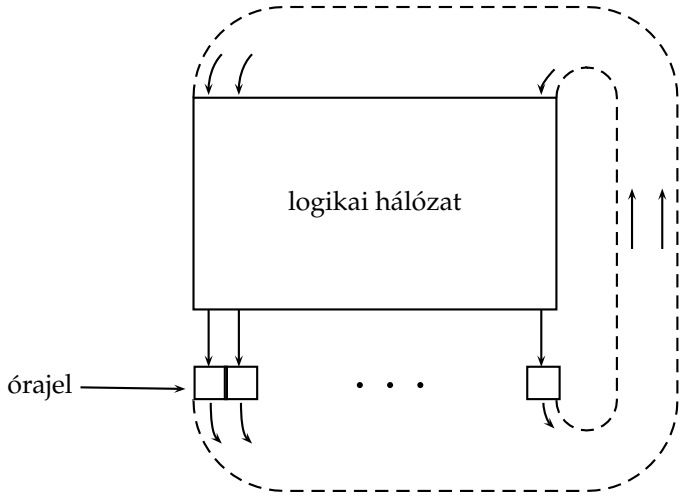
21.10. ábra. Egy hálózat része, mely két bináris szám, – x és y – összegét számítja ki. Az x és y számjegyeit a legkisebb helyi értéktől kezdve tápláljuk be a bemeneteken. Az összeg számjegyei a kimeneti élen bukkannak elő. Egy bit-tároló őrzi az átviteli számjegyet.

21.5. A megbízható információtárolás problémája

21.5.1. Ütemezett hálózatok

A közönséges számolásoknak egy eleme feltűnően hiányzik a fent leírt logikai hálózat modellből: az *ismétlések*. Egyes műveleteket megismételni csak akkor lehet, ha a számoló egységek munkáját időzítjük, és az egymást követő lépések között a részeredményeket *tároljuk*. Pillantsunk megint a hálózattervező rajzaira: olyan egységeket is látni fogunk, mint a 21.9. ábrán. Ezeknek egy bemenő éle van, és nincs logikai művelet hozzájuk rendelve; ***bit-tárolóknak*** fogjuk őket hívni. A bit-tárolót egy központi órajel-generátor vezéreli (ez nem látható az ábrán). Minden órajelre a bemenő élen érkező logikai érték átugrik a kimenő élekre, és a „tárolóban marad”. A 21.10. ábra mutatja bit-tárolók lehetséges felhasználását egy hálózatban.

21.25. definíció. *Egy **ütemezett hálózatot** a Q teljes bázis felett formálisan a logikai háló-*



21.11. ábra. Egy "számítógép" memóriából (bit-tárolókból) áll, és egy azt vezérlő logikai hálózatból. Egy ***számítás nagyságát*** a számítógép-nagyság és a lépésszám szorzatával definiálhatjuk.

zatokhoz hasonlóan adunk meg (lásd (21.10)). A hálózat ugyancsak hasonlóan definiál egy $G = (V, E)$ gráfot is. Emlékezzünk, hogy a csúcsokat az $1, \ldots, N$ természetes számokkal azonosítottuk. Minden v nem-bemeneti csúcshoz vagy egy b_v kaput rendelünk, mint azelőtt, vagy egy ***bit-tárolót:*** *ez esetben $k_v = 1$ (csak egy „argumentum" van). Nem kívánjuk, hogy a gráf aciklikus legyen, de megkívánjuk, hogy minden irányított ciklus (ha van ilyen) legalább egy bit-tárolón haladjon át.*

A hálózat működését a $t = 0, 1, 2, \ldots$-el jelzett ***óraciklusok*** *sorozatára bonthatjuk fel. A t-edik óraciklus bemenet-vektorát jelöljük $\boldsymbol{x}^t = (x_1^t, \ldots, x_n^t)$-vel, a bit-tárolók állapotát $\boldsymbol{s}^t = (s_1^t, \ldots, s_k^t)$-vel, és a kimenet-vektort $\boldsymbol{y}^t = (y_1^t, \ldots, y_m^t)$-vel.*

A hálózatnak az a része, mely a bemenetektől a bit-tárolókhoz megy, két Boole-vektorfüggvényt definiál: $\lambda : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$ és $\tau : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^k$. Az ütemezett hálózat működését a következő egyenletek írják le (lásd a 21.11. ábrát, mely nem mutatja a bemeneteket és kimeneteket).

$$\boldsymbol{y}^t = \lambda(\boldsymbol{s}^t, \boldsymbol{x}^t), \quad \boldsymbol{s}^{t+1} = \tau(\boldsymbol{s}^t, \boldsymbol{x}^t). \tag{21.23}$$

Gyakran, a hálózat számolása folyamán nincsenek kimenetek és bemenetek, ezért a (21.23) egyenleteket így egyszerűsíthetjük:

$$\boldsymbol{s}^{t+1} = \tau(\boldsymbol{s}^t). \tag{21.24}$$

Hogyan használjunk egy ilyen egyenlettel leírt ütemezett hálózatot számolásra? Valamilyen kezdeti értékeket írunk a bit-tárolókba, majd a logikai értékek hozzárendelését továbbvisszük a logikai kapukat használva, az adott óracikluson belül. Ezután küldünk egy órajelet a memóriába (a bit-tárolóknak), ez erre új értékeket ír kimeneti éleire (melyek azonosak a

hálózat bemeneti éleivel). Ezután kiszámoljuk az új hozzárendelést, és így tovább.

Hogyan számolhatunk ki egy $f(x)$ *függvényt* egy ilyen hálózat segítségével? Itt egy lehetséges konvenció. Beírjuk az x bemenetet (csak az első lépésben), aztán járatjuk a hálózatot, amíg csak egy extra kimenő élen nem jelzi, hogy a többi kimenő él tartalmazza a várt $f(x)$ eredményt.

21.8. példa. Ez a példa a fentitől eltérő konvenciót használ: a hálózat minden lépésben új bemenő biteket kap, és az eredményt folyamatosan szolgáltatja. A 21.10. ábra bináris összeadójában legyen u^t és v^t a két bemenő bit a t-edik ciklusban, legyen c^t az átvitel, és w^t a kimenet ugyanabban a ciklusban. A (21.23) egyenletek most a következő formát kapják:

$$w^t = u^t \oplus v^t \oplus c^t, \quad c^{t+1} = \mathrm{Maj}(u^t, v^t, c^t),$$

ahol Maj a többségi szavazás művelete.

21.5.2. Információtárolás

Az ütemezett hálózat érdekes párhuzamos számítógép modell, de adjunk most csak egy olyan feladatot neki, amely triviális a zajtalan esetben: információtárolást. Bizonyos mennyiségű információt szeretnénk tárolni olyan módon, hogy egy idő után elő tudjuk venni, annak ellenére, hogy a hálózatban tévedések fordulnak elő. Ebben az esetben a fent bevezetett τ átmenetfüggvény nem lehet egyszerűen az identitás: *hibajavító* műveleteket kell végrehajtania. Természetesen adódik, hogy az előző alfejezetben tárgyalt *felújító szerveket* használjuk. Tegyük fel, hogy k memóriacellát (bit-tárolót) szentelünk egy bit információ tárolására. Nevezzük ezt a k-ast ***biztonságosnak,*** ha a helyes értéktől eltérő memóriacellák száma valamilyen ϑk küszöb alatt van.

Legyen a hálózat maradék része egy (d, α, γ, k)-sűrítőre épített felújító szerv, melyben $\alpha = 0.9\vartheta$. Tegyük fel, hogy a bemenő kábel biztonságos. Ekkor annak valószínűsége, hogy az átmenet után a kimenő kábel (és így az új állapot) nem biztonságos, $O(e^{-ck})$ lesz valami c konstanssal. Tegyük fel, hogy a hálózatot t lépésen át működtetjük. Ekkor annak valószínűsége, hogy az állapot nem biztonságos ezen lépések valamelyikében, $O(te^{-ck})$. Ez kicsi, amennyiben t lényegesen kisebb, mint e^{ck}. Ha m bit információt tárolunk, akkor annak valószínűsége, hogy ezen bitek bármelyike valamelyik lépésben elveszti biztonságát, $O(mte^{-cm})$.

Ha szigorúvá akarjuk tenni tárgyalásunkat, az ütemezett hálózatokra is hibamodellt kell bevezetni. Mivel csak olyan egyszerű τ átmenetfüggvényeket fogunk vizsgálni, melyekben csak egyetlen számolási lépés történik a t és $t+1$ időpontok között (akárcsak a fenti többségi szavazásban), modellünk is egyszerű lesz.

21.26. definíció. *Tekintsünk egy ütemezett hálózatot, melyet a (21.24) egyenlet ad meg: állapotát ekkor minden $t = 0, 1, 2, \ldots$ időpontban az $\boldsymbol{s}^t = (s_1^t, \ldots, s_n^t)$ bitvektor írja le. Legyen $\boldsymbol{Y}^t = (Y_1^t, \ldots, Y_n^t)$ a véletlen bitvektorok sorozata $t = 0, 1, 2, \ldots$-re. A (21.13) egyenlethez hasonlóan legyen*

$$Z_{i,t} = \tau(\boldsymbol{Y}^{t-1}) \oplus Y_i^t. \tag{21.25}$$

Ekkor $Z_{i,t} = 1$ azt jelenti, hogy az (i, t) téridő pontban ***tévedés*** *történik. Az $\{\boldsymbol{Y}^t\}$ sorozatot ε-**megengedettnek** nevezzük, ha a (21.14) egyenlőtlenség áll a $t = 0$ után bekövetkező téridő pontok minden véges C halmazára.*

Az imént említett felújító-szerves konstrukcióval m bit információt T lépésen át lehet tartani, ha

$$O(m \lg(mT)) \tag{21.26}$$

memóriacellát használunk. Pontosabban, az $\boldsymbol{Y}^T$ kábel nagy valószínűséggel biztonságos lesz minden megengedett $\boldsymbol{Y}^t$ $(t = 0, \ldots, T)$ evolúcióban. Lehet ezen javítani?

A megbízható információtárolás feladata rokon az ***információ-továbbítás*** feladatával: egy ***feladó*** egy $\boldsymbol{x}$ *üzenetet* át akar juttatni egy ***zajos csatornán*** egy ***címzettnek.*** Csak éppen itt feladó és címzett ugyanaz a személy, és a zajos csatorna csak az idő folyása. Most ezért először bevezetjük a megbízható információátvitel néhány alapfogalmát, azután pedig alkalmazzuk ezeket egy adattároló rendszer készítésére, mely gazdaságosabb, mint a most látott naiv ismétléses megoldás.

21.5.3. Hibajavító kódok

Hibafelismerés

Információvédelem céljára az ismétléses módszernél hatékonyabban is használhatunk redundanciát. Még azt is megpróbálhatjuk, hogy az üzenethez csak egyetlen további bitet teszünk hozzá. Legyen $\boldsymbol{x} = (x_1, \ldots, x_6)$, $(x_i \in \{0, 1\})$ a védeni kívánt szó. Képezzük az

$$x_7 = x_1 \oplus \cdots \oplus x_6$$

hibajavító bitet. Például, $\boldsymbol{x} = 110010$, $\boldsymbol{x}' = 1100101$. Ha $\boldsymbol{x}' = (x_1, \ldots, x_7)$ ***kódszavunkat*** zajnak tesszük ki, egy új szóvá, $\boldsymbol{y}$-ná változik át. Ha $\boldsymbol{y}$ az $\boldsymbol{x}'$-től csak egyetlen megváltoztatott bitben különbözik, akkor ezt ***felismerjük,*** mert szavunk megszegi az

$$y_1 \oplus \cdots \oplus y_7 = 0$$

hibaellenőrző összefüggést. A hibát nem tudjuk kijavítani, mert nem tudjuk, melyik bit romlott el.

Egyetlen hiba javítása

Ha ***javítani*** is akarunk hibákat, akkor több hibaellenőrző bitet kell csatolni az üzenethez. Próbálkozhatunk két további bit hozzáadásával:

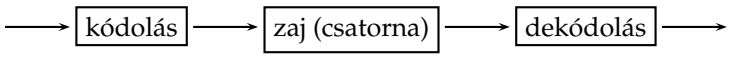
$$x_8 = x_1 \oplus x_3 \oplus x_5 \,,$$
$$x_9 = x_1 \oplus x_2 \oplus x_5 \oplus x_6 \,.$$

Ekkor a romlatlan $\boldsymbol{y}$ szónak a következő hibaellenőrző összefüggéseket kell teljesíteni:

$$y_1 \oplus \cdots \oplus y_7 = 0 \,,$$
$$y_1 \oplus y_3 \oplus y_5 \oplus y_8 = 0 \,,$$
$$y_1 \oplus y_2 \oplus y_5 \oplus y_6 \oplus y_9 = 0 \,,$$

vagy mátrix jelöléssel $\boldsymbol{Hy}$ mod $2 = 0$, ahol

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_9) \,.$$



21.12. ábra. Adatátvitel zajos csatornán

Észrevehetjük, hogy $\boldsymbol{h}_1 = \boldsymbol{h}_5$. A $\boldsymbol{H}$ mátrixot ***hibaellenőrző mátrixnak,*** vagy ***párosság-ellenőrző mátrixnak*** nevezik. A hibaellenőrző összefüggéseket más módon a következő formában írhatjuk:

$$y_1\boldsymbol{h}_1 \oplus \cdots \oplus y_5\boldsymbol{h}_5 \oplus \cdots \oplus y_9\boldsymbol{h}_9 = 0 \; .$$

Most, még ha $\boldsymbol{y}$ csak egyetlen helyen van is elrontva, sajnos még mindig nem javítható: mivel $\boldsymbol{h}_1 = \boldsymbol{h}_5$, a hiba lehet az 1. vagy az 5. helyen, nem tudnánk ezt a két esetet megkülönböztetni. Ha viszont $\boldsymbol{H}$ hibaellenőrző mátrixunkat úgy választjuk, hogy a $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots$ oszlopvektorok ***mind különbözők*** (persze nullától is), akkor ha csak egy hiba van, az javítható. Valóban, ha a hiba a 3. helyen van, akkor

$$\boldsymbol{H}\boldsymbol{y} \bmod 2 = \boldsymbol{h}_3 \; .$$

Mivel a $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots$ vektorok mind különbözők, ha a $\boldsymbol{h}_3$ vektort látjuk, következtethetjük, hogy az y_3 bit romlott el. Ezt a kódot a ***Hamming-kódnak*** nevezik. Például, a következő hibaellenőrző mátrix definiálja a 7 nagyságú Hamming-kódot:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_7) \; . \tag{21.27}$$

Általában, ha s hibaellenőrző bitünk van akkor kódunk nagysága $2^s - 1$ lehet, így az információtárolás („üzenet") számára fennmaradó bitek, az ***információ-bitek*** száma $m = 2^s - s - 1$. Tehát ahhoz, hogy m információbitet egyetlen hibától megvédjen, a Hamming-kód $\approx \lg m$ további, hibajavító bitet igényel. Ez sokkal jobb, mint minden bitet 3-szor ismételni.

Kódok

Foglaljuk össze a hibajavító felállást általánosabb formában. Zaj elleni védelem céljából a feladó ***kódolja*** az $\boldsymbol{x}$ ***üzenetet*** a ϕ_* ***kódoló függvény*** segítségével egy $\phi_*(\boldsymbol{x})$ hosszabb sorozatba, amelyről az egyszerűség kedvéért feltesszük, hogy bináris. Ezt a ***kódszót*** a zaj egy $\boldsymbol{y}$ sorozattá változtatja. A címzett megkapja $\boldsymbol{y}$-t és a ϕ^* ***dekódoló függvényt*** alkalmazza rá.

21.27. definíció. *A $\phi_* : \{0,1\}^m \to \{0,1\}^n$ és $\phi^* : \{0,1\}^n \to \{0,1\}^m$ függvényből álló párt* ***kódnak*** *nevezzük, ha minden $\boldsymbol{x} \in \{0,1\}^m$-re $\phi^*(\phi_*(\boldsymbol{x})) = \boldsymbol{x}$ teljesül. Az $\boldsymbol{x} \in \{0,1\}^m$ szavakat* ***üzeneteknek*** *nevezzük, az $\boldsymbol{y} = \phi_*(\boldsymbol{x}) \in \{0,1\}^n$ formájú szavakat* ***kódszavaknak.*** *(Néha a kódszavak halmazát egymagában is kódnak nevezik.) Minden $\boldsymbol{x}$ üzenethez, a $C_{\boldsymbol{x}} = \{\boldsymbol{y} : \phi^*(\boldsymbol{y}) = \boldsymbol{x}\}$ szóhalmazt az $\boldsymbol{x}$ **dekódoló halmazának** nevezzük. (Természetesen a dekódoló halmazok diszjunktak.) Az*

$$R = \frac{m}{n}$$

számot a kód ***sebességének (rátájának)*** *nevezzük.*

Azt mondjuk, hogy kódunk t ***hibát kijavít,*** *ha minden* $\boldsymbol{x} \in \{0,1\}^m$ *üzenetre, ha az* $\boldsymbol{y} \in \{0,1\}^n$ *megkapott szó a* $\phi_*(\boldsymbol{x})$ *kódszótól legfeljebb t helyen különbözik, akkor* $\phi^*(\boldsymbol{y}) = \boldsymbol{x}$.

Ha a sebesség R, akkor az n-bit kódszavak Rn bit hasznos információt hordoznak. A dekódoló halmazok nyelvén, a kód t hibát javít, ha minden dekódoló $C_{\boldsymbol{x}}$ halmaz minden olyan szót tartalmaz, mely a $\phi_*(\boldsymbol{x})$ kódszótól legfeljebb t jelben különbözik (ezeknek a szavaknak a halmaza egyfajta t sugarú "gömb").

A Hamming-kód egy hibát javít ki, és sebessége közel 1. A hibajavító kódokkal kapcsolatos egyik fontos kérdés az, mennyire kell a sebességet csökkenteni, ha több hibát akarunk kijavítani.

A kód-jelölések használatával most megfogalmazhatjuk ennek a fejezetnek az információtárolásra vonatkozó fő eredményét.

21.28. tétel (információtárolás hálózatban). *Léteznek* $\varepsilon, b, R > 0$ *konstansok, a következő tulajdonsággal. Minden m-re, minden* $n \geq m/R$*-re létezik egy* (ϕ_*, ϕ^*) *kód m üzenethosszal és n kódszóhosszal, és egy* $O(n)$ *nagyságú* $\mathcal{N}$ *ütemezett logikai hálózat n bemenettel és n kimenettel, és a következő képességgel: Tegyük fel, hogy a 0-dik időpontban, a hálózat memóriacellái egy tetszőleges* $\boldsymbol{Y}_0 = \phi_*(\boldsymbol{x})$ *kódszót tartalmaznak. Tegyük fel továbbá, hogy a hálózat* $\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_t$ *által leírt működése* ε*-megengedett. Ekkor* $\Pr[\,\phi^*(\boldsymbol{Y}_t) \neq \boldsymbol{x}\,] < te^{-bn}$.

A tétel azt mutatja, hogy lehetséges m bit információt t lépésen át tárolni, egy

$$O(\max(\lg t, m))$$

nagyságú ütemezett hálózatban. Amíg a tárolási t idő az exponenciális e^{cm} korlát alatt marad egy bizonyos c konstansra, addig ez a hálózatnagyság csak konstansszor nagyobb, mint a tárolt információ m mennyisége. (Amikor külön felújító szervet használtunk minden bithez, akkor további $\log m$ szorzóra volt szükség: lásd (21.26).) A tétel hallgat arról, milyen nehéz kiszámolni a $\phi_*(\boldsymbol{x})$ kódszót az induláskor, és milyen nehéz a $\phi^*(\boldsymbol{Y}_t)$ dekódolás a végén. Sőt, ezt a két műveletet is kívánatos lenne zajtűrő módon kivitelezni. Mindkét feladat megoldható, de a jelen fejezetben magára az információtárolásra koncentrálunk.

Lineáris algebra

Miután többet is fogunk bitmátrixokkal foglalkozni, kényelmes bevezetni az

$$\mathbb{F}_2 = (\{0,1\}, +, \cdot)$$

algebrai struktúrát, ami egy kételemű ***test***. Az összeadást és szorzást $\mathbb{F}_2$-ben modulo 2 definiáljuk (persze ez a szorzást illetően nem változás). Ugyancsak érdemes a bináris sorozatok $\{0,1\}^n$ halmazát az n-dimenziós vektortér $\mathbb{F}_2^n$ struktúrájával felruházni. Az elemi lineáris algebra legtöbb tétele és algoritmusa tetszőleges test felett is érvényes: többek között, definiálhatjuk egy mátrix sor-rangját, mint a lineárisan független sorok maximális számát, és az oszlop-rangot hasonlóan. Ekkor tétel az, hogy a sor-rang egyenlő az oszlop-ranggal. Mostantól, biteken és bitvektorokon végzett algebrai műveletek esetében + jelet írunk $\oplus$ helyett, amikor ez nem okozhat félreértést. Helymegtakarítás céljából, oszlopvektorokat néha vízszintesen fogunk írni: azaz

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = (x_1, \ldots, x_n)^T \ ,$$

ahol $\boldsymbol{A}^T$ jelöli az $\boldsymbol{A}$ mátrix transzponáltját. Az $\mathbb{F}_2^r$ vektortér feletti identitás mátrixot

$$\boldsymbol{I}_r$$

jelöli.

Lineáris kódok

Általánosítsuk a Hamming-kód gondolatát.

21.29. definíció. Egy (ϕ_*, ϕ^*) kód m üzenethosszal és n kódszóhosszal ***lineáris***, ha az üzenet- és kódvektorokat az $\mathbb{F}_2$ test feletti vektoroknak tekintve, a kódoló függvényt a

$$\phi_*(\boldsymbol{x}) = \boldsymbol{G}\boldsymbol{x}$$

képlet adja meg, ahol $\boldsymbol{G}$ egy $m \times n$ mátrix, amit a kód ***generáló mátrixának*** neveznek. Az m szám a kód ***információ-bitjeinek*** száma, a

$$k = n - m$$

szám pedig a ***hibaellenőrző*** biteké.

21.9. példa. A $\boldsymbol{H}$ mátrixot (21.27)-ben írhatjuk így: $\boldsymbol{H} = (\boldsymbol{K}, \boldsymbol{I}_3)$, ahol

$$\boldsymbol{K} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}.$$

Ekkor a hibaellenőrző összefüggés így írható:

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{I}_4 \\ -\boldsymbol{K} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_4. \end{pmatrix}.$$

Amint látszik, az $y_1, \dots, y_4$ biteket tekinthetjük a kód üzenet-bitjeinek, vagy "információ-bitjeinek", és ezzel a Hamming-kód lineáris kóddá válik, az $(\boldsymbol{I}_4, -\boldsymbol{K})^T$ generáló mátrixszal. (Persze, $-\boldsymbol{K} = \boldsymbol{K}$ az $\mathbb{F}_2$ test felett.)

A következő állítás rutin lineáris algebrai módszerekkel bizonyítható, és általánosítja a hibaellenőrző mátrix és generáló mátrix közötti kapcsolatot, amit a 21.9. példában láttunk.

21.30. állítás. *Legyen $k, m > 0$, és $n = m + k$.*

(a) *Minden, az $\mathbb{F}_2$ test feletti $n \times m$, m rangú $\boldsymbol{G}$ mátrixhoz létezik egy $k \times n$, k rangú $\boldsymbol{H}$ mátrix, melyre*

$$\{\, \boldsymbol{G}\boldsymbol{x} : \boldsymbol{x} \in \mathbb{F}_2^m \,\} = \{\, \boldsymbol{y} \in \mathbb{F}_2^n : \boldsymbol{H}\boldsymbol{y} = 0 \,\}. \tag{21.28}$$

(b) *Minden, a $\mathbb{F}_2$ test feletti $k \times n$, k rangú $\boldsymbol{H}$ mátrixhoz létezik egy $n \times m$, m rangú $\boldsymbol{G}$ mátrix, mely a (21.28) egyenlőséget teljesíti.*

21.31. definíció. *Jelölje* $|\boldsymbol{x}|$ *egy* $\boldsymbol{x}$ *vektor nemzéró elemeinek számát: ezt az* $\boldsymbol{x}$ ***súlyának*** *is fogjuk hívni.*

A következőkben kényelmes lesz kódjainkat inkább a $\boldsymbol{H}$ hibaellenőrző mátrixból kiindulva megadni. Ha a mátrix rangja k, akkor a kód sebessége

$$R = 1 - k/n \ .$$

Az oszlopok bármely lineárisan független S részhalmazát rögzíthetjük, és az $i \in S$ indexeket nevezhetjük ***hibaellenőrző biteknek;*** az $i \notin S$ indexek lesznek ekkor az ***információ-bitek.*** (A 21.9. példában $S = \{5, 6, 7\}$.) De fontos operációkat lehet végrehajtani a kódon anélkül, hogy bitjeit szétválasszuk információ-bitekre és hibaellenőrző bitekre.

21.5.4. Frissítők

Egyetlen hiba kijavítása nem volt túl nehéz; sokkal nehezebb hasonló elrendezést találni 2 hiba kijavítására. Pedig n bit tárolásánál általában εn (azaz sokkal több, mint 2) bitünk romlik el minden lépésben. Vannak leleményes és meglehetősen hatékony kódok (n-től független) pozitív sebességgel, melyek ennyi hibát is kijavítanak. De az információ-tárolóban a hibajavító mechanizmus maga is zajban fog működni, ezért valami különösen egyszerűt keresünk. Szerencsére nem muszáj minden hibát kijavítani: elég, ha alaposan csökkentjük a számukat, akárcsak a felújító szervben, amit megbízható logikai hálózatokhoz használtunk korábban.

Az egyszerűség kedvéért hálózatunk kapuiként olyan Boole-függvényeket is megengedünk, melyeknek változó-száma nagy (bár konstans). Cserébe viszont logikai hálónk mélysége csak 1 lesz, a 21.4. fejezet felújító szervéhez hasonlóan. Minden kapu kimenete egy memóriacella (bit-tároló) bemenete. Az egyszerűség kedvéért a kaput és a memóriacellát azonosítjuk, és ***sejtnek*** hívjuk. Minden órajelre, a sejt leolvassa bemeneteit a többi sejttől, egy Boole függvényt alkalmaz rájuk, és tárolja az eredményt (a következő órajelig). Csakhogy most az egyes sejtek által kiszámolt Boole-függvény kissé bonyolultabb lesz, mint a korábbi egyszerű többségi szavazás a bemenő értékek között.

Pontosabban, felújító műveleteinket egy bizonyos $k \times n$ méretű $\boldsymbol{H} = (h_{ij})$ párosságellenőrző mátrix segítségével definiáljuk. Legyen $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ egy bitvektor. A $j = 1, \ldots, n$ értékekre, legyen V_j (a "vertikális" szóból) azon i indexek halmaza, melyekre $h_{ij} = 1$. Az $i = 1, \ldots, k$ értékekre, legyen H_i (a "horizontális" szóból) azon j indexek halmaza, melyekre $h_{ij} = 1$. Ekkor a $\boldsymbol{Hx} = 0$ feltételt úgy is kifejezhetjük, hogy minden i-re, $\sum_{j \in H_i} x_j \equiv 0 \pmod 2$. A H_i halmazokat ***párosság-ellenőrző halmazoknak*** nevezzük. Mostantól kezdve az i indexeket ***vizsgálatoknak*** fogjuk nevezni, a j indexeket ***helyeknek***.

21.32. definíció. *Egy* $\boldsymbol{H}$ *lineáris kód* ***alacsony-sűrűségű párosság-ellenőrző kód*** *a* $K, N > 0$ *korlátokkal, ha a következő feltételek teljesülnek:*

(a) *Minden* j*-re* $|V_j| \leq K$;

(b) *Minden* i*-re* $|H_i| \leq N$.

Más szavakkal, minden sor súlya legfeljebb N*, és minden oszlop súlya legfeljebb* K*.*

Konstrukcióinkban a K, N korlátokat konstansnak tartjuk, míg a kódszavak n hossza növekszik. Tekintsük a helyzetet, amikor $\boldsymbol{x}$ egy kódszó, melyet néhány hiba elrontott. Ha az x_j bit helyességét akarjuk ellenőrizni, megvizsgálhatjuk az

$$s_i = \sum_{j \in H_i} x_j$$

összegeket, az összes $i \in V_j$-re. Ha mindezek értéke 0, akkor nem gyanakodnánk arra, hogy x_j hibás. Ha ezen összegeknek csak egyike különbözik 0-tól, akkor tudni fogjuk, hogy hibák vannak $\boldsymbol{x}$-ben, de még mindig gondolhatjuk, hogy a hiba nem az x_j bitben van. De ha az összegeknek jelentős hányada nem 0, akkor gyaníthatjuk, hogy x_j a bűnös, és javasolhatjuk megváltoztatását. Ez a gondolat sugallja a következő definíciót.

21.33. definíció. *A K, N korlátokkal rendelkező $\boldsymbol{H}$ alacsony-sűrűségű párosság-ellenőrző kódhoz egy **frissítő** műveletet rendelünk, melyet az összes j helyen egyidőben kell végrehajtani:*

> *Állapítsuk meg, hogy az s_i összegek között ($i \in V_j$-re) több, mint $\lfloor K/2 \rfloor$ különbözik-e nullától. Ha igen, billentsük át az x_j bitet.*

*Jelöljük $\boldsymbol{x}^{\boldsymbol{H}}$-val az $\boldsymbol{x}$-ből e művelettel kapott vektort. Az $0 < \alpha, \gamma < 1$ paraméterekre, nevezzük $\boldsymbol{H}$-t egy $(\alpha, \gamma, K, N, k, n)$-**frissítőnek**, ha minden n hosszúságú $\boldsymbol{x}$ vektorra, melynek súlya $|\boldsymbol{x}| \leq \alpha n$, az eredményvektor súlya így csökken: $|\boldsymbol{x}^{\boldsymbol{H}}| \leq \gamma\alpha n$.*

Könnyű észrevenni a sűrítőkhöz való hasonlóságot. A következő lemma a frissítők alkalmazását mutatja, és példát ad a lineáris kódok használatának előnyeire.

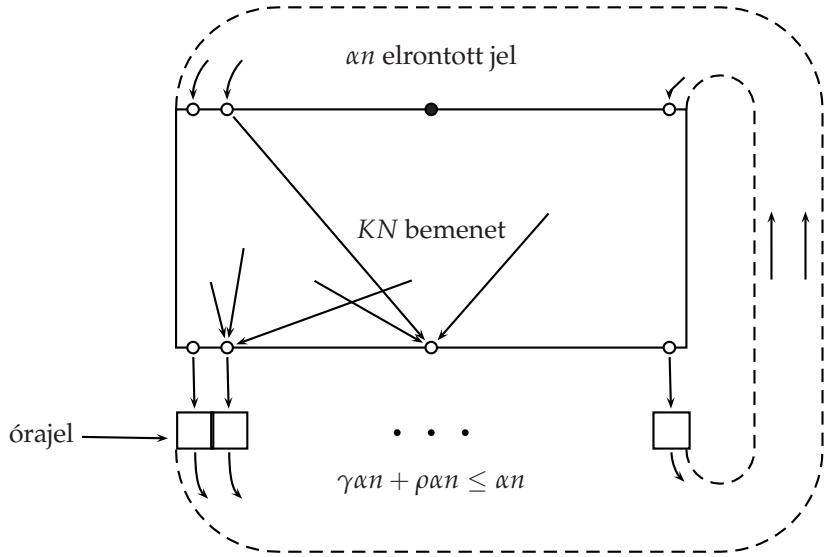
21.34. lemma. *Egy $(\alpha, \gamma, K, N, k, n)$-frissítő $\boldsymbol{H}$ mátrixhoz legyen $\boldsymbol{x}$ egy n-vektor és $\boldsymbol{y}$ egy n hosszúságú kódszó, melyekre $|\boldsymbol{x} - \boldsymbol{y}| \leq \alpha n$. Ekkor $|\boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{y}| \leq \gamma\alpha n$.*

Bizonyítás. Mivel $\boldsymbol{y}$ kódszó, tehát $\boldsymbol{H}\boldsymbol{y} = 0$, amiből $\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{y}) = \boldsymbol{H}\boldsymbol{x}$ következik. Tehát a hibajavítás ugyanazokat a biteket billenti át $\boldsymbol{x}-\boldsymbol{y}$-ben, mint $\boldsymbol{x}$-ben: $(\boldsymbol{x}-\boldsymbol{y})^{\boldsymbol{H}}-(\boldsymbol{x}-\boldsymbol{y}) = \boldsymbol{x}^{\boldsymbol{H}}-\boldsymbol{x}$, azaz $\boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{y} = (\boldsymbol{x} - \boldsymbol{y})^{\boldsymbol{H}}$. Tehát ha $|\boldsymbol{x} - \boldsymbol{y}| \leq \alpha n$, akkor $|\boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{y}| = |(\boldsymbol{x} - \boldsymbol{y})^{\boldsymbol{H}}| \leq \gamma\alpha n$. ■

21.35. tétel. *Minden $K \geq 11$ korláthoz léteznek α, γ, N és $R > 0$ paraméterek, melyekkel, minden elég nagy n kódhosszhoz van egy $(\alpha, \gamma, K, N, k, n)$-frissítő, legalább $n - k \geq Rn$ információ-bittel.*

Alkalmazzuk ezt a tételt, mielőtt bizonyítjuk.

A 21.28. tétel bizonyítása. A 21.35. tétel egy információtároló berendezést ad. Valóban, az $\boldsymbol{x} \to \boldsymbol{x}^{\boldsymbol{H}}$ műveletet megvalósíthatjuk úgy, hogy az $\boldsymbol{x}$ vektor minden j bitjéhez egyetlen g_j kaput használunk, melynek legfeljebb KN bemenete van. Most ha az $|\boldsymbol{x} - \boldsymbol{y}| \leq \alpha n$ egyenlőtlenség teljesül valamilyen $\boldsymbol{y}$ kódszóra, az $|\boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{y}| \leq \gamma\alpha n$ egyenlőtlenség következik. Persze minden kapu tévedhet $\leq \varepsilon$ valószínűséggel, és új eltéréseket vezethet be, valami $\boldsymbol{x}'$ vektort adva $\boldsymbol{x}^{\boldsymbol{H}}$ helyett. Legyen $e\varepsilon < \rho < 1 - \gamma$. Ekkor akárcsak korábban, annak valószínűségét, hogy több, mint $\rho\alpha n$ tévedés történik, az exponenciálisan csökkenő $(e\varepsilon/\rho)^{\rho n}$ kifejezés korlátozza. Kevesebb, mint ρn új eltéréssel még mindig $|\boldsymbol{x}' - \boldsymbol{y}| < (\gamma + \rho)\alpha n < \alpha n$. ■



21.13. ábra. Frissítő használata.

21.36. definíció. *Definiáljuk a 21.35. tétel* $\boldsymbol{H}$ *frissítőjét.*

Először, megfelelő k', n' *pozitív egész számokat választunk. Egy véletlen* $k' \times n'$ *nagyságú nemnegatív egész* $\boldsymbol{H}' = (h'_{ij})$ *mátrixot definiálunk a következőképpen. Válasszunk* Kn' *véletlen* $I_{js} \in \{1, \dots, k'\}$ *egész számot* $s = 1, \dots, K$*-re, egymástól függetlenül, és legyen*

$$h'_{ij} = \bigvee_s [I_{js} = i] \,.$$

Tehát minden i*-re* K *„golyót" dobálunk véletlenül a* $(j,1), \dots, (j,k')$ *„urnák" egyikébe, és* $h'_{ij} = 1$ *mutatja, került-e a* (j,i) *urnába golyó. Legyen*

$$V_j = \{\, i : h'_{ij} > 0 \,\}, \quad H_i = \{\, j : h'_{ij} > 0 \,\} \,.$$

(Ez nem fog félreértést okozni, noha korábban V_j, H_i *a* $\boldsymbol{H}$ *mátrixhoz volt rendelve hasonló módon.) A* $\boldsymbol{H}'$ *mátrix* $\boldsymbol{H} = (h_{ij})$ *részmátrixát a következőképpen definiáljuk. Legyen* $\mathcal{R} = \{r_1, \dots, r_k\}$ *a* $\boldsymbol{H}'$ *azon* i *sorainak halmaza, melyekre* $|H_i| \le N$*, és legyen* $\mathcal{C} = \{c_1, \dots, c_n\}$ *azon* j *oszlopok halmaza, melyekre* $V_j \subset \mathcal{R}$*. Ekkor*

$$h_{ij} = h'_{r_i c_j} \,.$$

Tehát a $\boldsymbol{H}$ *mátrixot úgy kapjuk, hogy csak azokat az* i *sorokat (vizsgálatokat) tartjuk meg* $\boldsymbol{H}'$*-ból, melyekre* $H_i \le N$*, és csak azokat a* j *oszlopokat (helyeket), amelyeket ezek a sorelhagyások nem befolyásolnak.*

21.37. lemma. *Minden elég nagy* n'*-re, ha* $k' = \lceil 0.7n' \rceil$*, akkor* ≥ 0.9 *valószínűséggel a* $\boldsymbol{H}$ *részmátrix* k, n *dimenziói eleget tesznek a* $k'/2 \le k \le 0.8n$ *feltételnek.*

A lemma bizonyítását a 21.5-2. feladat vázolja. Ennek a lemmának a segítségével fogunk egy $R \geq 0.2$ sebességű frissítőt készíteni.

Egy n'-dimenziós $\boldsymbol{z}$ vektorhoz definiáljuk az n-dimenziós $\mathrm{Le}(\boldsymbol{z})$ vektort: $(\mathrm{Le}(\boldsymbol{z}))_i = z_{c_i}$, és egy n-dimenziós $\boldsymbol{x}$ vektorhoz, legyen $\mathrm{Fel}(\boldsymbol{x})$ a következő n'-dimenziós vektor: $(\mathrm{Fel}(\boldsymbol{x}))_{c_j} = x_j$, és $(\mathrm{Fel}(\boldsymbol{x}))_s = 0$, ha $s \notin \mathcal{C}$. Dolgozhatunk a $\boldsymbol{H}'$ mátrixszal, még akkor is, ha végül a $\boldsymbol{H}$ részmátrix hibajavító tulajdonságai érdekelnek minket. A következő, könnyen ellenőrizhető egyenlőség teszi ezt lehetővé:

$$\boldsymbol{x}^{\boldsymbol{H}} = \mathrm{Le}((\mathrm{Fel}(\boldsymbol{x}))^{\boldsymbol{H}'}). \tag{21.29}$$

Egy n'-dimenziós $\boldsymbol{x}$ vektorra legyen

$$E_{\boldsymbol{x}} = \{\, j : x_j = 1 \,\}\,.$$

Célunk megmutatni, hogy nagy valószínűséggel olyan $\boldsymbol{H}'$ mátrixot kapunk a véletlen választás után, hogy az $|\boldsymbol{x}^{\boldsymbol{H}'}| \leq \gamma\alpha n$ összefüggés minden $\boldsymbol{x}$ vektorra igaz lesz, melyre $E_{\boldsymbol{x}} \subset \mathcal{C}$ és $|E_{\boldsymbol{x}}| \leq \alpha n$.

21.38. definíció. *Definiáljuk a*

$$T = \lfloor K/2 \rfloor$$

küszöböt.

- *Egy $j \in \mathcal{C} \setminus E_{\boldsymbol{x}}$ hely* ***rossz,*** *ha az $i \in V_j$ vizsgálatok közül több, mint T olyan, hogy $H_i \cap E_{\boldsymbol{x}} \neq \emptyset$. Legyen* $\mathrm{Rossz}(\boldsymbol{x})$ *a rossz helyek halmaza.*
- *Egy $j \in E_{\boldsymbol{x}}$ hely* ***jó,*** *ha több, mint T olyan $i \in V_j$ vizsgálat van, melyre $H_i \cap E_{\boldsymbol{x}} = \{j\}$. Legyen* $\mathrm{Jó}(\boldsymbol{x})$ *a jó helyek halmaza.*

Szemléletesen, egy $j \in E_{\boldsymbol{x}}$ „hiba” hely akkor jó, ha $\boldsymbol{H}'$ biztosan kijavítja, és egy $j \notin E_{\boldsymbol{x}}$ „nem-hiba” hely akkor rossz, ha $\boldsymbol{H}'$ esetleg „kijavítja”. Könnyű látni, hogy ha a j hely jó, akkor $x_j^{\boldsymbol{H}'} = 0$, és ha egy $j \notin E_{\boldsymbol{x}}$ hely nem rossz, akkor $x_j^{\boldsymbol{H}'} = 0$.

A következőkben feltesszük, hogy $|\boldsymbol{x}| \leq \alpha n$ egy megfelelően kicsi α konstansra. Először felülről becsüljük a rossz helyek számát, megmutatva, hogy $|\mathrm{Rossz}(\boldsymbol{x})| \leq c_1\alpha n$ egy alkalmasan kicsi c_1 konstansra. Ezután alulról becsüljük a jó helyek számát, megmutatva, hogy $|\boldsymbol{x}| < c_2\alpha n$ vagy $|\mathrm{Jó}(\boldsymbol{x})| \geq (1-c_2)\alpha n$ egy alkalmasan kicsi c_2 konstansra, ahol $c_1 + c_2 < 1$. Ebből a két eredményből az következik, hogy $|\boldsymbol{x}|^{\boldsymbol{H}'} \leq (c_1 + c_2)\alpha n$, ezért $\gamma = c_1 + c_2$ választható a frissítőhöz.

21.39. lemma. *Legyen*

$$c_1 > 1/T. \tag{21.30}$$

Ekkor létezik egy $\alpha > 0$ konstans, melyre ≥ 0.9 valószínűséggel a $\boldsymbol{H}'$ mátrix választásában, minden $\boldsymbol{x} \in \mathbb{F}_2^{n'}$-re, ha $|\boldsymbol{x}| \leq \alpha n$, akkor $|\mathrm{Rossz}(\boldsymbol{x})| \leq c_1\alpha n$.

Bizonyítás. Rögzítsünk egy j helyet és egy $\boldsymbol{x}$ értéket. Minden $s = 1, \ldots, K$ értékre, annak valószínűsége, hogy H_{I_s} metszi az $E_{\boldsymbol{x}}$ halmazt, nem nagyobb, mint annak valószínűsége, hogy a véletlen I_s szám benne van a $\bigcup_{p \in E_{\boldsymbol{x}}} V_p$ halmazban, azaz legfeljebb

$$K\alpha n/k' \leq K\alpha n'/k' = K\alpha n'/\lceil 0.7n' \rceil \leq K\alpha/0.8\,,$$

ha n' nagy. Annak valószínűsége, hogy j rossz legfeljebb akkora, mint annak valószínűsége, hogy több, mint T ilyen s érték van:

$$p_x := \Pr[\, j \in \text{Rossz}(x)\,] \le \binom{K+1}{T+1}(K\alpha/0.8)^{T+1}\ .$$

Jelöljük a jobb oldalt $K_1\alpha^{T+1}$-el. Minden a $\mathcal{C}\setminus E_x$ halmazba eső j helyre, azok az események, hogy j rossz, függetlenek. Legyen $\beta = |x|/n$, akkor $\mathcal{C}\setminus E_x = (1-\beta)n$. Becsüljük meg annak a valószínűségét, hogy legalább $c_1\alpha n$ hely rossz a $\mathcal{C}\setminus E_x$ halmazban. Vezessük be a $c_1\alpha n = f(1-\beta)n$ jelölést (így $f = c_1\alpha/(1-\beta)$). A (21.6) egyenlőtlenség ekkor ezt adja:

$$\begin{aligned}\Pr[\,|\text{Rossz}(x)| \ge c_1\alpha n\,] = \Pr[\,|\text{Rossz}(x)| \ge f(1-\beta)n\,] &\le \left(\frac{ep_x}{f}\right)^{f(1-\beta)n}\\ &= \left(\frac{ep_x(1-\beta)}{c_1\alpha}\right)^{c_1\alpha n} = \left(\frac{eK_1\alpha^T(1-\beta)}{c_1}\right)^{c_1\alpha n}\\ &\le \left((eK_1/c_1)^{c_1}\alpha^{c_1T}\right)^{\alpha n}\ .\end{aligned}$$

Legfeljebb $\sum_{i\le\alpha n}\binom{n}{i} \le (e/\alpha)^{\alpha n}$ lehetséges választás van x-re, ha $|x| \le \alpha n$ (itt a (21.8) egyenlőtlenséget használtuk). Ezért

$$\Pr[\,\exists x(|x| \le \alpha n \wedge |\text{Rossz}(x)| \ge c_1\alpha n)\,] \le U(\alpha)^{\alpha n}\ ,$$

ahol

$$U(\alpha) = (e/\alpha)(eK_1/c_1)^{c_1}\alpha^{c_1T} = e(eK_1/c_1)^{c_1}\alpha^{c_1T-1}\ .$$

Ha α elég kicsi, akkor a (21.30) feltételből $U(\alpha) < 1$ következik. ■

A jó helyek számának alsó korlátját előkészítve, nevezzük a

$$W_x = \bigcup_{j\in E_x} V_j$$

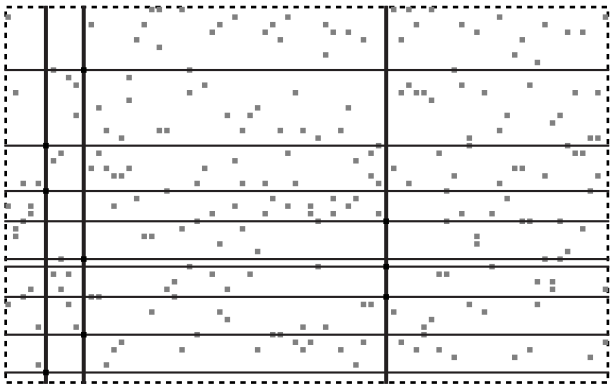
halmaz elemeit ***eleven vizsgálatoknak***: ezek a vizsgálatok találkoznak valóban hibával. A következő lemma azt mutatja, hogy ha az eleven vizsgálatok száma a maximumhoz közel van, akkor sok a jó hely.

21.40. lemma. *Legyen $0 < d$. Ha $|W_x| > K|x|(1-d)$, akkor $|\text{Jó}(x)| > (1-4d)|x|$.*

Általánosabban, $T \le K/2$-re, legyen $U_1,\dots,U_p$ egy halmazcsalád, ahol $|U_j| \le K$, és legyen $U'_j = U_j \setminus \bigcup_{s\ne j} U_s$. Ha $|\bigcup_j U_j| > (1-d)Kp$, akkor $|\{\, j : |U'_j| > T\,\}| > (1-4d)p$.

Bizonyítás. A speciális állítást az $E_x = \{s_1,\dots,s_p\}$, $U_j = V_{s_j}$ helyettesítéssel kapjuk az általánosabból. Legyen

$$U = \bigcup_j U_j,\quad U' = \bigcup_j U'_j,\quad J = \{\, j : |U'_j| > T\,\}\ .$$



21.14. ábra. A pöttyök 1-esek a $\boldsymbol{H}$ mátrixban, a függőleges vonalak mutatják a hibákat, a vízszintes vonalak pedig az eleven vizsgálatokat.

Tegyük fel, hogy $|\{\, j : |U'_j| > T \,\}| \le (1-4d)p$, megmutatjuk, hogy akkor $|U| \le Kp(1-d)$, az eredeti feltevéssel ellentétben. Az $|U| \le |U'|+(Kp-|U'|)/2$ egyenlőtlenség abból következik, hogy az U unióhalmaz $U \setminus U'$ részében minden elem legalább kétszer van fedve. Továbbá, a $c = 1-4d$ jelöléssel teljesül a

$$\begin{aligned} |U'| &\le |J|K + (p-|J|)T = pT + |J|(K-T) \le p(T + c(K-T)), \\ |U| &\le |U'| + (Kp-|U'|)/2 = Kp/2 + |U'|/2 \\ &\le Kp/2 + (pT + c(K-T))/2 = (p/2)(K(1+c) + T(1-c)) \\ &\le (p/2)(K(1+c) + (K/2)(1-c)) = Kp(3+c)/4 = Kp(1-d) \end{aligned}$$

egyenlőtlenség. ■

A következő lemma a 21.40. lemmával együtt már maga után vonja, hogy sok jó hely van.

21.41. lemma. *Létezik olyan $\alpha > 0$ konstans, hogy minden elég nagy n-re, ≥ 0.9 valószínűséggel a $\boldsymbol{H}'$ véletlen mátrix választásánál, minden olyan $\boldsymbol{x}$ vektorra, ha $4d\alpha n \le |\boldsymbol{x}| \le \alpha n$, akkor a*

$$2.8d^2K > 1 \tag{21.31}$$

egyenlőtlenségből $|W_{\boldsymbol{x}}| > (1-d)K|\boldsymbol{x}|$ következik.

Bizonyítás. Rögzítsünk egy $\boldsymbol{x}$ vektort, és bármilyen $\mathcal{W} \subset \{1,\dots,k'\}$ halmazt, melyre $|\mathcal{W}| \le (1-d)K|\boldsymbol{x}|$. A $W_{\boldsymbol{x}} \subset \mathcal{W}$ esemény akkor következik be, ha $I_{js} \in \mathcal{W}$ minden $j \in E_{\boldsymbol{x}}$-re, és $s = 1,\dots,K$-ra. Ennek valószínűsége legfeljebb $((1-d)K|\boldsymbol{x}|/k')^{K|\boldsymbol{x}|}$, ezért

$$\Pr[\,|W_{\boldsymbol{x}}| < (1-d)K|\boldsymbol{x}|\,] \le \binom{k'}{(1-d)K|\boldsymbol{x}|}((1-d)K|\boldsymbol{x}|/k')^{K|\boldsymbol{x}|}\,.$$

Jelöljük a jobb oldalt q_1-el. A $\beta = |\boldsymbol{x}|/k'$ jelöléssel, a (21.7) egyenlőtlenségből:

$$\binom{k'}{(1-d)K|\boldsymbol{x}|} \le ((1-d)K\beta/e)^{-(1-d)K\beta k'} .$$

Feltettük, hogy $4d\alpha \le \beta \le \alpha$ és $k' = \lceil 0.7n' \rceil \ge 0.7n$, ezért

$$\begin{aligned} q_1 &\le e^{(1-d)K\beta k'}((1-d)K\beta)^{dK\beta k'} \le e^{K\beta k'}(K\alpha)^{dK\beta k'} \\ &= \left(e^{1/d}K\alpha\right)^{dK\beta k'} \le \left(e^{1/d}K\alpha\right)^{4d^2K\alpha k'} \le \left(e^{1/d}K\alpha\right)^{2.8d^2K\alpha n} , \end{aligned}$$

ahol feltettük még, jogosan, hogy α elég kicsi az $e^{1/d}K\alpha \le 1$ teljesüléséhez is. Mint a 21.39. lemma bizonyításában, legfeljebb $(e/\alpha)^{\alpha n}$ lehetőség van $\boldsymbol{x}$ számára, ezért

$$\begin{aligned} &\Pr[\exists \boldsymbol{x}(4d\alpha n \le |\boldsymbol{x}| \le \alpha n \wedge |W_x| \le (1-d)K|\boldsymbol{x}|)] \\ &\qquad \le (e/\alpha)^{\alpha n}\left(e^{1/d}K\alpha\right)^{2.8d^2K\alpha n} = \left(e^{0.7cK+1}K^{2.8d^2K}\alpha^{2.8d^2K-1}\right)^{\alpha n}. \end{aligned}$$

Mivel feltettük a (21.31) feltételt, a zárójeles kifejezés kisebb lesz, mint 1, ha α elég kicsi. ■

A 21.35. tétel bizonyítása. Megfelelő c_1, d számokat fogunk választani a (21.30) és (21.31) feltételek teljesüléséhez, továbbá legyen $c_2 = 4d$, $\gamma = c_1 + c_2$. A 21.39. lemma szerint $\Pr[\,|\text{Rossz}(\boldsymbol{x})| \le c_1\alpha n\,] > 0.9$ a $\boldsymbol{H'}$ mátrix véletlen választásakor. Tegyük fel először, hogy $|\boldsymbol{x}| \le c_2\alpha n$, akkor

$$|\boldsymbol{x}^{\boldsymbol{H'}}| \le (c_1 + c_2)\alpha n. \tag{21.32}$$

Most pedig tegyük fel, hogy $c_2\alpha n \le |\boldsymbol{x}| \le \alpha n$. Ekkor a 21.41. lemma szerint $\Pr[\,|W_x| \ge (1-d)K|\boldsymbol{x}|\,] > 0.9$ a $\boldsymbol{H'}$ mátrix véletlen választásakor. A 21.40. lemmából $\Pr[\,|\text{Jó}(\boldsymbol{x})| \ge (1-c_2)|\boldsymbol{x}|\,] > 0.9$ következik. Tehát a frissítő művelet az $\boldsymbol{x}$ vektornak legfeljebb $c_2|\boldsymbol{x}| \le c_2\alpha n$ bitjét hagyja javítatlanul, azaz (21.32) teljesül, legalább 0.8 valószínűséggel.

Az van még hátra, hogy a c_1 és d paramétereket a (21.30) és (21.31) feltételek teljesítésével válasszuk. Például, $K = 21$-gyel a $c_1 = 0.15$, $d = 0.14$ választás $\gamma = c_1 + c_2 = 0.71$-et ad. Könnyű látni, hogy még akkor is, ha $K = 11$, lehet a c_1, d párt úgy választani, hogy $c_1 + c_2 < 1$ legyen. ■

A 21.4. fejezet végén felhozott összes kifogás még inkább érvényes a megbízható információtárolás itt bemutatott eredményeire. Az α, ε-ra kapott korlátok nagyon kicsik, és ennek megfelelően az az n tárnagyság, melyre ez az elrendezés először értelmet kap, nagyon nagy. (Lásd a 21.5-4. gyakorlatot.)

Gyakorlatok

21.5-1. Bizonyítsuk be a 21.30. állítást.

21.5-2. A $\boldsymbol{H}$ mátrix konstrukciójában, a $k' = 0.6n'$ választással, legyen $\mathcal{C}_0$ a kihagyott oszlopok halmaza. Bizonyítsuk be, hogy minden $i \in \{1,\dots,k'\}$, $j \in \{1,\dots,n'\}$-re

$$\Pr[\, j \in H_i,\ |H_i| > N \,] \le (K/k')\binom{n'-1}{N}(K/k')^N \le (K/k')\frac{(K/(1-r'))^N}{N!},$$

$$\mathbf{E}|\mathcal{C}_0|/n' \le K\frac{(K/(1-r'))^N}{N!} \to 0,$$

ha $N \to \infty$. Mutassuk meg ebből, hogy $\Pr[\, k'/2 \le k \le 0.8n \,] \to 1$, ha $N \to \infty$.

21.5-3. Bizonyítsuk be a (21.29) egyenlőséget.

21.5-4. A $K = 21$ esetre, számoljunk ki felső korlátokat N, α, ε-ra és alsó korlátot n-re, melyek biztosítják, hogy létezik $(\alpha, \gamma, K, N, k, n)$-frissítő a $k \le 0.8n$ tulajdonsággal. Ekkor létezik tehát információ-tároló hálózat n kódszó-hosszúsággal, és $R \ge 0.2$ sebességgel.

Feladatok

21-1. Kritikus érték

Vegyük a 21.2-5. gyakorlat $\mathcal{M}_k$ hálózatát, feltételezve, hogy minden kapu $\le \varepsilon$ valószínűséggel egymástól függetlenül téved. Tegyük fel, hogy a bemenetvektor csupa 0, és legyen $p_k(\varepsilon)$ annak valószínűsége, hogy a kimenet 1. Mutassuk meg, hogy létezik egy $\varepsilon_0 < 1/2$ érték azzal a tulajdonsággal, hogy minden $\varepsilon < \varepsilon_0$ esetén $\lim_{k\to\infty} p_k(\varepsilon) = 0$,, és $\varepsilon_0 < \varepsilon \le 1/2$ esetén $\lim_{k\to\infty} p_k(\varepsilon) = 1/2$. Mindkét esetben becsüljük meg a konvergencia sebességét.

21-2. Reguláris sűrítő

Egy sűrítőt úgy definiáltunk, mint egy d félreguláris páros multigráfot. Nevezzünk egy sűrítőt ***regulárisnak,*** ha d reguláris multigráf (a bemenő csúcsok foka is d). Bizonyítsuk a 21.21. tétel megfelelőjét: minden $\gamma < 1$-re létezik egy egész $d > 1$ és egy $\alpha > 0$ melyekkel minden pozitív egész k-ra létezik reguláris (d, α, γ, k)-sűrítő. *Útmutatás.* Válasszunk egy véletlen d-reguláris páros multigráfot a következő eljárással: (1. Helyettesítsünk minden csúcsot egy d csúcsból álló csoporttal. 2. Válasszunk egy teljes párosítást az új bemenő és kimenő csúcsok között. 3. Olvasszuk újra egy csúcsba össze mindegyik d csúcsból álló csoportot.) Bizonyítsuk, hogy e választás után kicsi annak a valószínűsége, hogy a kapott d-reguláris multigráf nem sűrítő. Ehhez, fejezzük ki ezt a valószínűséget faktoriálisokkal, és becsüljük azokat a Stirling-formulával.

21-3. Kétirányú tágító

Emlékezzünk a tágítók definíciójára a 21.4-3. gyakorlatban. Nevezzünk egy (d, α, λ, k)-tágítót ***regulárisnak,*** ha d-reguláris multigráf (a bemeneti csúcsok foka d). Ezt a multigráfot ***kétirányú tágítónak*** nevezzük, ha mindkét irányba tágító: A-ból B-be, és B-ből A-ba. Bizonyítsunk egy tételt, mely a 21-2. feladathoz hasonló: minden $\lambda < d$-re létezik egy $\alpha > 0$, mellyel minden pozitív egész k-ra létezik kétirányú (d, α, λ, k)-tágító.

21-4. Felújító szerv csak hármas szavazásból

A 21.21. tétel bizonyítása nem garantál (d, α, γ, k)-sűrítőt, ha $\gamma < 1/2$, $d < 7$. Ha csak

3-bemenetű többségi kapukat akarunk használni, akkor a következő konstrukció kínálkozik. Először egy 3-félreguláris páros G multigráfot készítünk $u_1, \ldots, u_k$ bemenetekkel és $v_1, \ldots, v_{3k}$, kimenetekkel, egy 3-bemenetű többségi kapuval minden v_i-ben. Azután a $w_1, \ldots, w_k$ új csúcsokat vezetjük be, minden w_j-ben egy 3-bemenetű többségi kapuval. A w_1 kapu a v_1, v_2, v_3 csúcsok többségét számolja ki, a w_2 kapu a v_4, v_5, v_6 többségét, és így tovább. Számítsuk ki, hogy a G gráf véletlen választása után a hálózat az $(u_1, \ldots, u_k)$ bemenetekkel és $(w_1, \ldots, w_k)$ kimenetekkel frissítő szervként tud-e működni. Ezután próbálkozzunk három lépéssel a kettő helyett (amikor G-nek $9k$ kimenete van), és állapítsuk meg, mit nyerünk.

21-5. Felújító szerv NOR kapukból

A többségi kapu nem az egyetlen kapu, mely a többséget erősíteni tudja. Emlékezzünk a 21.2-2. gyakorlatban bevezetett NOR kapura, és képezzük a következőt: $\mathrm{N}OR_2(x_1, x_2, x_3, x_4) = (x_1 \,\mathrm{N}OR\, x_2)\,\mathrm{N}OR\,(x_3\,\mathrm{N}OR\, x_4)$. Mutassuk meg, hogy egy, a 21-4.. feladathoz hasonló konstrukcióban a 3-bemenetű többségi kapu helyett $\mathrm{N}OR_2$ is használható.

21-6. Több véletlenség, kisebb felújító szervek

A 21.4-4. gyakorlat jelölésével tegyük fel, hogy mint ott, az F_v valószínűségi változók eloszlása nem függ az egész hálózat bemenő vektorától. Járjunk el a 21.4-6. gyakorlathoz hasonlóan minden felújító szerv építésében. Az eredeti hálózat minden kapujához egy új felújító szervet válasszunk: a választás függ az elkészült hálózatnak ezt a kaput megelőző részétől. Mutassuk meg, hogy ebben az esetben hibabecsléseink lényegesen megjavulnak. A javulás abból jön, hogy, mint a 21.4-6.. gyakorlatban, most nem kell a hibavalószínűséget megszorozni a fertőzött drótok összes lehetséges $\leq \alpha k$ nagyságú halmazainak számával. Mivel ezen véletlen halmaz eloszlása ismert, átlagolhatunk felette.

21-7. Közel-sorbarendezés tágítókkal

Ebben a feladatban megmutatjuk, hogy tágítókat „közel-sorbarendezésre” lehet alkalmazni. Legyen G egy reguláris kétirányú (d, α, λ, k)-tágító, melynek két k-nagyságú része A és B. Kőnig tétele szerint minden d reguláris páros multigráf (élhalmaza) d teljes párosítás (élhalmazainak) diszjunkt uniója: legyenek ezek $M_1, \ldots, M_d$. Egy ilyen tágítóhoz egy d mélységű logikai hálózatot rendelünk a következőképpen. A csúcsokat az $i = 0, 1, \ldots, d$ szintekbe csoportosítjuk. Az i szinten két diszjunkt k nagyságú csúcshalmaz, A_i, B_i helyezkedik el, az a_{ij}, b_{ij} $(j = 1, \ldots, k)$ csúcsokkal. Az a_{ij}, b_{ij} csúcsokban található érték x_{ij} illetve y_{ij} lesz. Jelöljük az i szint $2k$-elemű vektorát így: $\boldsymbol{z}_i = (x_{i1}, \ldots, y_{ik})$. Ha (p, q) az M_i párosítás egy éle, akkor egy $\wedge$ kaput teszünk a_{ip}-be és egy $\vee$ kaput b_{iq}-ba:

$$x_{ip} = x_{(i-1)p} \wedge y_{(i-1)q}, \quad y_{iq} = x_{(i-1)p} \vee y_{(i-1)q} \ .$$

Ez a hálózat a 0-kat A_i-be és az 1-eseket B_i-be igyekszik küldeni d lépésben. Általánosabban, a $\boldsymbol{z}_i$ vektorokban található értékek tetszőleges számok is lehetnek. Ekkor az $x \wedge y$ jelentése továbbra is $\min(x, y)$, és az $x \vee y$ jelentése $\max(x, y)$, és ezzel minden $\boldsymbol{z}_i$ vektor a $\boldsymbol{z}_0$ vektor permutációja. Legyen $\beta = (1 + \lambda)\alpha$. Bizonyítsuk be, hogy $\boldsymbol{z}_d$ a következő értelemben

β-rendezett: minden m-re, a z_d-nek m legkisebb számából legalább βm van a bal félben, és legnagyobb számaiból legalább βm van a jobb félben.

21-8. Felújító szerv közel-sorbarendezőkből
Építsünk felújító szervet tágítók felhasználásával, a következőképpen. Először is, az A bemeneti kábel minden vezetékét ágaztassuk ketté, hogy az A'_0, B'_0 halmazokat kapjuk. Most illesszük ezekhez a 21-7. feladat β-sorbarendezőjét: a kimenet az A'_d, B'_d halmazokra oszlik. Most ágaztassuk a B'_d halmaz vezetékeit az A''_0, B''_0 halmazokba. Illesszünk oda megint egy β-sorbarendezőt, ez az A''_d, B''_d kimenetekhez vezet. Tartsuk meg csak a $B = A''_d$ halmazt a kimenő kábelnek. Bizonyítsuk be, hogy az A-ból B-be vezető logikai vektor-hálózat felújító szerv.

Megjegyzések a fejezethez

A nagy eltérések tételét (21.1. tétel), vagy hasonló tételeket, sokszor Chernoffnak vagy Bernsteinnek tulajdonítják. A 21.1-2. gyakorlat az egyik gyakran használt változatot mutatja be.

A megbízhatatlan elemekkel végrehajtható megbízható számolások problémáját a [259] cikkben vizsgálta Neumann János a logikai hálózatok modelljén. Az ottani eredmény teljes bizonyítása először R. L. Dobrusin és Sz. I. Ortyukov [92] cikkében jelent meg (melyben a felújító szerv nem ugyanaz, mint amit Neumann János javasolt). Fejezetünk előadása N. Pippenger [280] cikkének egyes részeire épül.

Dobrusin és Ortyukov alsó korlátja a [91] cikkben (melynek hibáit a [279], [294] és [131] cikk javítja) azt mutatja, hogy az $\lg n$ redundancia általában elkerülhetetlen egy olyan megbízható számolásban, melynek bonyolultsága n. De ez az alsó korlát csak annak szükségességét mutatja, hogy a bemenetet redundánsan kódolt formába kell tenni (különben az első lépésben kritikus információ veszhet el). Amint [280] mutatja, több fontos függvényosztály esetén lineáris redundancia is elegendő.

Természetesnek látszik a kezdeti kódolás költségét különválasztani: akkor talán a számolás többi része sokkal kevesebb redundanciával is végrehajtható. D. Spielman [320] cikke ebben az irányban fontos lépést tett, (lényegében) az ütemezett logikai hálózatok modelljében. Spielman egy w elemi alkotórészből álló hálózaton t ideig futó számítást vesz, és megbízhatóvá teszi, csak $(\lg w)^c$-szer több processzort használva, és $(\lg w)^c$-szer tovább futtatva. A hibavalószínűség $t\exp(-w^{1/4})$. Ez kicsi mindaddig, amíg t nem sokkal nagyobb, mint $\exp(w^{1/4})$. Tehát a redundanciát a *térszükséglet* logaritmusának egy hatványával lehet korlátozni; az időszükséglet nem is jelenik meg nyíltan. Egy (aciklikus) logikai hálózatban a tér- és idő-bonyolultság nincs külön definiálva: a hálózat elemszáma azzal a mennyiséggel analóg, amit más modellekben az idő- és a tér-bonyolultság összeszorzásával kapunk.

Az információtárolási eredményekhez A. V. Kuznyecov [215] cikkét használtuk (a cikk főtétele, a frissítők létezéséről, M. Pinszker eredménye). Az alacsony-sűrűségű párosságellenőrző kódokat R. G. Gallager vezette be a [125] könyvben, és információ-tárolási használatukat először M. G. Taylor javasolta [333] cikkében. Ilyen kódoknak új, konstruktív változatait fejlesztették ki M. Sipser és D. Spielman a [319] cikkben, szupergyors kódolással és dekódolással.

A logikai hálózatoknál sokkal szabályosabb párhuzamos számolási modellben, a sejtautomatákban is lehetséges megbízható számolás. Ezeket az eredményeket itt nem volt alkalmunk bemutatni: lásd például a [133] és [130] cikkeket.

A 21.2-4. gyakorlat C. Shannontól származik: lásd [309]. A legrosszabb függvényekre az aszimptotikusan legjobb hálózatnagyságot O. B. Lupanov találta meg a [235] cikkben. A 21.3-1. gyakorlat a [92] cikken alapul, a 21.3-2. gyakorlat pedig az [91] cikken (és javításain).

A 21.4-3. gyakorlatban bevezetett tágítókat az elméleti számítógéptudományban kiterjedten használják: bevezetésnek, lásd a [253] könyvet. A könyv kis sajátérték-arányú gráfok konstrukciójára is ad utalásokat. A 21.4-5. gyakorlat és a 21-6. feladat a [92] cikken alapul.

A 21-1. feladatnál bonyolultabb kérdéseket tárgyal N. Pippenger könyve [281]. A 21-2. feladat módszerét véletlen d-reguláris multigráfok generálására például a [37] cikk elemzi. Sokkal nehezebb egyszerű reguláris gráfokat (nem multigráfokat) generálni egyenletes eloszlással: lást például a [198] cikket.

A 21-7. feladat az $\lg n$ mélységű sorbarendező hálózatok indító ötletén alapul (lásd [14]).

22. Számelmélet

Nem vállalkozhatunk arra, hogy a számelmélet által vizsgált összes kérdéssel foglalkozzunk. Célunk ebben a fejezetben, hogy néhány fontos eredményt ismertessünk, és rámutassunk ezeknek az informatikai algoritmusok fejődésére gyakorolt hatására. Érintünk több ezer, illetve néhány éves elméletet, amelyekben a közös az, hogy a jelen kor programozói is használják őket. Érdekes lehet az Olvasó számára az is, hogy miképpen találkoznak össze az informatikában olyan területek, mint például prímszámok, elliptikus görbék, vagy véges algebrai struktúrák elmélete.

Különös figyelmet szentelünk a prímtesztelő algoritmusoknak, hiszen a tárgyalt eredmények nagy része ezek tökéletesítését szolgálja. A ***prímteszt*** az az eljárás, amely során eldöntjük adott egész számról, hogy prím, vagy összetett. Az ismertetett eljárások futásidejének elemzésével általában csak érintőlegesen foglalkozunk, kivéve az ***AKS-algoritmust,*** amelynek részletes tanulmányozására külön alfejezetet szentelünk. Tesszük ezt azért, mert annak ellenére, hogy az eljárás gyakorlati megvalósítása még sok problémát vet fel a számítógép programozóknak, gondolunk itt például a nagy tárigényre, bizonyos számelméleti sejtések azt sugallják, valószínűleg nem pusztán elméleti jelentőséggel bír. Ez a 2002-ben publikált algoritmus azért is érdemel különleges figyelmet, mert az eddig ismerteknél kisebb futási idejű ***determinisztikus*** algoritmust ígér a prímtesztelés problémájának megoldására. Itt meg kell jegyeznünk, hogy determinisztikus prímteszteknek azokat az eljárásokat nevezzük, amelyek száz százalékos biztonsággal döntik el egy egész szám prím mivoltát. A programok futási idejének lecsökkentésének igénye a gyakorlati életben megteremtette az úgynevezett ***valószínűségi prímteszteket***. Ezeknek a lényege az, hogy az algoritmus gyorsabban lefut, de bizonyos összetett számokat prímnek ítél meg (részletesebben lásd a 22.4. alfejezetben). Természetesen a tesztek gyakorlati haszna függ attól, hogy a tévedés valószínűsége mekkora.

22.1. Véges kommutatív csoportok alaptétele, karakterek

Ebben a fejezetben bevezetünk számos olyan fogalmat, ami nem csak a véges struktúrák elméletében játszik fontos szerepet, hanem sok informatikai algoritmus fejlesztésében is.

22.1.1. Az alaptétel

Jelölje $O(A)$ a véges A csoport rendjét (azaz elemeinek számát). Tudjuk, hogy ha B részcsoport A-ban, akkor $O(B)$ osztója $O(A)$-nak.

Tetszőleges (multiplikatív) csoportban az $a \in A$ elem hatványai, azaz

$$\ldots, a^{-3}, a^{-2}, a^{-1}, \varepsilon = a^0, a, a^2, \ldots$$

vagy mind különbözőek, vagy nem, s az utóbbi esetben ezek az

$$U_a = \{a, a^2, \ldots, a^{n-1}, a^n = \varepsilon\}$$

halmazból kerülnek ki, ahol ε a csoport egységeleme. Ha n a legkisebb pozitív egész ezzel a tulajdonsággal, akkor az U_a halmazban nincs ismétlődés. Az n számot az a elem rendjének nevezzük, és $o(a)$-val jelöljük. U_a nyilván csoport (részcsoport A-ban), hiszen $a^n = \varepsilon$ miatt $a^k \cdot a^l = a^{(k+l) \pmod n}$, továbbá a^k inverze

$$\left(a^k\right)^{-1} = a^{n-k} \in U_a \ .$$

Világos, hogy $o(a)$ osztója $O(A)$-nak. Az egy elem által generált csoportokat (van olyan elem, amely hatványaival előállítja az összes többit) ***ciklikus csoportnak*** nevezzük. Az a elem az U_a csoport ***generáló eleme (generátora)***. Természetesen U_a generálható más, alkalmas elemével is. Az $a^k \in U_a$ akkor és csak akkor generálja az U_a csoportot, ha az

$$\{a^h, a^{2h}, \ldots, a^{nh} = \varepsilon\}$$

elemek valamennyien különbözők, azaz $a^{l_1 h} \neq a^{l_2 h}$, ha $1 \leq l_1 \leq l_2 \leq n$, ami akkor és csak akkor teljesül, ha $(h, n) = 1$ (relatív prímek). Érvényes tehát a következő állítás:

22.1. tétel. *Az a^k elem akkor és csak akkor generálja az U_a csoportot, ha $(h, n) = 1$. U_a-nak $\varphi(n)$ számú különböző generáló eleme van.*

Azt mondjuk, hogy valamely A kommutatív csoport a B és C részcsoportjainak ***direkt szorzata***, ha minden $a \in A$-hoz egyetlen $b \in B$ és $c \in C$ tartozik úgy, hogy $a = bc$ teljesül. Hasonlóan, A a $B_1, B_2, \ldots, B_k$ részcsoportok direkt szorzata, ha minden $a \in A$ elemet pontosan egyféleképpen írhatunk $a = b_1 \ldots b_k$ alakban, ahol $b_j \in B_j$ és $(j = 1, \ldots, k)$. A továbbiakban használjuk az $A = BC$, illetve az $A = B_1 B_2 \ldots B_k$ jelölést. A ***véges kommutatív csoportok alaptétele*** a következő:

22.2. tétel (véges kommutatív csoportok alaptétele). *Véges kommutatív csoport prímhatványrendű ciklikus csoportok direkt szorzata.*

A tétel bizonyítását a következő három segédtétel felhasználásával végezzük.

22.3. lemma. *Legyen A N-edrendű kommutatív csoport, $O(A) = N$, $N = N_1 N_2$, $(N_1, N_2) = 1$,*

$$E_1 = \{\alpha \in A \mid o(\alpha) \mid N_1\}, \quad E_2 = \{\beta \in A \mid o(\beta) \mid N_2\} \ .$$

Ekkor E_1, E_2 részcsoport A-ban, és $E_1 E_2 = A$.

Bizonyítás. Nyilvánvaló, hogy E_1, E_2 csoport és az is, hogy $E_1E_2 \subseteq A$. Legyen $\gamma \in A$. Az

$$1 = N_1u_1 + N_2u_2$$

egyenlet alkalmas egészekkel megoldható (az euklidészi-algoritmus következménye), ezért γ felírható a következő alakban:

$$\gamma = \left(\gamma^{N_1}\right)^{u_1} \cdot \left(\gamma^{N_2}\right)^{u_2} .$$

Legyen $\zeta = \left(\gamma^{N_1}\right)^{u_1}$ és $\xi = \left(\gamma^{N_2}\right)^{u_2}$. Ekkor $\zeta^{N_2} = \xi^{N_1} = \varepsilon$, mivel minden $\delta \in A$ elemre teljesül, hogy $\delta^N = \varepsilon$. Végül a felbontás egyértelműségét látjuk be. Ha az $\alpha = \xi_1\zeta_2 = \xi_2\zeta_1$ egyenlet megoldható lenne nem triviálisan, azaz úgy, hogy $\xi_1, \xi_2 \in E_1$, $\zeta_1, \zeta_2 \in E_2$, $\xi_1 \neq \xi_2$, akkor $\xi_1\xi_2^{-1} = \zeta_2\zeta_1^{-1}$, és így $\gamma = \xi_1\xi_2^{-1}$-re $\gamma \neq \varepsilon$, $\gamma \in E_1 \cap E_2$ teljesülne. Ezért

$$o(\gamma) \mid N_1, \quad o(\gamma) \mid N_2, \quad o(\gamma) \mid (N_1, N_2) = 1 ,$$

azaz $o(\gamma) = 1$ teljesülne, ami azt jelenti, hogy $\gamma = \varepsilon$, ez pedig egyértelműséghez vezet. ■

22.4. lemma. *Legyen A N-edrendű kommutatív csoport, és $N = P_1P_2 \dots P_r$, ahol $P_j = p_j^{\alpha_j}$, és $p_1, \dots, p_r$ különböző prímek. Legyen továbbá*

$$E_j = \{\alpha \in A \mid o(\alpha) \mid P_j\}, \quad (j = 1, \dots, r) .$$

Ekkor $A = E_1E_2 \dots E_r$.

A bizonyítás az előző lemmából, r-szerinti teljes indukcióval következik, ennek részletes közlésétől eltekintünk.

22.5. lemma. *Legyen az A kommutatív csoport rendje prímhatvány, és $O(A) = p^a$. Ekkor $A = U_1U_2 \dots U_t$, ahol az U_j csoportok ciklikusak.*

Bizonyítás. Legyen B az A-nak olyan részcsoportja, amelyre a lemma állítása igaz. $B = \{\varepsilon\}$-ra ez biztosan teljesül. $B \neq A$ esetén található olyan $\alpha \in A \setminus B$, hogy az állítás az $\{\alpha, B\}$ (α és B által generált csoport) csoportra is igaz. Elegendő ezt belátni.

Legyen $\beta \in A \setminus B$. $o(\beta) \mid O(A)$ miatt a $\beta, \beta^p, \beta^{p^2} \dots$ sorozatban fellép az ε egységelem. Van tehát olyan legnagyobb $(h-1)$ index, amelyre

$$\alpha = \beta^{p^{(h-1)}} \notin B .$$

Ekkor $\alpha^p \in B$. Világos, hogy az $\{\alpha, B\}$ csoport az $\{\alpha, \dots, \alpha^p = \varepsilon\}$ és a B csoport direkt szorzata, amiből következik, hogy $O(\{\alpha, B\}) = p \cdot O(B)$. Legyen $B = U_1U_2 \dots U_h$ a B ciklikus csoportokra bontása, $U_j = \{\gamma_j, \dots, \gamma_j^{k_j} = \varepsilon\}$, $O(U_j) = k_j$. Mivel $\alpha^p \in B$, ezért $\alpha^p = \gamma_1^{i_1} \cdot \dots \cdot \gamma_h^{i_h}\}$ $(1 \leq i_\nu \leq k_\nu)$, ahol k_ν a p hatványa. Az U_j indexeit permutálva, ha szükséges, elérhetjük, hogy $p \mid i_j$, ha $j = \nu+1, \dots, h$ és $p \nmid i_j$, ha $j = 1, \dots, \nu$.

Legyen

$$\alpha^* = \alpha \cdot \gamma_{\nu+1}^{-\frac{i_{\nu+1}}{p}} \dots \gamma_h^{-\frac{i_h}{p}} .$$

Ekkor $\alpha^* \notin B$, $(\alpha^*)^p = \alpha^p$. Cseréljük az α-t α^*-gal. Ekkor az új α-val kapjuk, hogy

$$\alpha^p = \gamma_1^{i_1} \dots \gamma_m^{i_m}, \quad (1 \le i_j < k_j, \ p \nmid i_j) \ .$$

Az egyenlet mindkét oldalát $o(\alpha)$-adik hatványra emelve:

$$\varepsilon = \alpha^{p \cdot o(\alpha)} = \gamma_1^{i_1 \cdot o(\alpha)} \dots \gamma_m^{i_m \cdot o(\alpha)} \ .$$

Mivel B az $U_1, \dots, U_m$ csoportok direkt szorzata, ezért minden B-beli elemnek egyetlen $\prod \gamma_\nu^{e_\nu}$ alakú felbontása van, ezért $\gamma_j^{i_j \cdot o(\alpha)} = \varepsilon$, tehát

$$k_j \mid i_j \cdot o(\alpha), \quad k_j \mid o(\alpha), \quad (j = 1, \dots, m) \ .$$

Legyen $k_j = p^{e_j}$. Feltehető, hogy $e_1 \ge e_2 \ge \dots \ge e_m$. Ekkor

$$\alpha^{p^{e_1+1}} = \prod_{j=1}^{m} \gamma_j^{p^{e_1}} = \varepsilon, \quad \alpha^{p^{e_1}} = \prod_{j=1}^{m} \gamma_j^{p^{e_1-1}} \neq \varepsilon \ ,$$

mivel $\gamma_1^{p^{e_1-1}} \neq \varepsilon$. Tehát $o(\alpha) = p^{e_1+1} = p \cdot o(\gamma_1)$.

$\gamma_1 \in \{\alpha, \gamma_2 \dots, \gamma_t\}$ miatt

$$\{\alpha, B\} = \{\alpha, \gamma_2 \dots, \gamma_h\} \ ,$$

azaz lemmánk igaz az $\{\alpha, B\}$ csoportra is. ■

A fenti három lemma bizonyításával egyúttal a *véges kommutatív csoportok alaptételét* is igazoltuk.

22.1.2. Csoportkarakterek

Legyen G véges kommutatív csoport. Jelölje X_G a G-n értelmezett χ komplex értékű függvényeknek a halmazát, amelyre teljesülnek a következő összefüggések minden $g, g_1, g_2 \in G$ esetén:

$$\chi(g_1 g_2) = \chi(g_1)\chi(g_2) \ , \tag{22.1}$$

illetve

$$|\chi(g)| = 1 \ . \tag{22.2}$$

Ekkor állíthatjuk, hogy X_G csoport. Igazolásképpen vegyük észre, hogy $\chi_1, \chi_2 \in X_G$ esetén $\chi_1 \cdot \chi_2 \in X_G$ is fennáll, mivel $\chi_1\chi_2(g) = \chi_1(g) \cdot \chi_2(g)$. A χ_0 függvény, amelyre minden $g \in G$ esetén $\chi_0(g) = 1$ teljesül, X_G-hez tartozik. Továbbá $\chi \in X_G$ esetén $\overline{\chi}$ is eleme X_G-nek, ahol $\overline{\chi}(g) := \overline{\chi(g)}$, és a felülvonás a komplex konjugáltat jelenti. Világos tehát, hogy X_G csoportot alkot a függvényszorzás műveletével. Tudjuk továbbá, hogy χ_0 az egységelem, és tetszőleges χ elem inverze $\overline{\chi}$.

22.6. definíció. *X_G a G karaktereinek csoportja.*

22.7. lemma. *Legyen G véges multiplikatív csoport, $G = AB$ a G felbontása az A és B részcsoportok direkt szorzatára.*
(i) Legyen $\chi_A \in X_A$, $\chi_B \in X_B$, és az $f : G \to \mathbb{C}$ függvényt értelmezzük a következő módon: ha $g \in G$, $g = a \cdot b$, $a \in A$, $b \in B$, akkor legyen $f(g) = \chi_A(a)\chi_B(b)$. Ekkor állítjuk, hogy $f \in X_G$.
(ii) Legyen $\chi_G \in G$ $g = \chi_G|_A$ a χ_G megszorítása az A halmazra, azaz $g(a) = \chi_G(a)$ minden $a \in A$-ra. Ekkor $g \in X_A$.
(iii) Az (i) és (ii) állításokból következik, hogy X_G az X_A és X_B részcsoportok direkt szorzata.
(iv) $\chi(g)^{O(G)} = 1$ minden $g \in G$-re.

Bizonyítás. A lemma állításai nyilvánvalóan igazak. Némi érvelés csak (iv)-hez szükséges. $\chi(\varepsilon^2) = \chi(\varepsilon) = \chi(\varepsilon)\chi(\varepsilon)$ miatt $\chi(\varepsilon) = 1$, és $g^{O(G)} = 1$ miatt teljesül a

$$1 = \chi(g^{O(G)}) = \chi(g)^{O(G)}$$

egyenlőség. ■

22.8. lemma. *Legyen G ciklikus csoport g generáló elemmel, és $N = O(G)$. A $\chi(g)$ érték teljesen meghatározza a $\chi \in X_G$ függvényt, másrészt $\chi(g)$ tetszőleges N-edik egységgyök lehet, azaz $O(X_G) = N$.*

Bizonyítás. Legyen $\omega = e^{2\pi i/N}$ és $\chi_k(g) = \omega^k$ $(k = 0, \dots, N-1)$. Ekkor $\chi_k(g^r) = \omega^{kr}$. Továbbá

$$\chi_k(g^r) \cdot \chi_l(g^r) = \omega^{kr} \cdot \omega^{lr} = \omega^{(k+l)r} ,$$

és a jobb oldal értéke megegyezik a $\chi_{k+l \pmod N}(g^r)$ értékkel. Vegyük észre, hogy $\chi_k(\zeta) = \chi^k(\zeta)$ minden $\zeta \in G$-re, azaz X_G-ben $\chi_k = \chi^k$, ezért

$$X_G = \{\chi, \chi^2, \dots, \chi^{N-1}, \chi^N = \chi_0\} ,$$

X_G N-edrendű ciklikus csoport. ■

A következő állítás az előző segédtételből következik.

22.9. lemma. *Ha G N-edrendű ciklikus csoport, akkor X_G izomorf G-vel.*

Észrevehetjük továbbá, hogy fennáll a következő összefüggés:

$$\sum_{k=0}^{N-1} \chi_k(g^r) = \sum_{k=0}^{N-1} \chi_k \omega^{kr} = \begin{cases} N, & \text{ha } r = 0 , \\ 0, & \text{ha } f = 1, \dots, N-1 . \end{cases}$$

azaz

$$\frac{1}{N} \sum_{\chi \in X_G} \chi(a) = \begin{cases} 1, & \text{ha } a = \varepsilon , \\ 0, & \text{ha } a \neq \varepsilon . \end{cases}$$

22.10. tétel. *Legyen G az $U_{a_1}, \dots, U_{a_h}$ ciklikus csoportok direkt szorzata, $O(G) = N$, $O(U_{a_j}) = n_j$ (j=1,...,h). Ekkor X_G az $X_{U_{a_1}}, \dots, X_{U_{a_h}}$ csoportok direkt szorzata, X_G izomorf G-vel. Továbbá*

$$\frac{1}{N}\sum_{\chi\in X_G}\chi(a)=\begin{cases}1, & \text{ha } a=\varepsilon\,,\\ 0, & \text{ha } a\neq\varepsilon\,.\end{cases} \tag{22.3}$$

$$\frac{1}{N}\sum_{a\in X_G}\chi(a)=\begin{cases}1, & \text{ha } \chi=\chi_0\,,\\ 0, & \text{ha } \chi\neq\chi_0\,.\end{cases} \tag{22.4}$$

Bizonyítás. A tétel első része az előző lemmákból világos, csak a (22.3) és (22.4) képleteket kell bizonyítani.

(22.3) igaz, ha $a=\varepsilon$. Legyen most $a\neq\varepsilon$, $a=a_1^{\nu_1},\dots,a_h^{\nu_h}$. Létezik olyan ν_j, amelyre $1\le\nu_j\le n_j$: $\chi=\chi^{(1)}\dots\chi^{(h)}$ végigfut X_G-n, ha $\chi^{(j)}$ az $X_{U_{a_j}}$ karaktercsoportban fut végig minden j-re. Ekkor

$$\frac{1}{N}\sum_{\chi\in X_G}\chi(a)=\prod_{j=1}^{h}\frac{1}{n_j}\left(\sum_{\chi(j)}\chi^{(j)}(a_j^{\nu_j})\right)\,.$$

A jobb oldalon álló

$$\frac{1}{n_j}\sum_{\chi(j)}\chi^{(j)}(a_j^{\nu_j})$$

tényező nulla, ha $\nu_j\neq n_j$, (22.3) tehát fennáll.

Világos, hogy

$$a_1^{l_1}\cdot\dots\cdot a_h^{l_h}\quad(l_j=1,\dots,n_j,\ j=1,\dots,h)$$

minden G-beli elemet pontosan egyszer állít elő. Ezért fennáll a

$$\sum_{a\in G}\chi(a)=\prod_{j=1}^{h}\left(\sum_{l_j=1}^{n_j}\chi(a_j)^{l_j}\right)$$

egyenlőség. ∎

Ha $\chi=\chi_0$, akkor a jobb oldal N. Ha $\chi\neq\chi_0$, akkor legalább egy a_j-re $\chi(a_j)\neq 1$, és ezért a jobb oldal zérus, amivel a tételt beláttuk.

22.1.3. A redukált maradékosztályok csoportja

Adott $m\neq 0$ egészre jelölje $\mathbb{Z}_m^*$ a ***redukált maradékosztályok multiplikatív csoportját,*** azaz legyen

$$\mathbb{Z}_m^*=\{a\pmod m\mid(a,m)=1\}\,.$$

Világos, hogy $\mathbb{Z}_m^*$ csoport, ugyanis $(a,m)=1$, $(b,m)=1$ esetén $(ab,m)=1$, $1\pmod m\in\mathbb{Z}_m^*$, és

$$a^{\varphi(m)-2}\equiv a^{-1}\pmod m\,.$$

A kínai maradéktételből következik, hogy

$$\mathbb{Z}_m^*=\mathbb{Z}_{p_1^{a_1}}^*\otimes\dots\otimes\mathbb{Z}_{p_r^{a_r}}^*\,,$$

ha m törzstényezős felbontása $m = p_1^{a_1} \dots p_r^{a_r}$.

Mivel p prímszámra $\mathbb{Z}_p$ test, és $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$, mint ahogy azt a 18. fejezetben láttuk, $\mathbb{Z}_p^*$ ciklikus csoport. Létezik tehát olyan $g \in \mathbb{Z}_p^*$ elem, amelyre

$$\mathbb{Z}_p^* = \{g, g^2, \dots, g^{p-1}\} .$$

Az ilyen g generáló elemet a számelméletben ***primitív gyöknek*** nevezzük.

Adott g primitív gyök és $y \in \mathbb{Z}_p^*$ esetén jelölje $ind(y) = ind_g(y)$ azt a $(\mathrm{mod}\ p-1)$ meghatározott egész számot, amelyre

$$g^{ind(y)} \equiv y \pmod{p} .$$

$ind_g(y)$ neve ***y g alapú (g szerinti) indexe***. Használatos még index helyett a ***véges logaritmus*** kifejezés is.

Világos, hogy

$$ind(y_1 y_2) \equiv ind(y_1) + ind(y_2) \pmod{p-1} ,$$

valamint

$$ind(1) \equiv 0 \pmod{p-1} ,$$

és így az $y \mapsto ind(y)$ függvény $\mathbb{Z}_p^*$-ot a $\mathbb{Z}_{p-1}$-re leképező lineáris függvény, tehát izomorfizmus.

Ha adott egy primitív gyök, $g \pmod{p}$, akkor a többi primitív gyököt nagyon egyszerűen meghatározhatjuk. Legyen $g_h = g^h$. Ahhoz, hogy g_h primitív gyök legyen, szükséges és elégséges feltétel, hogy g_h rendje $p-1$, azaz $g_h^\nu = 1$ nem teljesül egyetlen $1 \le \nu < p-1$ esetén sem. Mivel $g_h^\nu = 1$ akkor teljesül, ha

$$\nu \cdot h \equiv 0 \pmod{p-1} ,$$

ezért $(h, p-1) = 1$ esetén g_h primitív gyök, $(h, p-1) > 1$ esetén nem.

Következményként megállapíthatjuk, hogy adott p prímszámhoz pontosan $\varphi(p-1)$ számú különböző primitív gyök létezik.

$\mathbb{Z}_p^*$ karaktereit a következő módon állíthatjuk elő. Válasszunk egy $(p-1)$-edik komplex egységgyököt, legyen ez ω: $\chi_\omega(g^r) = \omega^r$ $(r = 0, 1, \dots, p-2)$.

Az $x^k \equiv a \pmod{p}$ kongruenciát adott g primitív gyökkel

$$k \cdot ind(x) \equiv ind(a) \pmod{p-1}$$

ekvivalens alakban írhatjuk. Az egyenlet megoldható, ha $(k, p-1) \mid ind(a)$, és e feltétel teljesülése esetén pontosan $(k, p-1)$ számú különböző megoldás van.

További vizsgálataink szempontjából érdekes a $k = 2$ eset.

22.11. definíció. *Legyen $p > 2$ prím és $(a, p) = 1$. Ekkor az a számot* ***kvadratikus maradéknak*** *nevezzük, ha megoldható az*

$$x^2 \equiv a \pmod{p}$$

kongruencia, és ***kvadratikus nemmaradéknak,*** *ha nem oldható meg.*

Megjegyezzük, hogy az $a \equiv 0 \pmod{p}$ számokat egyik kategóriába sem soroljuk be.

22.12. definíció. *Legyen p páratlan prím. A* ***Legendre-szimbólumot*** *$(a \mid p)$-vel, vagy $\left(\frac{a}{p}\right)$-vel jelöljük, és a következőképpen definiáljuk:*

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{ha } a \text{ kvadratikus maradék} \pmod{p}, \\ -1, & \text{ha } a \text{ kvadratikus nemmaradék} \pmod{p}. \end{cases}$$

Általánosíthatjuk a Legendre-szimbólum definícióját, ha $\mathbb{Z}$-n értelmezett függvényként fogjuk fel, vagyis

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{ha } p \mid a, \\ 1, & \text{ha } a \text{ kvadratikus maradék} \pmod{p}, \\ -1, & \text{ha } a \text{ kvadratikus nemmaradék} \pmod{p}. \end{cases}$$

Világos, hogy $a_1 \equiv a_2 \pmod{p}$ esetén $\left(\frac{a_1}{p}\right) = \left(\frac{a_2}{p}\right)$, továbbá

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right).$$

Ez utóbbi összefüggés a

$$\left(\frac{a}{p}\right) = \begin{cases} (-1)^{ind_g(a)}, & \text{ha } (a,p) = 1, \\ 0, & \text{ha } p \mid a \end{cases}$$

egyenletből nyilvánvalóan következik.

A Legendre szimbólum azon tulajdonságát, amely több valószínűségi prímtesztelő algoritmusok alapját képezi, ***Euler-kritériumnak*** szokták nevezni.

22.13. lemma (Euler). *Bármely $p > 2$ prímre és $0 < a < p$-re*

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}.$$

Bizonyítás. Legyen $y = a^{(p-1)/2}$. Mivel

$$y^2 = a^{p-1} \equiv 1 \pmod{p},$$

ezért $y \in \{-1, 1\}$. Ha a kvadratikus maradék, azaz $a \equiv b^2 \pmod{p}$, akkor $y \equiv b^{p-1} \equiv 1 \pmod{p}$, tehát

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}$$

teljesül.

Ha a kvadratikus nemmaradék, akkor $a \equiv g^{ind(a)}$ és $(ind(a), 2) = 1$, következésképpen

$$a^{\frac{p-1}{2}} \equiv g^{ind(a)\frac{p-1}{2}} \pmod{p}.$$

Mivel

$$ind(a)\frac{p-1}{2} \not\equiv 0 \pmod{p-1},$$

ezért $y \not\equiv 1 \pmod{p}$, tehát $y \equiv -1 = \left(\frac{a}{p}\right)$, amivel a bizonyítást befejeztük. ■

22.14. lemma. *Ha $p > 2$ prímszám, akkor $\mathbb{Z}_{p^\alpha}^*$ ciklikus csoport.*

Bizonyítás. Tudjuk, hogy az állítás igaz, ha $\alpha = 1$. Legyen most $\alpha = 2$. Mivel

$$(g+p)^{p-1} \equiv g^{p-1} + (p-1)pg \pmod{p^2},$$

teljesül, ezért vagy

$$g^{p-1} - 1 \not\equiv 0 \pmod{p^2},$$

vagy

$$(g+p)^{p-1} \not\equiv 1 \pmod{p^2}$$

fennáll, tehát g, vagy $(g+p)$ rendje biztosan $\varphi(p^2)$, ezért g, vagy $(g+p)$ primitív gyök (mod p^2). Indukcióval, tegyük fel, hogy

$$g_0^{\varphi(p^{\alpha-1})} \not\equiv 1 \pmod{p^\alpha}.$$

Az Euler–Fermat tétel szerint

$$g_0^{\varphi(p^{\alpha-1})} = 1 + tp^{\alpha-1}$$

alkalmas t-vel, ahol $p \nmid t$. Ezért

$$g_0^{\varphi(p^\alpha)} = (1+tp^{\alpha-1})^p \equiv 1 + tp^\alpha + \binom{p}{2}(p-1)t^2p^{2\alpha-2} \equiv 1 + tp^\alpha \not\equiv 1 \pmod{p^{\alpha+1}}$$

teljesül. ■

Nagy számok prímtesztelésénél gyakran használjuk a Legendre-szimbólum általánosítását tetszőleges egész számra:

22.15. definíció. *Legyen $m = p_1p_2\dots p_n$ tetszőleges páratlan egész, ahol a p_i-k nem feltétlenül különböző prímek és $(a,m) = 1$. Ekkor a következő szorzatot* ***Jacobi-szimbólumnak*** *nevezzük:*

$$\left(\frac{a}{m}\right) = \prod_{i=1}^{n}\left(\frac{a}{p_i}\right).$$

Az eddigiek ismeretében könnyen belátható az alábbi hét állítás, melynek bizonyítását a (3)-as és (4)-es pont kivételével az Olvasóra bízzuk.

22.16. tétel. *Legyen $m, m' > 2$ tetszőleges páratlan egész szám, ekkor*

$$(1)\quad a \equiv b \pmod m \Longrightarrow \left(\frac{a}{m}\right) = \left(\frac{b}{m}\right),$$

$$(2)\quad \left(\frac{ab}{m}\right) = \left(\frac{a}{m}\right)\left(\frac{b}{m}\right),$$

$$(3)\quad \left(\frac{-1}{m}\right) = (-1)^{\frac{m-1}{2}},$$

$$(4)\quad \left(\frac{2}{m}\right) = (-1)^{\frac{m^2-1}{8}},$$

$$(5)\quad \left(\frac{a}{m}\right)\left(\frac{b}{m}\right) = \left(\frac{a}{m\cdot m'}\right),$$

$$(6)\quad \left(\frac{a^2}{m}\right) = \left(\frac{a}{m^2}\right),$$

$$(7)\quad \left(\frac{a}{m}\right) = (-1)^{\frac{a-1}{2}\cdot\frac{m-1}{2}}\left(\frac{m}{a}\right).$$

Bizonyítás. (3) az Euler-kritériumból és abból a tényből következik, hogy páratlan t, s esetén

$$\frac{s-1}{2} + \frac{t-1}{2} \equiv \frac{st-1}{2} \pmod 2.$$

Mivel $(-1)^k k \equiv k \pmod k$, ha k páros, és $(-1)^k k \equiv p-k \pmod k$, ha k páratlan valamely p prímre, a (4) állítást a következőképpen láthatjuk be. Futtassuk k-t 1-től $(p-1)/2$-ig, és a megfelelő kongruenciák összeszorzásával kapjuk a következő levezetést:

$$\begin{aligned}
\left(\frac{p-1}{2}\right)!(-1)^{\frac{p^2-1}{8}} &= (-1)^1 1 (-1)^2 2 \ldots (-1)^{\frac{p-1}{2}} \frac{p-1}{2} \\
\left(\frac{p-1}{2}\right)!(-1)^{\frac{p^2-1}{8}} &\equiv 2\cdot 4\cdot 6 \cdot \cdots \cdot (p-1) \\
\left(\frac{p-1}{2}\right)!(-1)^{\frac{p^2-1}{8}} &\equiv 2^{\frac{p-1}{2}}\left(\frac{p-1}{2}\right)! \\
\left(\frac{p-1}{2}\right)!(-1)^{\frac{p^2-1}{8}} &\equiv \left(\frac{2}{p}\right)\left(\frac{p-1}{2}\right)!
\end{aligned}$$

Mivel $(\frac{p-1}{2})!$ relatív prím p-hez, van inverze $\pmod p$, amivel szorozva adódik, hogy

$$\left(\frac{2}{p}\right) \equiv (-1)^{\frac{p^2-1}{8}}.$$

A kongruencia jobb és bal oldalán is csak ± 1 szerepelhet, tehát írhatunk = jelet $\equiv$ helyett. Ha m összetett, akkor is visszavezethetjük a problémát erre az esetre a következő összefüggés segítségével:

$$\frac{s^2-1}{8} + \frac{t^2-1}{8} \equiv \frac{(st)^2-1}{8} \pmod 2,$$

ahol s, t páratlan számok. ■

A teljesség kedvéért megjegyezzük még, hogy

$$\left(\frac{2}{m}\right) = (-1)^{\frac{m^2-1}{8}} = \begin{cases} 1, & \text{ha } m \equiv \pm 1 \pmod 8, \\ -1, & \text{ha } m \equiv 3, 5 \pmod 8. \end{cases}$$

Ezek után egyszerűen el tudjuk dönteni, hogy az

$$x^2 \equiv n \pmod p$$

kongruenciának létezik-e megoldása, azaz, hogy $\left(\frac{n}{p}\right) = 1$, vagy $p \mid n$ teljesül-e.

22.17. tétel. *Legyen $p > 2$ prímszám, és $\left(\frac{n}{p}\right) = 1$. Az*

$$x^2 \equiv n \pmod p$$

kongruencia egyik megoldása x_0, ahol

(1) $p = 4k + 3$ *esetén* $x_0 \equiv n^{k+1} \pmod p$,

(2) $p = 8k + 5$, $n^{2k+1} \equiv 1 \pmod p$ *esetén* $x_0 \equiv n^{k+1} \pmod p$,

(3) $p = 8k + 5$, $n^{2k+1} \equiv -1 \pmod p$ *esetén* $x_0 \equiv \left(\frac{p+1}{2}\right)(4n)^{k+1} \pmod p$.

Bizonyítás. Mivel $\left(\frac{n}{p}\right) = 1$, ezért

$$n^{\frac{p-1}{2}} \equiv 1 \pmod p.$$

Az (1) esetben

$$(n^{k+1})^2 \equiv n^{2k+2} \equiv n^{\frac{p-1}{2}} \cdot n \equiv n \pmod p.$$

A (2), (3) esetben

$$1 \equiv n^{\frac{p-1}{2}} \equiv (n^{2k+1})^2 \pmod p,$$

és így (2) esetén $(n^{k+1})^2 \equiv n$, (3) esetén

$$\frac{(4n)^{2k+2}}{4} \equiv 2^{4k+2} \cdot n^{2k+2} \equiv n \pmod p,$$

mivel $\left(\frac{2}{p}\right) = -1$. ■

A Legendre-, illetve Jacobi-szimbólumot kiszámoló algoritmusoknál hasznos eszköz az úgynevezett ***kvadratikus reciprocitás törvénye,*** melynek bizonyítása Gausstól származik (itt nem közöljük).

22.18. tétel. *Legyen $m, n > 2$ relatív prím, páratlan számok. Ekkor*

$$\left(\frac{n}{m}\right)\left(\frac{m}{n}\right) = (-1)^{\frac{(m-1)(n-1)}{4}}.$$

Most tekintsünk néhány, az eddig ismertetett eredményekkel kapcsolatos alkalmazást. A következő algoritmus $m > 1$ páratlan szám és $n \geq 0$ egész bemenet esetén szolgáltatja az $\left(\frac{n}{m}\right)$ Jacobi-szimbólum értékét. Felhasználjuk a $b_i(x)$ függvényt, amely az x nemnegatív egész bináris alakjában a 2^i-es tag együtthatóját adja vissza, továbbá a CSERE(x, y) eljárást, amely x és y változó értékét cseréli meg. p változót az előjelváltások paritásának tárolására használjuk.

JACOBI(n, m)

```
 1  p ← 0
 2  n ← n (mod m)
 3  if n = 0
 4     then return 0
 5  if b_0(n) = 1
 6     then goto 8
 7  p ← p ⊕ b_1(m) ⊕ b_2(m)
 8  n ← n/2
 9  goto 4
10  if n = 1
11     then return 1 − 2p
12  p ← p ⊕ (b_1(n) ∧ b_1(m))
13  CSERE(n, m)
14  goto 2
```

Az Euler-kritérium vizsgálatán alapszik a ***Solovay–Strassen-teszt*** néven ismert eljárás is, amelynek ismertetésénél felhasználjuk a JACOBI eljárást, illetve feltesszük, hogy a RANDOM(x, y) függvény egy véletlen számot szolgáltat az $[x, y]$ intervallumból.

SOLOVAY–STRASSEN-PRÍMTESZT(n)

```
1  d ← RANDOM(1, n)
2  if a^((n−1)/2) ≢ JACOBI(a, n) (mod n)
3     then return ÖSSZETETT
4     else return VALÓSZÍNŰLEG PRÍM
```

Az eljárás érdekessége, hogy valójában valószínűségi teszt, de a szerzők megmutatták, hogy csak véges sok összetett szám esetén „téved”, azaz n bemeneti érték esetén legfeljebb $(n-1)/2$ összetett szám elégíti ki a vizsgált kongruenciát. Ez azt jelenti, hogy az algoritmus megfelelő számú ismétléssel és megfelelően választott a értékekkel tetszőlegesen megbízhatóvá tehető. Természetesen a determinisztikusság eléréséhez, azaz a nulla hibaszázalékhoz, annyi ismétlésre van szükség, ami gyakorlatilag használhatatlanná lassítja az algoritmust.

Itt kell még szólnunk a *Miller–Rabin-tesztről,* mivel bizonyítható, hogy az imént ismertetett módszernél „biztonságosabb”, azaz nem téved, ha a SOLOVAY–STRASSEN-PRÍMTESZT nem téved. A megfelelő algoritmus ismertetése az első kötetben már megtörtént, így közlésétől eltekintünk.

22.1.4. Index kalkulus

A vizsgálandó probléma a következő: adott p prímszámra és g primitív gyökre keresendő a

$$g^{L(h)} \equiv h \pmod{p}$$

kongruenciához tartozó $L(h)$ függvény. Világos, hogy $L(h)$ értelmezve van $h \in \mathbb{Z}_p^*$-ra, $L(h)$ (mod $p-1$) egyértelműen meghatározott, és

$$L(h_1 h_2) \equiv L(h_1) + L(h_2) \pmod{p-1}\,.$$

Az $L(h)$ függvény adott h értékre való meghatározását ***index kalkulusnak*** nevezzük.

22.1. példa. Legyen $p = 1217$. Némi számolással beláthatjuk, hogy 3 primitív gyök (mod p), ugyanis $p - 1 = 1216 = 2^6 \cdot 19$, és

$$3^{\frac{p-1}{2}} \not\equiv 1 \pmod{p},\ 3^{\frac{p-1}{19}} \not\equiv 1 \pmod{p}\,.$$

Az $L(37)$ értékét szeretnénk meghatározni.

Választunk egy B ***faktor bázist*** (kis prímszámok egy halmazát),

$$B = \{2, 3, 5, 7, 11, 13\}$$

és kísérletezünk, olyan y_j egészeket keresünk (viszonylag sokat), amelyekkel $3^{y_j} \pmod{p}$ B-beli prímek hatványainak szorzataként áll elő.

$$3^1 \equiv 3,\ 3^{24} \equiv -2^7 \cdot 7 \cdot 13,\ 3^{25} \equiv 5^3,\ 3^{30} \equiv -2 \cdot 5^2,\ 3^{54} \equiv -5 \cdot 11,\ 3^{87} \equiv 13, \pmod{p}\,,$$

ahonnan az alábbi kongruencia egyenleteket kapjuk (mod $p-1$):

$$608 \equiv L(-1),\ 1 \equiv L(3),\ 24 \equiv 608 + 2L(2) + L(7) + L(13),\ 25 \equiv 3L(5)\,,$$
$$30 \equiv 608 + L(2) + 2L(5),\ 54 \equiv 608 + L(5) + L(11),\ 87 \equiv L(13)\,.$$

Innen

$$L(3) \equiv 1,\ L(5) \equiv 819,\ L(13) \equiv 87\,,$$

majd

$$L(2) \equiv 30 - 608 - 2 \cdot 819 \equiv 216, L(11) \equiv 54 - 608 - L(5) \equiv 1059\,,$$

végül

$$L(7) \equiv 24 - 608 - 2L(2) - L(13) \equiv 113$$

adódik.

Így B valamennyi elemének indexét kiszámítottuk. Ezek után olyan j-t keresünk, amelyre $3^j \cdot 37$ (mod p) előáll B-beli elemek szorzataként.

$$3^{16} \cdot 37 \equiv 2^3 \cdot 7 \cdot 11 \pmod{p}\,,$$

ezért

$$16 + L(37) \equiv 3L(2) + L(7) + L(11) \pmod{p-1}\,,$$

ahonnan

$$L(37) \equiv 588 \pmod{p}\,,$$

azaz

$$3^{588} \equiv 37 \pmod{p}\,.$$

22.1.5. Sejtések primitív gyökökről

Amint azt már láttuk, adott p-re $\varphi(p-1)$ primitív gyök létezik. Tudjuk, hogy

$$\frac{\varphi(n)}{n} > \frac{C}{\lg\lg n}$$

alkalmas $C > 0$ állandóval, ezért nagy p-re viszonylag sok primitív gyök található. Várható, hogy a legkisebb pozitív primitív gyök $\pmod p$ (jelben: $\pi(p)$) viszonylag kicsi.

Bizonyítható, hogy amennyiben az ***általános Riemann-sejtés*** igaz, akkor

$$\pi(p) < C \cdot (\lg p)^2$$

teljesül, alkalmas C állandóval.

Nagy jelentőségű lenne, ha e tételt sikerülne minden állítás feltételezése nélkül bizonyítani. A Riemann-sejtés nélkül annyit tudunk, hogy

$$\pi(p) < C_\varepsilon \cdot p^{\frac{1}{4}} + \varepsilon$$

minden $\varepsilon > 0$-ra és alkalmas C_ε állandóra igaz. Ez az eredményt Wang és Burgess publikálták, míg Turán Pál bizonyította, hogy

$$\limsup_{p\to\infty} \frac{\pi(p)}{\lg p} > 0\,.$$

Egy másik nevezetes sejtés E. Artintól származik.

22.19. sejtés (Artin). *Legyen $a \in \mathbb{Z}$ nem teljes négyzet, továbbá $a \neq 0, \pm 1$. Jelölje N_a azon p prímeknek a halmazát, amelyekre a a p prímszám primitív gyöke, továbbá legyen*

$$n_a(x) = \sharp\,(N_a \cap [1, x])\ ,$$

valamint $\pi(x)$ az x-nél nem nagyobb prímek száma. Ekkor ***Artin sejtése*** *szerint*

$$\lim_{x\to\infty} \frac{n_a(x)}{\pi(x)} = A(a)\ ,$$

ahol $A(n)$ az úgynevezett ***Artin-konstans,*** *az a-tól függő pozitív állandó (amelyről tudjuk, hogy $A(n) > 0.35$).*

Megjegyezzük, hogy a bebizonyítatlan Riemann-sejtés egyfajta általánosításának igaz voltát feltételezve C. Hooley bebizonyította Artin sejtését.

Gyakorlatok

22.1-1. Bizonyítsuk be, hogy $\mathbb{Z}_4$ nem test. Adjuk meg a négy elemű testet. Anélkül, hogy ismernénk $\mathbb{F}_8$ műveleteit, meg tudjuk mondani, hogy hány primitív eleme van a multiplikatív csoportjának.

22.1-2. Írjunk programot a vázolt algoritmus segítségével a Jacobi-szimbólum kiszámolására.

22.1-3. Írjunk programot a Solovay–Strassen-prímtesztre.

22.2. Diofantikus approximáció, lánctörtek, Minkowski tétele

Mint szokásos, $\alpha \in \mathbb{R}$ esetén $[\alpha] = \alpha$ egész része, $\{\alpha\} = \alpha - [\alpha] = \alpha$ törtrésze, és

$$\| \alpha \| = \min\{\{\alpha\}, 1 - \{\alpha\}\}$$

α-nak a legközelebbi egésztől való távolsága. Világos, hogy $0 \le \{\alpha\} < 1$, $0 \le \| \alpha \| \le 1/2$.

A következő tételt Dirichlet bizonyította 1842-ben.

22.20. tétel (Dirichlet). *Legyenek α, Q valós számok, $Q > 1$. Ekkor létezik p, q egész szám úgy, hogy*

$$1 \le q < Q, \text{ és } |\alpha q - p| \le \frac{1}{Q} .$$

Bizonyítás. Tegyük fel, hogy Q egész. A $[0, 1)$ intervallumot osszuk Q részre:

$$I_u = \left[\frac{u}{Q}, \frac{u+1}{Q}\right) \ (u = 0, \dots, Q-1) .$$

Tekintsük az $\{\alpha\}, \{2\alpha\}, \dots, \{(Q-1)\alpha\}$ számokat. Ha ezek közül valamelyik I_0-ba, vagy I_{Q-1}-be esik, akkor az állítás igaz. Egyébként van olyan u, amelyre $\{r_1\alpha\}, \{r_2\alpha\} \in I_u$, $0 \le r_i < Q$, $r_1 \ne r_2$, és így

$$|(r_1\alpha - s_1) - (r_2\alpha - s_2)| \le \frac{1}{Q} .$$

Ha $r_1 > r_2$, $q = r_1 - r_2$, $p = s_1 - s_2$, akkor

$$1 \le q < Q, \text{ és } |\alpha q - p| \le \frac{1}{Q}$$

fennáll, amivel beláttuk a tételt abban az esetben, ha Q egész.

Tegyük fel, hogy Q nem egész szám. Alkalmazzuk a tétel bizonyított részét $\tilde{Q} = [Q] + 1$ választással. Az $1 \le q \le [Q]$ egyenlőtlenségből $1 \le q \le Q$, az $|\alpha q - p| \le 1/Q$ egyenlőtlenségből $|\alpha q - p| < 1/Q$ következik, bizonyítva a tételt. ■

22.21. megjegyzés. *(1) A 22.20. tételből következik, hogy*

$$\left|\alpha - \frac{p}{q}\right| \le \frac{1}{Qq} < \frac{1}{q^2} ,$$

(2) Legyen α irracionális szám. Ekkor létezik végtelen sok q egész, amellyel, és alkalmas p egésszel :

$$\left|\alpha - \frac{p}{q}\right| < \frac{1}{q^2} .$$

(3) Könnyen beláthatjuk Dirichlet tételének következő általánosítását. Legyenek $\alpha_1, \dots, \alpha_k$ valós számok, $Q > 1$. Ekkor léteznek $q, p_1, \dots, p_k$ valós számok úgy, hogy $1 \le q < Q^k$ és

$$\left|q\alpha - p_i < \frac{1}{Q}\right| \ (i = 1, \dots, k) .$$

22.2.1. Lánctörtek és általánosított lánctörtek

Legyen α tetszőleges valós szám, $a_0 = [\alpha]$, és $\alpha \notin \mathbb{Z}$ esetén α_1-et definiáljuk az $\alpha = a_0 + 1/\alpha_1$ formulával. Az eljárást ismételve $\alpha_k = a_k + 1/\alpha_{k+1}$ adódik, ahol $a_k = [\alpha_k]$. Tehát α_{k+1} az

$$\alpha_{k+1} = \frac{1}{a_k - \alpha_k}$$

képlettel számolható ki.

Ha α irracionális szám, akkor α_j irracionális, és e kifejtési algoritmus soha sem áll le. Világos, hogy

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots \cfrac{1}{a_n + \cfrac{1}{a_{n+1}}}}}} \tag{22.5}$$

fennáll. A (22.5) alakú kifejezést ***(egyszerű) lánctörtnek*** nevezzük. Ha az $1/\alpha_{n+1}$ tagot elhagyjuk, az így adódó racionális számot P_n/Q_n-nel jelöljük. Azt reméljük, hogy P_n/Q_n az α szám „jó közelítése". A lánctörtek fő előnye abban áll, hogy az a_ν „lánctörtjegyek" ismeretében a P_n/Q_n törtek egy rekurzív eljárással könnyen meghatározhatók. A következő algoritmus kiszámolja és tárolja az *avektor* nevű tömbben az a_i értékeket adott $\alpha = a/b$ számhoz, ahol $a, b \in \mathbb{R}$.

EGYSZERŰ-LÁNCTÖRT(a, b)

```
1  r_0 ← a
2  r_1 ← b
3  r ← 1
4  i ← 1
5  while r ≠ 0
6        do avektor[i] ← ⌊r_0/r_1⌋
7           r ← r_0 (mod r_1)
8           r_0 ← r_1
9           r_1 ← r
10          i ← i + 1
11 return avektor
```

Általában az

$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \cfrac{b_4}{\ddots}}}} \tag{22.6}$$

alakú kifejezéseket is lánctörtnek nevezzük. Szokás még az egyszerűbb

$$\left[a_0; \frac{b_1}{a_1}, \frac{b_2}{a_2}, \dots\right] = \left[a_0; \frac{b_k}{a_k}\right]_1^\infty ,$$

és

$$\left[a_0; \frac{b_1}{a_1}, \ldots, \frac{b_n}{a_n}\right] = \left[a_0; \frac{b_k}{a_k}\right]_1^n$$

jelölést is használni.

Legyen

$$\frac{P_n}{Q_n} = \left[a_0; \frac{b_1}{a_1}, \ldots, \frac{b_n}{a_n}\right] \quad (n = 1, 2 \ldots) , \tag{22.7}$$

$$P_0 = a_0, \; Q_0 = 1, \; P_{-1} = 1, \; Q_{-1} = 0 . \tag{22.8}$$

A (22.7) közelítő törtet egyszerűen kiszámíthatjuk az alábbi algoritmussal. A bemenet a fenti jelöléseket használva a_0, továbbá az *avektor*, *bvektor* tömbök, melyek tartalmazzák rendre az a_i, b_i értékeket ($i = 1, \ldots, n$).

KÖZELÍTŐ-TÖRT(a_0, *avektor*, *bvektor*)

```
1  c ← bvektor[n]/avektor[n]
2  for i ← 1 to n
3      do d ← avektor[n − i] + c
4         r ← bvektor[n − i]/d
5  return a_0 + c
```

Érvényes a következő formula a (22.8) jelöléseket is felhasználva

$$\begin{cases} P_k &= a_k P_{k-1} + b_k P_{k-2} , \\ Q_k &= a_k Q_{k-1} + b_k Q_{k-2} , \end{cases} \tag{22.9}$$

ahol $k = 1, 2, \ldots$.

Jelölje R_n a (22.7) kifejezés jobb oldalát, és tegyük fel, hogy a P_k, Q_k sorozatot a (22.9) formulával definiáljuk. Megmutatjuk, hogy $R_n = P_n/Q_n$. k-szerinti teljes indukciót alkalmazunk.

$k = 1$-re

$$R_1 = a_0 + \frac{b_1}{a_1} = \frac{a_0 a_1 + b_1}{a_1} = \frac{P_1}{Q_1} ,$$

ami (22.9) és (22.8)-ből rögtön következik.

Tegyük fel, hogy $R_j = P_j/Q_j$ ($j = 1, \ldots, k$). Ekkor

$$R_k = \frac{a_k P_{k-1} + b_k P_{k-2}}{a_k Q_{k-1} + b_k Q_{k-2}} = \frac{P_k}{Q_k} .$$

Világos, hogy R_{k+1}-et úgy kapjuk meg R_k-ból, hogy a_k helyébe $a_k + b_{k+1}/a_{k+1}$-et helyettesítünk, így

$$\begin{aligned} R_k &= \frac{(a_k + \frac{b_{k+1}}{a_{k+1}})P_{k-1} + b_k P_{k-2}}{(a_k + \frac{b_{k+1}}{a_{k+1}})Q_{k-1} + b_k Q_{k-2}} \\ &= \frac{a_{k+1}(a_k P_{k-1} + b_k P_{k-2}) + b_{k+1} P_{k-1}}{a_{k+1}(a_k Q_{k-1} + b_k Q_{k-2}) + b_{k+1} Q_{k-1}} \\ &= \frac{a_{k+1} P_k + b_{k+1} P_{k-1}}{a_{k+1} Q_k + b_{k+1} Q_{k-1}} = \frac{P_{k+1}}{Q_{k+1}} , \end{aligned}$$

amivel a bizonyítást befejeztük.

Érvényesek a következő formulák:

$$\frac{P_k}{Q_k} - \frac{P_{k-1}}{Q_{k-1}} = (-1)^{k-1} \cdot \frac{b_1 b_2 \dots b_k}{Q_{k-1} Q_k} \quad \text{ahol } (k \geq 1) , \tag{22.10}$$

$$\frac{P_k}{Q_k} - \frac{P_{k-2}}{Q_{k-2}} = (-1)^{k} \cdot \frac{b_1 b_2 \dots b_{k-1} a_k}{Q_{k-2} Q_k} \quad \text{ahol } (k \geq 3) , \tag{22.11}$$

Továbbá

$$\frac{P_k}{Q_k} - \frac{P_{k-1}}{Q_{k-1}} = \frac{\triangle_k}{Q_{k-1} Q_k} ,$$

ahol $\triangle_k$ a

$$\begin{vmatrix} P_k & P_{k-1} \\ Q_k & Q_{k-1} \end{vmatrix} \tag{22.12}$$

determináns. (22.9) miatt $\triangle_k = -b_k \triangle_{k-1}$, és így

$$\triangle_k = (-1)^k b_1 \dots b_k \triangle_0 ,$$

ahol

$$\triangle_0 = \begin{vmatrix} P_0 & P_{-1} \\ Q_0 & Q_{-1} \end{vmatrix} = -1 ,$$

ezért (22.10) teljesül. A (22.11) formula hasonlóan bizonyítható.

Ha a lánctört minden a_0, a_k, b_k $(k = 1, 2 \dots)$ jegye pozitív, akkor a páros indexű közelítő törtek monoton növekvő, a páratlan indexűek monoton csökkenő sorozatot alkotnak. Minden páros indexű tört kisebb bármelyik páratlan indexűnél.

Tegyük fel, hogy

$$\alpha = \lim_{n \to \infty} \frac{P_n}{Q_n}$$

létezik. Mivel

$$\frac{P_n}{Q_n} = \frac{P_0}{Q_0} + \sum_{k=1}^{n} \left(\frac{P_k}{Q_k} - \frac{P_{k-1}}{Q_{k-1}} \right) = \frac{P_0}{Q_0} + \sum_{k=1}^{n} (-1)^{k-1} \cdot \frac{b_1 b_2 \dots b_k}{Q_{k-1} Q_k} ,$$

ezért a P_n / Q_n határértéke akkor létezik, ha a jobb oldali sorozat végtelenhez tartva konvergens. Ha ez fennáll, akkor érvényes az

$$\left| \alpha - \frac{P_n}{Q_n} \right| \leq \sum_{k=n+1}^{\infty} \left| \frac{P_k}{Q_k} - \frac{P_{k-1}}{Q_{k-1}} \right| \leq \sum_{k=n+1}^{\infty} \left| \frac{b_1 \dots b_k}{Q_k Q_{k-1}} \right|$$

egyenlőség.

Igaz a következő állítás: ha a lánctört jegyei mind pozitívak,

$$b_k \leq a_k \text{ és } a_k \geq d > 0 \quad (k = 1, 2 \dots) , \tag{22.13}$$

akkor p_n/Q_n konvergens. A monotonitás miatt $\alpha = P_{2k}/Q_{2k}, \beta = P_{2k+1}/Q_{2k+1}$ határértéke létezik. Továbbá

$$\frac{P_{2k}}{Q_{2k}} < \alpha \le \beta < \frac{P_{2k+1}}{Q_{2k+1}} ,$$

és így

$$0 \le \beta - \alpha < \frac{P_{2k+1}}{Q_{2k}} - \frac{P_{2k}}{Q_{2k}} = \frac{b_1 \dots b_{2k} b_{2k+1}}{Q_{2k} Q_{2k+1}} = \zeta_k .$$

(22.9) második lépésében k-ra és $(k-1)$-re is felírva $Q_k \ge b_k(Q_{k-1} + Q_{k-2})$, azaz $Q_k \ge dQ_{k-2}$, és így $Q_k \ge b_k(1+d)Q_{k-2}$ adódik. Innen a

$$Q_{2k} \ge \frac{Q_{2k}}{Q_{2k-2}} \cdot \frac{Q_{2k-2}}{Q_{2k-4}} \cdot \dots \cdot \frac{Q_2}{Q_0} \cdot Q_0 \ge b_2 b_4 \dots b_{2k} (1+d)^k ,$$

illetve hasonlóan

$$Q_{2k+1} \ge b_1 b_3 \dots b_{2k+1} (1+d)^k ,$$

azaz

$$Q_{2k} Q_{2k+1} \ge b_1 b_2 \dots b_{2k+1} (1+d)^{2k} ,$$

vagyis

$$\zeta_k \le \frac{1}{(1+d)^{2k}} .$$

A következő konvergencia-kritérium Pringsheimtől származik. Ha a

$$|b_n| + 1 \le |a_n| \text{ ahol } (n = 1, 2 \dots) \tag{22.14}$$

egyenlőtlenségek teljesülnek, akkor a lánctört konvergens.

Térjünk vissza a (22.5) formulával definiált egyszerű lánctört kifejezéshez. Ekkor $b_n = 1$ minden $(n = 1, 2 \dots)$-re, $a_n > 0$ $(n \ge 1)$ esetén. Az előzőekből következik, hogy az α szám közelítő törtjeire

$$0 < \alpha - \frac{P_n}{Q_n} < \frac{1}{Q_n^2}, \quad \text{ha } n \text{ páros} ,$$

és

$$0 < \frac{P_n}{Q_n} - \alpha < \frac{1}{Q_n^2}, \quad \text{ha } n \text{ páratlan} ,$$

és így

$$\left| \alpha - \frac{P_n}{Q_n} \right| < \frac{1}{Q_n^2}$$

mindig fennáll (lásd a 22.21. megjegyzés (2) pontját).

A lánctörtek érdekes számelméleti alkalmazása az alábbi.

22.22. tétel. *Legyen $n > 2$ páratlan egész. Ha van olyan x egész, amelyre $x^2 \equiv -1 \pmod{n}$, akkor $n = a^2 + b^2$, $a, b \in \mathbb{N}$, $(a, b) = 1$ megoldható.*

Bizonyítás. Legyen $\omega^2 \equiv -1 \pmod{n}$, $\omega \in \mathbb{N}$. Ekkor a ω/n törtet alkalmas b nevezőjű törttel közelítve, úgy hogy $1 \le b \le \sqrt{n}$,

$$\left| \frac{\omega}{n} + \frac{k}{b} \right| < \frac{1}{b\sqrt{n}} \tag{22.15}$$

adódik Dirichlet tételéből. Legyen $a = \omega b + kn$. (22.15) miatt $|a| < \sqrt{n}$, és így $a^2 + b^2 < 2n$. Másrészt $a^2 + b^2 \equiv 0 \pmod{n}$, ezért $a^2 + b^2 = n$. ■

22.23. következmény. *Ha p prím, $p = 2$, vagy $p \equiv 1 \pmod 4$, akkor az $a^2+b^2 = p$ egyenlet megoldható egész a, b-vel. Ha $p \equiv -1 \pmod 4$, akkor az egyenlet nem oldható meg.*

Bizonyítás. $2 = 1^2 + 1^2$. $p \equiv 1 \pmod 4$ esetén $(-1/p) = 1$, tehát az $x^2 \equiv -1 \pmod p$ kongruencia megoldható, s az előző tétel szerint ekkor $p = a^2 + b^2$ megoldható.

Legyen $p \equiv -1 \pmod 4$. Ha $a^2 + b^2 = p$ megoldható, akkor az a, b megoldások relatív prímek p-hez, továbbá $(a, b) = 1$. Ha $p = a^2 + b^2$, akkor $a^{-1} \pmod p$-vel szorozva az egyenlet mindkét oldalát, $pb^{-1} = (ab^{-1})^2+1$, azaz $x = ab^{-1} \pmod p$ választással $x^2+1 \equiv 0 \pmod p$, és ez $(-1/p) = -1$ miatt nem teljesülhet. ■

Racionális számok lánctörtkifejtése és az euklidészi algoritmus között az alábbi nyilvánvaló összefüggés áll fenn. Ha $a, b \in \mathbb{N}$ $a = bq + r$, ahol $0 < r < b$, akkor

$$\frac{a}{b} = q + \frac{r}{b} = q + \frac{q}{\frac{b}{r}} \quad q = \left[\frac{a}{b}\right], \ \frac{r}{b} = \{\frac{a}{b}\} \ .$$

Tehát, ha

$$a = bq_1 + r_1, \ b = r_1q_2 + r_2, \ r_1 = r_2q_3 + r_3, \ \ldots, \ r_{t-2} = r_{t-1}q_t + r_t, \ r_{t-1} = r_tq_{t+1} \ ,$$

akkor

$$\frac{a}{b} = b + \cfrac{1}{r_1 + \cfrac{1}{r_2 + \cfrac{1}{\ddots_{r_{t-1} + \frac{1}{r_t}}}}} \ .$$

22.2.2. Minkowski tétele

Az n dimenziós euklidészi térben ($\mathbb{R}^n$) legyen adott valamely K tartomány (minden pontja belső pont), amely az origóra szimmetrikus ***(centrálszimmetrikus),*** és ***konvex,*** azaz $\underline{x}, \underline{y} \in K$ esetén $\mu\underline{x} + (1 - \mu)\underline{y} \in K$, ha $0 \le \mu \le 1$, továbbá K korlátos, ($\sup_{\underline{x}\in K} \| \underline{x} \| < \infty$).

Rácspontoknak nevezzük azokat az $\underline{m} = (m_1, \ldots, m_n)$ vektorokat, amelyek koordinátái egész számok. A rácspontok halmazát jelölje $\mathbb{Z}^n$. ***Minkowski tételét*** először az úgynevezett *egyszerű esetben* ismertetjük.

22.24. tétel. *Legyen a K korlátos, centrálszimmetrikus, konvex tartomány mértéke*

$$\lambda(K) > 2^n \ .$$

Ekkor K-ban van az origón kívül rácspont.

Bizonyítás. A bizonyításhoz elég belátni a következő állítást, amely Blichfeldt nevéhez fűződik. Ha

$$S = \frac{1}{2}K, \ S_{\underline{m}} = S + m \ ,$$

valamely $\underline{m} \in \mathbb{Z}^n$-re, akkor létezik olyan $\underline{m}^{(1)} \neq \underline{m}^{(2)}$ pár, amelyre

$$S_{\underline{m}^{(1)}} \cap S_{\underline{m}^{(2)}} \neq \emptyset \ .$$

Ezt a következő módon láthatjuk be. Mivel K korlátos, ezért K részhalmaza az

$$\{\underline{x} = (x_1, \ldots, x_n) \mid |x_j| \leq A, \ j = 1, \ldots, n\} = E(A)$$

négyzetnek, ha A elég nagy. Tekintsük most az $E(T)$ és $E(T-A)$ négyzetet. Az $E(T-A)$-hoz tartozó rácspontok száma $[2E(T-A)]^2$, feltéve, hogy T, A egész számok. Tekintsük most az

$$U = \cup_{\underline{m} \in E(T-A)} S_{\underline{m}}$$

halmazt. Világos, hogy $U \subseteq E(T)$. Ha Blichfeld tétele nem lenne igaz, akkor

$$\lambda(U) = [2(T-A)+1]^2 \lambda(S) \leq \lambda(E(F)) = (2T+1)^2 \ ,$$

és így fennáll a

$$\lambda(S) \leq \frac{(2T+1)^2}{(2T+1-2A)}$$

egyenlőtlenség, ahonnan $T \to \infty$ esetén $\lambda(S) \leq 1$, és

$$\lambda(K) = \lambda(2S) = 2^n \lambda(S) \leq 2^n$$

igaz lenne, amely ellentmond a feltételeinknek.

Ha

$$S_{\underline{m}^{(1)}} \cap S_{\underline{m}^{(2)}} \neq \emptyset \ (m^{(1)} \neq m^{(2)})$$

áll fenn, akkor

$$S_{\underline{0}} \cap S_{\underline{m}^{(2)} - \underline{m}^{(1)}} \neq \emptyset$$

is, tehát létezik olyan $\underline{m} \in \mathbb{Z}^n \setminus \{0\}$, nevezetesen $\underline{m}^{(2)} - \underline{m}^{(1)}$, amelyre $S \cap S_{\underline{m}} = \underline{0}$, tehát $\underline{u} = \underline{v} + \underline{m}$ alkalmas $\underline{u}, \underline{v} \in S$-sel. Mivel $\underline{u} = 1/2\underline{U}, \underline{v} = 1/2\underline{V}$, ahol $\underline{U}, \underline{V} \in S$, ezért $-\underline{v} \in S$ (S centrálszimmetrikus), és $1/2\underline{U} - 1/2\underline{V} = \underline{u} - \underline{v} \in S$ (S konvex), tehát $\underline{m} \in S$, így a bizonyításunk kész. ■

Befejezésül kimondjuk Minkowski tételét *általános esetben* is, amelyet nem bizonyítunk. Legyenek $\underline{a}_1, \ldots, \underline{a}_n$ az n dimenziós euklidészi térben lineárisan független vektorok, $\underline{a}_j = (a_{j_1}, \ldots, a_{j_n})$. Λ jelölje a rácspontok halmazát, azaz a

$$\sum_{j=1}^{n} u_j \underline{a}_j$$

alakú vektorok halmazát, ahol $u_j \in \mathbb{Z}$. Legyen $d(\Lambda) = d = \det a_{ij}$ ($d \neq 0$ a lineáris függetlenség miatt).

22.25. tétel. *Legyen a K korlátos, centrálszimmetrikus, konvex tartomány mértéke*

$$\lambda(K) > 2^n d(\Lambda) \ .$$

Ekkor K-ban van az origótól különböző Λ-beli pont.

22.2.3. A kvadratikus szita

A szita módszereket prímfaktorizációra használják. Hátrányuk, hogy a kis tényezőket is ugyanannyi idő alatt találják meg, mint a nagyokat, viszont nagyon gyorsan. Ezért alkalmazásuk, úgy kifizetődő, ha előbb valamely egyszerűbb módszerrel leválasztjuk a kis faktorokat. Így viszont a mai napig a legjobbnak tekinthető algoritmusok alapjául szolgálnak.

Ha x, y az $x^2 \equiv y^2 \pmod{n}$ kongruencia megoldásai, akkor n prímszám esetén az $x \equiv y \pmod{n}$, vagy $x \equiv -y \pmod{n}$, azaz $n \mid x - y$, vagy $n \mid x + y$. Ez azonban nem igaz, ha n összetett szám. Abban az esetben, ha n nem prím, az $x^2 \equiv y^2 \pmod{n}$ kongruencia teljesül olyankor is, amikor $1 < (x - y, n) < n$. Erre épül a ***kvadratikus szita módszer,*** amelynek alapötlete Fermat-tól származik.

A következő egyszerűsített algoritmust, amely a szita módszerek elődjének tekinthető, Dixon írta le. A különbség Fermat módszeréhez képest az, hogy nem feltétlenül kell olyan x, y egészeket találnunk, amelyekre $n = x^2 - y^2$. Elég olyan párt találni, amelyre $x^2 \equiv y^2 \pmod{n}$. Az algoritmusban használunk egy „megfelelő” K korlátot. K választása tulajdonképpen programozói feladat, de döntő fontosságú, hiszen túl kicsi korlát esetén csökken az esélyünk arra, hogy nemtriviális prímfaktort találjunk, túl nagy korlát esetén túl sok ellenőrzést kell elvégeznie a programnak, növelve a futásidőt. Azt mondjuk egy természetes számra, hogy ***K-sima,*** ha faktorai kisebbek, mint K. Természetesen prímfaktorizáció szempontjából azok a számok jók, amelyek kis K értékekre is K-simák.

Legyen adott $1 < r < n$ számra $g(r) = r^2 \pmod{n}$, tehát $g(r) \in [1, n]$. Választunk egy viszonylag nagy K korlátot, s véletlenszerűen választva az r számokat, kiszámítjuk a $g(r)$ értéket, és megvizsgáljuk, hogy mely $p < K$ prímek osztói $g(r)$-nek. Ezután $p^\alpha \parallel g(r)$ már könnyen számolható.

Kísérletezzünk, és gyűjtsünk össze valamennyi $r_1, \ldots, r_s \in [1, n]$ különböző egészet, amely K-sima: $g(r_j) = \prod_{i=1}^{T} p_i^{a_i(r_j)}$, ahol $(p_T \le K < p_{T+1})$ és p_l az l-edik prímszámot jelenti. Legyen

$$v(r_j) = (a_1(r_j), \ldots, a_T(r_j)) \,,$$

továbbá

$$b_l(r_j) = \begin{cases} 0, & \text{ha } a_l(r_j) \text{ páros} \,, \\ 1, & \text{ha } a_l(r_j) \text{ páratlan} \,. \end{cases}$$

Továbbá legyen

$$t(r_j) = (b_1(r_j), \ldots, b_T(r_j)) \,.$$

Tekintsük a $t(r_j)$-ket $\mathbb{Z}_2$-beli vektoroknak. $\mathbb{Z}_2$ a pusztán az additív és a multiplikatív egységelemből álló test. Ha nem okoz félreértést, $\mathbb{Z}_2$ elemeit értelemszerűen 0 és 1 jelöli. Számítsuk ki azokat a $\underline{h} = (h_1, \ldots, h_T) \in \mathbb{Z}_2^T$ vektorokat, amelyekre

$$\sum_{j=1}^{T} h_j t(r_j) = \underline{0} \,.$$

Ha $\underline{h} \neq \underline{0}$ egy megoldás, legyen

$$E(\underline{h}) = \prod_{h_j=1}^{T} g(r_j) = \prod_{h_j=1}^{T} r_j^2 \pmod{n} \,.$$

Világos, hogy $E(\underline{h})$ négyzetszám, hasonlóan a jobb oldal is. Legyen ekkor $x = \sqrt{E(\underline{h})}$, és

$y = \prod_{h_j=1}^{T} r_j$. Ezek az értékek könnyen számíthatóak.

Számítsuk ki végül a $D = (x - y, n)$ értéket. Szerencsés esetben $1 < D < n$ értéket kapunk, ami azt jelenti, hogy $D \mid n$, azaz találtunk egy nemtriviális faktort. A DIXON algoritmus bemenetként adott n pozitív egészhez próbál keresni nemtriviális prímosztót. Felhasználjuk a már ismert RANDOM(x,y) függvényt.

DIXON(n)

```
 1  j ← 0
 2  for i ← 1 to K_1
 3      do r ← RANDOM(2, n − 1)
 4         g(r) ← r^2 (mod n)
 5         if g(r) K-sima
 6            then j ← j + 1
 7                 R[j] ← g(r)
 8  if Σ_{j=1}^{T} h_j t(R[j]) = 0 nem megoldható
 9     then return "sikertelen keresés"
10  E(h) ← Π_{h_j=1}^{T} R[j]^2 (mod n), ahol h megoldás
11  x ← √E(h)
12  y ← Π_{h_j=1}^{T} R[j] (mod n)
13  D ← (x − y, n)
14  if 1 < D < n
15     then return x − y
16     else return "sikertelen keresés"
```

Természetesen sok módszer létezik a számítások meggyorsítására. Nyilvánvaló például, hogy $p \mid g(r)$ csak $(n \mid p) = 1$ esetén fordul elő, ezért $a_i(r) = 0$, ha $(n \mid p_i) = -1$ fennáll. Az r érték megválasztásával is ügyeskedhetünk. Érdemes r-t a $(k, \sqrt{2}k)$ intervallumból választani, ahol $k = \lfloor \sqrt{n} \rfloor + 1$. Nyilván ekkor $g(r) = r^2 - n$. Ha r közel van k-hoz, akkor vélhetőleg könnyebb faktorizálni a $g(r)$ számot. Montgomery a következő általánosítást javasolta. Az $r^2 - n$ polinom helyett tekintsük az

$$F(r) = ar^2 + 2br + c$$

alakú polinomokat. Ha $n = b^2 - ac$, akkor

$$aF(r) = (ar + b)^2 - n \ .$$

Világos, hogy $r = -b/a$ az $aF(r)$ minimumhelye, továbbá

$$F(-M - \frac{b}{a}) = F(M - \frac{b}{a}) = aM^2 - \frac{n}{a} \ .$$

A jobb oldal abszolút értéke adott M esetén akkor a legkisebb, ha $a \sim \sqrt{2n}/M$. A gyakorlatban a értékét ($\sqrt{2n}/M$)-hez közeli prímnek választják. Ekkor ugyanis az $x^2 \equiv n \pmod{a}$ kongruencia könnyen megoldható. Legyen b e kongruencia egy megoldása, és legyen $c = (b^2 - n)/a$. Most $g(r)$ helyett a $\tilde{g}(r) = aF(r) \pmod{n}$ számokkal (közben a, b változhat) alkalmazzuk az előbb ismertetett eljárást.

Gyakorlatok

22.2-1. Legyen $\alpha = (\sqrt{5}+1)/2$. Bizonyítsuk be, hogy α egyszerű lánctört közelítéseinél minden $1 \le i \le n$ esetén $a_i = 1$.

22.2-2. Írjunk programot, amely megadja egy racionális szám lánctört közelítéseit. Lássuk be, hogy adható olyan algoritmus, amely véges lépésben megáll.

22.2-3. Bizonyítsuk be, hogy $p \equiv 1 \pmod 4$ esetén a $p = a^2 + b^2$ egyenletnek lényegében egy megoldása van, azaz egyetlen olyan a, b pár létezik, amelyre $1 \le a < b$.

22.2-4. Bizonyítsuk be Diophantos azonosságát, miszerint

$$(a^2+b^2)(a_1^2+b_1^2) = (aa_1+bb_1)^2 + (ab_1-ba_1)^2 \,.$$

22.3. A racionális számtest algebrai bővítései

Valamely valós, vagy komplex számot algebrainak nevezünk a racionális számok teste felett, ha gyöke egy nem csupa 0-ból álló egész együtthatós

$$P(x) = a_0 + a_1 x + \cdots + a_n x^n$$

polinomnak. Természetesen valamely α algebrai számhoz több olyan $P(x)$ polinom is létezik, amelyre $P(\alpha) = 0$. Ezek közül a legalacsonyabb fokszámúak irreducibilisek. Egyetlen olyan $P(x)$ polinom van, amelyre még az a feltétel is teljesül, hogy $a_n > 0$ és $a_0, a_1, \ldots a_n$ relatív prímek. Ezt a $P(x)$ polinomot α ***minimálpolinomának*** nevezzük.

Legyenek az α szám $P(x)$ minimálpolinomának gyökei $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n$. Világos, hogy $(P(x), P'(x)) = 1$, ezért $P(x)$ gyökei egyszeresek, azaz bármely $1 \le i, j \le n$-re, ha $i \ne j$, akkor $\alpha_i \ne \alpha_j$. Az α_i számokat α ***konjugáltjainak*** nevezzük.

A szimmetrikus polinomok alaptételének ismeretében könnyen beláthatjuk, hogy az algebrai számok testet alkotnak, jelben $\overline{\mathbb{Q}}$.

Az $\alpha \in \overline{\mathbb{Q}}$ számot ***algebrai egésznek*** nevezzük, ha a minimálpolinomának főegyütthatója 1.

Legyen θ algebrai szám, $F(x) = a_0 + a_1 x + \cdots + a_n x^n$ minimálpolinommal. Tekintsük az összes

$$c_0 + c_1\theta + \cdots + c_{n-1}\theta^{n-1} \quad (c_j \in \mathbb{Q}) \tag{22.16}$$

alakú számot, azaz az összes legfeljebb $(n-1)$-edfokú $r(x)$ polinomnak a θ helyen vett helyettesítési értékét. Jelölje ezt a halmazt L. Világos, hogy a (22.16) kifejezés a 0 értéket csak akkor veheti fel, ha $c_0 = c_1 = \cdots = c_{n-1} = 0$, azaz az $1, \theta, \ldots, \theta^{n-1}$ számok bázist alkotnak.

Megmutatjuk, hogy L test. Ha r_1, r_2 legfeljebb $n-1$ fokú polinom, akkor $(r_1+r_2)(x)$ fokszáma is legfeljebb $n-1$, és így $(r_1+r_2)(\theta) \in L$. Továbbá elvégezve az

$$r_1(x)r_2(x) = q(x)F(x) + s(x)$$

maradékos osztást, azt kapjuk, hogy $\deg s \le n-1$, és $r_1(\theta)r_2(\theta) = s(\theta) \in L$.

Legyen végül $\beta = r(\theta) \in L$, és $\beta \ne 0$. Mivel $F(x)$ irreducibilis, $(r(x), F(x)) = 1$, és így

az

$$1 = a(x)r(x) + b(x)F(x)$$

egyenlet megoldható alkalmas $a(x), b(x) \in \mathbb{Q}[x]$ polinomokkal. $x = \theta$ helyettesítéssel $1 = a(\theta)r(\theta)$, azaz

$$\frac{1}{\beta} = \frac{1}{r(\theta)} = a(\theta) \ .$$

Ha $\deg a(x) \geq n$, akkor az

$$a(x) = q(x)F(x) + a_1(x)$$

maradékos osztással kapjuk, hogy $\deg a_1(x) \leq n-1$, $a(\theta) = a_1(\theta)$, ezért $1/\theta \in L$.

Tehát beláttuk, hogy L test. A továbbiakban használjuk az $L = \mathbb{Q}(\theta)$ jelölést. Világos, hogy $\mathbb{Q} \subseteq \mathbb{Q}(\theta) \subseteq \mathbb{C}$, és a $\mathbb{Q}(\theta)$-ban definiált műveletek megegyeznek a $\mathbb{C}$-ben definiált műveletekkel.

Legyen $\alpha = r(\theta)$, $\alpha_j = r(\theta_j)$ $(j = 1, 2, \ldots, n)$, és tekintsük a

$$g(x) = \prod_{j=1}^{n}(x - \alpha_j) = \prod_{j=1}^{n}(x - r(\theta)) = A_0 + A_1 x + \cdots + A_n x^n$$

polinomot. Világos, hogy az $A_\nu = A(\theta_1, \theta_2, \ldots, \theta_n)$ együtthatók a $\theta_1, \theta_2, \ldots, \theta_n$ változók szimmetrikus polinomjai, és mivel e változók elemi szimmetrikus polinomjai (lévén $P(x)$ együtthatói) racionális számok, ezért $g(x)$ n-edfokú polinom, (Nem biztos, hogy $g(x)$ irreducibilis!) ezért $\alpha(= \alpha_1)$ legfeljebb n-edfokú algebrai szám.

Legyen α, β algebrai egész. Azt mondjuk, hogy α a β osztója, ha $\beta = \alpha\gamma$, ahol γ algebrai egész. Az ε egész számot ***egységnek*** nevezzük, ha $1/\varepsilon$ is algebrai egész. Az α és β algebrai egészeket egymás ***asszociáltjának*** nevezzük, ha $\alpha = \varepsilon\beta$ teljesül, alkalmas ε egységgel.

22.3.1. Kvadratikus testek

Legyen $1 \neq d$ négyzetmentes egész szám. Ekkor

$$\mathbb{Q}(\sqrt{d}) = \{u + v\sqrt{d} \mid u, v \in \mathbb{Q}\} \ .$$

Az $\alpha = u + v\sqrt{d}$ szám konjugáltja $\overline{\alpha} = u - v\sqrt{d}$, ***normája*** $N(\alpha) = \alpha\overline{\alpha} = u^2 - v^2 d$.

Világos, hogy $\alpha, \beta \in \mathbb{Q}$ esetén $\overline{\alpha\beta} = \overline{\alpha}\,\overline{\beta}$, és így $N(\alpha\beta) = N(\alpha)N(\beta)$.

A $d = -1$ $(\sqrt{-1} = i)$ választással kapott $\mathbb{Q}(i)$ testet ***Gauss-féle számtestnek*** nevezzük. $\mathbb{Q}(i)$ (algebrai) egészei azok az $\alpha = u + vi$ alakú komplex számok, amelyek minimálpolinoma vagy elsőfokú, és ekkor $u \in \mathbb{Z}$, $v = 0$, vagy másodfokú, s az utóbbi esetben

$$(x - \alpha)(x - \overline{\alpha}) = x^2 - 2ux + u^2 + v^2$$

egész együtthatós. Könnyen belátható, hogy $2u \in \mathbb{Z}$, és $u^2 + v^2 \in \mathbb{Z}$ csak $u, v \in \mathbb{Z}$ esetén teljesülhet.

Legyen d tetszőleges, és tegyük fel, hogy $\alpha = u + v\sqrt{d}$ algebrai egész, azaz az

$$(x - \alpha)(x - \overline{\alpha}) = x^2 - 2ux + (u^2 - v^2 d)$$

polinom egész együtthatós. Legyen $a = 2u$, $b = 2v$, $c = u^2 - v^2 d$. Tehát $a, c \in \mathbb{Z}$, és fennáll, hogy $4c = a^2 - b^2 d$. Mivel $b^2 \in \mathbb{Z}$ és d négyzetmentes, ezért $b \in \mathbb{Z}$ is teljesül.

Tekintsük most azokat az eseteket, amikor $d \equiv 2 \pmod 4$, vagy $d \equiv 3 \pmod 4$. Mivel minden négyzetszám 4-gyel osztva 0, vagy 1 maradékot ad, ezért a és b csak páros szám lehet, és így $u, v \in \mathbb{Z}$.

Végül nézzük a $d \equiv 1 \pmod 4$ esetet. Nyilván $a \equiv b \pmod 2$, azaz $2(u - v) \equiv 0 \pmod 2$, tehát $u - v \in \mathbb{Z}$. Ekkor a $\mathbb{Q}(\sqrt{d})$ egészeiben szereplő u, v számok nemfeltétlenül egészek!

Eredményeinket összefoglalva kapjuk, hogy $\mathbb{Q}(\sqrt{d})$ egészeinek R_d halmaza a következő:

$d \equiv 2, 3 \pmod 4$ esetén

$$R_d = \{m + n\sqrt{d} \mid m, n \in \mathbb{Z}\} ,$$

$d \equiv 1 \pmod 4$ esetén

$$R_d = \{m + n\omega \mid m, n \in \mathbb{Z}\}, \quad \omega = \frac{1 + \sqrt{d}}{2} .$$

Végül vizsgáljuk meg az egységek E_d halmazát R_d-ben. Világos, hogy $1, -1 \in E_d$. Továbbá $\alpha \in R_d$ akkor és csak akkor egység, ha $\alpha \mid 1$, amiből $\alpha\overline{\alpha} = \pm 1$ adódik szükséges és elégséges feltételként.

Ha $d = -1$, akkor az $m^2 + n^2 = 1$ egyenlet összes megoldásából az

$$E_{-1} = \{1, -1, i, -i\}$$

halmazt kapjuk.

A $d = -3$ esetben az

$$(m + n\omega)(m + n\overline{\omega}) = m^2 + mn + n^2 = \pm 1$$

egyenlet megoldásai $(\pm 1, 0)$, $(1, -1)$, $(0, \pm 1)$, $(-1, 1)$, azaz

$$E_{-3} = \{\pm 1, \pm\omega, \pm\overline{\omega}\} .$$

Könnyű belátni, hogy $d \neq -1, -3$, $d < 0$ esetén

$$E_d = \{1, -1\} .$$

Tekintsük most a $d > 0$ esetet. Megmutatjuk, hogy ekkor végtelen sok egység van. Ehhez elég belátni, hogy létezik olyan ζ egység, amelyre $\zeta \neq \pm 1$ ugyanis $\zeta \in E_d$ esetén ζ^m $(m = \pm 1, \pm 2, \dots)$ is egység.

Tegyük fel, hogy $d > 0$, és $d \equiv 2 \pmod 4$, vagy $d \equiv 3 \pmod 4$. Alkalmazzuk Dirichlet tételét α-t $\sqrt{d}$-vel helyettesítve: tetszőleges Q-ra létezik $p, q \in \mathbb{N}$ úgy, hogy

$$\left|q\sqrt{d} - p\right| < \frac{1}{Q}, \quad 1 \leq q \leq Q ,$$

amiből kapjuk, hogy

$$0 < q\sqrt{d} + p < \frac{1}{Q} + 2q\sqrt{d} \leq 2Q\sqrt{d} + \frac{1}{Q} ,$$

ezért

$$\left|q^2 d - p^2\right| \leq 2\sqrt{d} + \frac{1}{Q^2} = C .$$

Mivel $\sqrt{d}$ irracionális, ennek az egyenlőtlenségnek végtelen sok megoldása van egész (p, q) változókban. Van tehát olyan $\mathbb{N} \in \mathbb{Z}$, amelyre $|N| \leq C$, és a

$$q^2 d - p^2 = N$$

egyenletnek végtelen sok különböző gyöke van. Világos, hogy $N \neq 0$. Van tehát olyan (p_1, q_1), (p_2, q_2) megoldáspár, amelyre $p_1 \equiv p_2 \pmod{N}$, $q_1 \equiv q_2 \pmod{N}$. Legyen $\alpha_1 = p_1 - q_1\sqrt{d}$, $\alpha_2 = p_2 - q_2\sqrt{d}$, és $\zeta = \alpha_1/\alpha_2$. Ekkor

$$\zeta = \frac{\alpha_1\overline{\alpha_2}}{\alpha_2\overline{\alpha_2}} = \frac{(p_1p_2 - q_1q_2d)}{N} + \frac{(p_1q_2 - p_2q_1d)\sqrt{d}}{N} = u + v\sqrt{d}\,,$$

és a feltételeink miatt $u, v \in \mathbb{Z}$. Továbbá

$$N(\zeta) = \frac{N(\alpha_1)}{N(\alpha_2)} = 1\,,$$

tehát ζ egység és $\alpha_1 \neq \alpha_2, \alpha_1 \neq -\alpha_2$, tehát $\zeta \neq \pm 1$. Ezért a $\pm\zeta, \pm 1/\zeta$ elemek között, amelyek mind egységek, van 1-nél nagyobb. Tekintsük az 1-nél nagyobb egységeket. Legyen $1 < \zeta = u + v\sqrt{d}$. Világos, hogy $u, v > 0$, mert $\overline{\zeta} \in (-1, 1)$, ahol $\overline{\zeta} = u + v\sqrt{d}$. Legyen ε a legkisebb 1-nél nagyobb egység, és $\kappa > 1$ egy tetszőleges másik egység. Ha $\kappa \neq \zeta^l$ semmilyen $l \in \mathbb{N}$-re, akkor $\varepsilon^m < \kappa < \varepsilon^{m+1}$, ezért $(1 <)\lambda := \kappa/\varepsilon^m < \varepsilon$ is egység, ami lehetetlen.

Az összes 1-nél nagyobb egység tehát ε^m alakú $(m = 1, 2, \dots)$, és így

$$E_d = \{\pm\varepsilon^m \mid m = 0, \pm 1, \pm 2, \dots\}\,.$$

A $d \equiv 1 \pmod{4}$ esetben E_d struktúrája hasonló. Végtelen sok egység van, amelyeket az

$$\left(x + \frac{1}{2}y\right)^2 - \frac{dy^2}{4} = \pm 1$$

egyenlet egész megoldásai szolgáltatnak.

Gyakorlatok

22.3-1. Igaz-e, hogy tetszőleges kvadratikus test algebrai egészeinek halmaza Gauss-gyűrű, azaz érvényes benne a számelmélet alaptétele? Válaszát próbálja indokolni.

22.3-2. Írjunk programot, amely egységeket szolgáltat adott kvadratikus test algebrai egészei közül.

22.3-3. Írjunk programot, amely adott valós kvadratikus testben olyan algebrai egészeket keres, amelyek abszolút értéke közel van 1-hez.

22.4. Prímszámok

A prímszámokkal kapcsolatos kérdések több, mint kétezer éve foglalkoztatják az embereket. Még mielőtt rátérnénk ezek tárgyalására, a különböző elnevezésekből adódó esetleges félreértések elkerülése végett, tekintsük a következő két definíciót.

22.26. definíció. *Legyen R gyűrű és $U(R)$ az egységek halmaza. Ekkor a $p \in R^* \setminus U(R)$ elem*
(1) ***felbonthatatlan (irreducibilis)****, ha minden $a, b \in R$ esetén $p = ab$-ből $a \in U(R)$ vagy $b \in U(R)$ következik,*
(2) ***prím****, ha minden $a, b \in R$ esetén $p \mid ab$-ből $p \mid a$ vagy $p \mid b$ következik.*

Az olyan, két binér művelettel rendelkező algebrai struktúrákat, amelyekben érvényes a ***számelmélet alaptétele (Gauss-gyűrűk),*** a prímek és felbonthatatlanok halmaza egybeesik. Mivel az egész számok halmaza is Gauss-gyűrű az összeadás és szorzás műveletével, bocsánatos bűn, hogy általános és középiskolákban a felbonthatatlan elem definícióját használják prímekre, mint ahogy azt mi is tesszük a fejezet hátralévő részében. Feltesszük továbbá, hogy ha másképp nem rendelkezünk, prímszám alatt, mindig egynél nagyobb egész számot értünk.

Azt, hogy végtelen sok prímszám létezik, már Euklidész bizonyította, viszont a mai napig nem tudjuk, hogy az ***ikerprímek*** száma véges, vagy végtelen (p_1, p_2 prímek ikerprímek, ha $|\ p_1 - p_2\ | = 2$). Nagyon valószínű, hogy az x-nél kisebb ikerprímek száma aszimptotikusan $cx/(\ln^2 x)$, alkalmas c pozitív konstanssal. Brun 1919-ben megmutatta, hogy az ikerprímek reciprokösszege konvergens sort alkot. Ez a tény arra utal, hogy ha nincsenek is feltétlenül véges sokan az ikerprímek, de egyre ritkábban fordulnak elő a végtelen felé haladva. A szám, amihez az előbb említett sor konvergál,

$$B = 1.9021605823\ldots\ ,$$

az úgynevezett ***Brun-konstans.*** B értékének, minél pontosabb meghatározása már a ***prímrekordok*** problémakörébe tartozik. Ezen a területen kerülnek terítékre olyan feladatok, mint például: keressünk minél nagyobb, bizonyos tulajdonságokkal rendelkező prímeket, vagy ellenőrizzük egy sejtés helyességét számítógéppel a lehető legnagyobb korlátig. Az egyik ilyen probléma ***páros Goldbach-sejtés*** néven vált híressé. 1742-ben egy Eulernek írott levelében vetette fel a gondolatot Goldbach, miszerint minden $n > 5$ egész szám felírható három prím összegeként. Euler válaszában megírta, hogy ez a probléma ekvivalens azzal, hogy bármely háromnál nagyobb páros egész szám felírható-e két prím összegeként. A „páros" megkülönböztetés azért ragadt a Goldbach-sejtésre, mert korábban a matematikus egy másik gondolata, amely szerint minden 5-nél nagyobb páratlan szám előáll három prím összegeként, kapta a ***páratlan Goldbach-sejtés*** elnevezést. Megjegyezzük, hogy általában a páros eset vizsgálata van napirenden, mivel ebből rögtön következik a páratlan eset. Vinogradov 1937-ben bebizonyította a páratlan Goldbach-sejtést „nagy számokra". Sajnos nagy számon, a későbbi tökéletesítések ellenére is 10 a több ezrediken nagyságrendet kell értenünk. Az a tény, hogy számítógéppel a Goldbach-sejtés 2004-ben körülbelül 10^{17} nagyságrendig ellenőrzött, jól mutatja, hogy van még mit „lefaragni" az elméleti korlátból.

A gyakran vizsgált kérdések közé tartozik még, hogy adott számig hány prím fordul elő, mekkora a legnagyobb távolság (***hézag***) két szomszédos prím között. Az ikerprímekhez hasonlóan, bizonyos szabályok szerint, 3-as, 4-es stb. minták is létrejöhetnek prímszámokból. Ezek előfordulásainak megkeresése, illetve reciprokösszegük nagy pontosságú kiszámolása a számítógépes számelmélet témakörébe tartozó feladat.

Talán a két legfontosabb prímszámokkal kapcsolatos fogalom a ***faktorizáció*** és a bevezetésben már említett ***prímteszt.*** Előbbi azt jelenti, hogy adott számot megpróbálunk felbonthatatlanok szorzataként felírni. Meglepő lehet, hogy a két probléma megoldásának bonyolultsága mennyire eltérő. Míg prímteszt eredményesen végezhető több ezer jegyű számokra is, addig a prímfaktorizációt legfeljebb 150-200 jegyű számokra hajthatjuk végre, legalábbis ez a helyzet 2004-ben. A legmodernebb titkosítási eljárások is azon alapulnak, hogy nagy számokat prímek szorzatára bontani „nehéz" feladat.

Sokszor merül fel az igény arra, hogy nagy számú prímet kell előállítanunk, például a Goldbach-sejtés ellenőrzésénél, vagy prímek közti hézagok vizsgálatánál. Erre a mai napig

is talán a legalkalmasabb módszer egy többezer éve ismert eljárás, amely ***eratosztenészi szita*** néven vált ismertté. Az algoritmus valamely N bemenő értékre megadja 3-tól kezdve az N-nél nem nagyobb prímszámokat. Az eredmény a $Tábla[j]$ vektorban képződik úgy, hogy ha a j-edik elem 1, akkor $2j + 3$ prímszám.

ERATOSZTENÉSZ(N)

```
 1  n ← ⌊(N − 1)/2⌋
 2  j ← 0
 3  for i ← 1 to n
 4       do Tábla[i] ← 1
 5  while Tábla[j] = 0
 6          j ← j + 1
 7  i ← 2j² + 6j + 3
 8  if i ≥ n
 9    then return Tábla
10  Tábla[i] ← 0
11  i ← i + 2j + 3
12  if i ≥ n
13    then j ← j + 1
14         goto 5
15    else goto 10
```

22.4.1. Alapok

Ha valamely $n \in \mathbb{N}$ szám összetett, akkor felírható $n = ab$ alakban, ahol $a, b \in \mathbb{N}$, továbbá $a > 1$, $b > 1$, és így

$$\min(a, b) \leq \sqrt{(n)} \ .$$

Annak eldöntését, hogy n prím vagy összetett, elvégezhetjük a következő módon, egy korai algoritmus segítségével:

ELSŐ-PRÍMTESZT(n)

```
1  for d ← 2 to ⌊√(n)⌋
2       do if d | n
3            then return ÖSSZETETT
4  return PRÍM
```

Látható, hogy ez az algoritmus a gyakorlati életben nagy számokra használhatatlan, annak ellenére, hogy elméletileg faktorizálásra is használható, hiszen ha n összetett a d változóban előáll az első prímfaktor. Mivel a prímtesztelő algoritmusoknak külön fejezetet szentelünk, ezért itt csak azok megértéséhez szükséges alapfogalmakat és állításokat ismertetünk. Ilyen például egy jól ismert elméleti kritérium, a ***Wilson-féle kongruencia tétel***, mely szerint:

22.27. tétel. *Az*

$$(n-1)! + 1 \equiv 0 \pmod{n}$$

kongruencia akkor és csak akkor teljesül, ha n prímszám.

Bizonyítás. Először tegyük fel, hogy $n \in \mathbb{N}$ összetett. Ekkor van van olyan $d \in \mathbb{N}$ osztója, amelyre $1 < d < n$, és így $d \mid (n-1)!$. Tehát $d \nmid (n-1)! + 1$ fennáll, következésképpen $n \nmid (n-1)! + 1$, azaz kaptuk, hogy

$$(n-1)! + 1 \not\equiv 0 \pmod{n}\,.$$

Most legyen n prímszám. Ha $n = 2$, akkor

$$(2-1)! + 1 \equiv 0 \pmod{n}$$

fennáll, tehát feltehetjük a továbbiakban, hogy $n > 2$. Tudjuk, hogy az

$$ax \equiv 1 \pmod{n}$$

kongruenciának pontosan egy megoldása van, ha $(a,n) = 1$. Ez n prím volta miatt teljesül minden $1 \leq a \leq n-1$ esetén. Jelölje az egyetlen megoldást $a^{-1} \pmod{n}$. Vegyük észre, hogy

$$a \equiv a^{-1} \pmod{n}$$

csak az $a \equiv 1 \pmod{n}$ és $a \equiv n-1 \pmod{n}$ esetben fordulhat elő. Mivel

$$\prod_{a=1}^{n-1} a \equiv \prod_{a=1}^{n-1} a^{-1} \pmod{n}$$

fennáll, és a bal, illetve jobb oldalon lévő tényezőkkel, kivéve $a = 1$ és $a = n-1$ értékeket, egyszerűsíthetünk. Tehát kapjuk, hogy

$$(n-1)! \equiv n-1 \equiv -1 \pmod{n}\,,$$

amivel az állítást bizonyítottuk. ■

A következő tétel ***Lucas-tesztként*** ismert.

22.28. tétel. *Az $n \in \mathbb{N}$ szám akkor és csak akkor prím, ha létezik olyan $g \in \mathbb{N}$, amire $(n,g) = 1$ és*

$$g^{n-1} \equiv 1 \pmod{n}\,,$$

továbbá minden olyan p prímre, amelyre $p \mid n-1$, fennáll a

$$g^{\frac{n-1}{p}} \not\equiv 1 \pmod{n}\,.$$

kongruencia.

Bizonyítás. Tegyük fel, hogy n prím. Ekkor $\mathbb{Z}_n$ test, tehát $\mathbb{Z}_n^*$ ciklikus csoport, azaz létezik benne g primitív elem, melynek rendje $n-1$. Ekkor nyilvánvalóan teljesül

$$g^{\frac{n-1}{p}} \not\equiv 1 \pmod{n} .$$

Most tegyük fel, hogy $(n,g)=1$, $g^{n-1} \equiv 1 \pmod{n}$ és

$$g^{\frac{n-1}{p}} \not\equiv 1 \pmod{n}$$

Minden $p \mid n-1$, p prím esetén. Ekkor $g \in \mathbb{Z}_n^*$ és a $\{g, g^2, \ldots, g^{n-1}\} \subseteq \mathbb{Z}_n^*$ elemek mind különbözőek, azaz $n-1 \leq \left|\mathbb{Z}_n^*\right| = \varphi(n)$, és így $\varphi(n) = n-1$. Ez pontosan azt jelenti, hogy n prím. ■

Tehát észrevehetjük, hogy $a^{n-1} \equiv 1 \pmod{n}$ fennáll minden $(a,n)=1$ választás esetén, ha n prím (tulajdonképpen ezt az állítást fogalmazza meg az úgynevezett ***kis Fermat-tétel***). Viszont ha n összetett, „megbukik" a teszten, ezért mondhatjuk, hogy a Lucas-teszt tételre alapozott LUCAS-PRÍMTESZT determinisztikus prímteszt, amelynek ismertetésénél FAKTOR(x,i) az x pozitív egész i-edik prímosztóját adja, azaz, ha $x = p_1^{\alpha_1} \ldots p_k^{\alpha_k}$, akkor p_i-t $(1 \leq i \leq k)$, továbbá NUMFAKTOR(x) a prímosztók számát jelenti, esetünkben k-t. Mivel sok megfelelő a szám létezik, ezért megkeresése a gyakorlatban nem jelent problémát, itt K-val jelöljük azt a korlátot, amelynél legkésőbb megtalálja az algoritmus a kívánt a értéket. A programban a *rossz* változó mutatja, hogy ha az a érték nem megfelelő.

LUCAS-PRÍMTESZT(n,m)

```
 1  rossz ← HAMIS
 2  a ← 1
 3  if a^(n−1) ≢ 1 (mod n)
 4     then a ← a + 1
 5          goto 3
 6  for i ← 1 to NUMFAKTOR(n − 1)
 7      if a^((n−1)/FAKTOR(n−1)) ≡ 1 (mod n)
 8         then rossz ← IGAZ
 9              a ← a + 1
10  if a > K
11     then if rossz
12             then return ÖSSZETETT
13             else return PRÍM
14  if rossz
15     then goto 3
16     else return PRÍM
```

Ennek a módszernek a fő problémája az, hogy működéséhez szükségünk van $n-1$ faktorizációjára, ami nagyon megnöveli a műveletigényt. Éppen ezért a Lucas-tesztet csak olyan speciális n számok esetén célszerű használni, ahol $n-1$ prímtényezős felbontását könnyű elvégezni. Ilyen speciális számok például a ***Fermat-számok,*** amelyek a 2^k+1 alakú számok. Jó példa az úgynevezett ***Pépin-teszt,*** amely azon az észrevételen alapszik, hogy ha

$2^n + 1$ prím, akkor $n = 2^m$ alakú, ahol n, m pozitív egészek. Ez nyilvánvaló, hiszen

$$2^{q2^m} + 1 = \left(2^{2^m} + 1\right)\left(2^{2^m(q-1)} - 2^{2^m(q-2)} + \cdots + 2^0\right) .$$

$m \geq 2$ esetén $F_m = 2^{2^m} + 1$ minden q prímosztója $k2^{m+2} + 1$ alakú, mivel

$$2^{2^m} \equiv -1 \pmod{q} ,$$

valamint

$$2^{q-1} \equiv 1 \pmod{q} ,$$

ami azt jelenti, hogy $2^{m+1} \mid q - 1$, azaz $q = k2^{m+1} + 1$. Tudjuk továbbá, hogy

$$2^{(q-1)/2} \equiv \left(\frac{2}{q}\right) = (-1)^{(q^2-1)/8} = 1 ,$$

ha $m \geq 2$, amiből kapjuk, hogy

$$2^{m+1} \mid (q-1)/2 \text{ azaz } q = k2^{m+2} + 1 .$$

Tehát a továbbiakban kereshetjük a Fermat-prímeket $2^{2^m} + 1$ alakban a következő állítás segítségével, melynek bizonyítását nem közöljük.

22.29. tétel. *Ha F_m jelöli a $2^{2^m} + 1$ alakú Fermat-számot, akkor $m > 0$ esetén F_m akkor és csak akkor prím, ha*

$$3^{(F_m-1)/2} \equiv -1 \pmod{F_m} .$$

PÉPIN-PRÍMTESZT(m)

```
1  F ← 2^(2^m) + 1
2  if 3^((F−1)/2) ≡ −1 (mod F)
3     then return PRÍM
4     else  return ÖSSZETETT
```

A kis Fermat-tételre alapozva születtek gyorsabb futási idejű valószínűségi prímtesztek. Ezeknek az elve az, hogy $n > 2$ egész számra, ha

$$2^{n-1} \not\equiv 1 \pmod{n} ,$$

akkor n összetett. A

$$2^{n-1} \equiv 1 \pmod{n}$$

esetben nem mondhatjuk biztosan, hogy n prím. A gyakorlatban a valószínűségi tesztek jól használhatónak bizonyultak gyorsaságuk miatt, és amiatt, hogy kevés olyan összetett szám van, ami „átmegy” a teszten prím minősítéssel. Ezek a nem prímszámok, amelyek mégis prímként viselkednek bizonyos szituációkban, elrontva például prímtesztek determinisztikusságát, fontos szerepet játszanak a számelméleti kutatásokban, így nevesítjük is őket.

22.30. definíció. *Az n páratlan összetett számot **a alapú pszeudoprímnek**, vagy **álprímnek** nevezzük, ha*

$$a^{n-1} \equiv 1 \pmod{n}$$

fennáll.
*Az n összetett számot **Carmichael-számnak,** vagy **univerzális álprímnek** nevezzük, ha minden $(a,n) = 1$ esetben fennáll az*

$$a^{n-1} \equiv 1 \pmod{n}$$

kongruencia.

Az alábbi állítást, bár könnyen belátható, bizonyítás nélkül közöljük:

22.31. tétel. *Az $n \in \mathbb{N}$ összetett páratlan szám akkor és csak akkor Carmichael-szám, ha négyzetmentes, vagyis az $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_r^{\alpha_r}$ prímtényezős felbontásban $\alpha_1 = \alpha_2 \cdots = \alpha_r = 1$ és $p_i - 1 \mid n - 1$ $(i = 1, 2, \dots, r)$ teljesül.*

Carl Pomerance bizonyította az állítás, miszerint végtelen sok Carmichael-szám létezik.

22.2. példa. Az $n = 91 = 7 \cdot 13$ szám 3 alapú pszeudoprím, de nem pszeudoprím a 2 alaphoz. Az $n = 561 = 3 \cdot 11 \cdot 17$ Carmichael-szám.

22.4.2. A prímszámok eloszlása

Ebben a szakaszban néhány néhány prímszámokkal kapcsolatos alapvető tételt ismertetünk, bizonyítás nélkül.

22.32. tétel (Dirichlet). *Legyen $k > 0$, l egészek és $(k,l) = 1$. Ekkor az $n + lk$, $l = 0, 1, 2, \dots$ számtani sorozat végtelen sok prímet tartalmaz.*

A továbbiakban a szokásoknak megfelelően jelöljük $\pi(x)$-szel az x-nél nem nagyobb pozitív prímek számát és $\pi(x,k,l)$-lel azoknak a $p \le x$ prímek számát, amelyekre $p \equiv l \pmod{k}$ teljesül. Az úgynevezett ***prímszámtétel*** segítségével $\pi(x)$ értékét becsülhetjük.

22.33. tétel.

$$\lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\ln x}} = 1 \ .$$

A prímszámtétel segítségével aszimptotikusan becsülhetjük az n-edik prímszám értékét is. Jelölje p_n az n-edik prímszámot, ekkor

$$\lim_{x\to\infty} \frac{p_n}{n \ln n} = 1 \ .$$

Igaz továbbá, hogy

$$\lim_{x\to\infty} \frac{\pi(x,k,l)}{\frac{x}{\ln x}} = \frac{1}{\varphi(k)} \ .$$

A prímszámtételt egymástól függetlenül Hadamard és de la Vallée Poussin bizonyította 1896-ban. A fiatal Gauss már a *XVIII.* század végén használta a

$$Li(x) = \int_2^x \frac{du}{\ln u}$$

formulát $\pi(x)$ közelítésére. Ismeretes, hogy alkalmas C és A, illetve C_k és A_k állandókkal

$$|\pi(x) - Li(x)| \le C \cdot x \cdot e^{-A\sqrt{\ln x}} ,$$

illetve

$$\left|\pi(x,k,l) - \frac{Li(x)}{\varphi(x)}\right| \le C_k \cdot x \cdot e^{-A_k\sqrt{\ln x}} .$$

Az $\sqrt{\ln x}$ érték tovább finomítható. A prímszámok eloszlásának elméletében alapvető jelentőségű az úgynevezett Riemann-féle ζ függvény, amelyet komplex s értékekre értelmezünk, a $\Re(s) > 1$ félsíkban egyszerűen a

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

abszolút konvergens sorral. A $\zeta(s)(s-1)$ függvényt analitikusan folytatni lehet az egész komplex síkra.

Az a kérdés, hogy a prímszámtételt milyen maradéktaggal tudjuk igazolni, lényegében attól függ, hogy a ζ függvényre „milyen nagy" gyökmentes tartományt tudunk megadni. Riemann egy máig bebizonyítatlan sejtése szerint a $\zeta(s)$-nek nincs gyöke a $\Re(s) > 1/2$ félsíkban. Ha ez igaz lenne, akkor innen a

$$|\pi(x) - Li(x)| < C_\epsilon \cdot x^{\frac{1}{2}+\epsilon}$$

egyenlőtlenség következne, tetszőleges $\epsilon > 0$ és alkalmas C_ϵ állandókkal. A ***Riemann-sejtés*** bizonyítása a számelmélet, sőt az egész matematika alapvető jelentőségű eredménye lenne. Ez azonban eddig a legnagyobb matematikusok erőfeszítéseinek is ellenállt.

A ζ függvény a $\Re(s) > 1$ félsíkban előállítható végtelen szorzat alakban is:

$$\zeta(s) = \prod_p \left(1 + \frac{1}{p^s} + \frac{1}{p^{2s}} + \cdots\right) = \prod_p \frac{1}{1 - \frac{1}{p^s}} ,$$

ahol p a prímszámokon fut végig. Ez a formula egyszerűen levezethető a számelmélet alaptételéből, vagyis abból, hogy minden $n \in \mathbb{N}$ a sorrendtől eltekintve egyértelműen írható fel prímhatványok szorzataként.

22.4.3. Mersenne-prímek, tökéletes számok

A 2^n-1 alakú számokat ***Mersenne-számoknak*** nevezzük, jelben $M(n) = 2^n-1$. A következő összefüggés nyilvánvalóan teljesül $n = dk$ esetén:

$$2^n - 1 = \left(2^d - 1\right)\left(2^{d(k-1)} + 2^{d(k-2)} + \cdots + 1\right) ,$$

azaz minden $d \mid n$-re igaz, hogy $M(d) \mid M(n)$. Az eddigiekből rögtön adódik, hogy ha $(n,m) = d$, akkor $M(d) \mid (M(n), M(m))$, továbbá $M(n)$ csak akkor lehet prím, ha n prím.

Könnyen látható, hogy $(n,m) = 1$ esetén $M(d) \mid (M(n), M(m)) = 1$. Legyen $p \mid M(n)$, és t az a legkisebb pozitív egész, amelyre $2^t \equiv 1 \pmod{p}$. Nyilvánvaló, hogy $t > 1$ és $t \mid n$. Ekkor $p \mid (M(n), M(n))$ esetén $t \mid (n,m)$ következne, ami lehetetlen, ha $(n,m) = 1$.

Vegyük észre, hogy n prím mivolta nem elégséges feltétele annak, hogy $M(n)$ prím legyen. Például az $M(2) = 3$, $M(3) = 7$, $M(5) = 31$, $M(7) = 127$ számok prímek, de az $M(11) = 2047 = 23 \cdot 89$ összetett. Nyitott kérdés, hogy létezik-e végtelen sok olyan p prím, amelyre $M(p)$ prímszám, azaz létezik-e végtelen sok Mersenne-prím. Ennek eldöntése a számelmélet tudományának jelenlegi szintjén reménytelennek látszik, ugyanakkor igen nagy p értékekre hatékonyan eldönthető, hogy $M(p)$ prím, vagy összetett. Erre a célra szolgál a klasszikus ***Lucas–Lehmer-teszt.***

22.34. tétel. *Legyen p prím, $s_1 = 4$, $s_k = s_{k-1}^2 - 2$, $k = 1, 2, \ldots$. Az $M(p)$ szám akkor és csak akkor prím, ha $M(p)$ az s_{p-1} osztója.*

Bizonyítás. Először néhány segédtételt és észrevételt közlünk. Legyen $\alpha = 1 + \sqrt{3}$, $\overline{\alpha} = 1 - \sqrt{3}$, továbbá

$$E_n := \frac{\alpha^n - \overline{\alpha}^n}{\alpha - \overline{\alpha}}, \quad F_n := \alpha^n + \overline{\alpha}^n \ .$$

Nyilvánvaló, hogy $\alpha + \overline{\alpha} = 2$ és $\alpha\overline{\alpha} = -2$. A binomiális tételből következik a (0) észrevétel, a továbbiak bizonyítását az olvasóra bízzuk. Tehát k, l pozitív egészekre:

$$(0) \quad E_n = \binom{n}{1} + \binom{n}{3}3 + \binom{n}{1}3^2 + \ldots, \quad F_n = 2\{1 + \binom{n}{2}3 + \binom{n}{4}3^2 + \ldots\} \ ,$$

$$(1) \quad 2E_{k+l} = E_k F_l + F_k E_l \ ,$$

$$(2) \quad (-2)^{l+1} E_{k-l} = E_l F_k + E_k F_l \ ,$$

$$(3) \quad E_{2k} = E_k F_k \ ,$$

$$(4) \quad F_{2k} = F_k^2 + (-2)^{k+1} \ ,$$

$$(5) \quad F_k^2 - 12E_k^2 = (-2)^{k+2} \ ,$$

$$(6) \quad 2F_{k+l} = F_k F_l + 12E_k E_l \ .$$

Adott páratlan q prímre jelölje $\omega(q)$ azt az n legkisebb pozitív egész számot, amelyre $q \mid E_n$. Ha nincs ilyen n, akkor $\omega(q) = \infty$.

22.35. lemma. *Legyen $S = \{\nu \in \mathbb{N},\ q \mid E_\nu\}$. Ekkor*

$$S = \{k\omega(q) \mid k \in \mathbb{N}\} \ .$$

Bizonyítás. Ha $k, l \in S$, akkor (1),(2) miatt $k + l$, és $k > l$ esetén $k - l \in S$. Ezért $\omega(q)$ többszörösei S-hez tartoznak. Tegyük fel indirekt módon, hogy létezik $m \in S$, amelyre $\omega(q) \nmid m$. Ekkor $m = l\omega(q) + r$, $0 < r < \omega(q)$, és $m \in S$, $l\omega(q) \in S$ miatt

$$m - l\omega(q) = r \in S \ .$$

Ez ellentmond $\omega(q)$ definíciójának. ∎

22.36. lemma. *Ha $q > 3$ prím, akkor*

$$(7) \quad q \mid E_q - 3^{\frac{q-1}{2}},$$

és

$$(8) \quad q \mid E_q - 2.$$

Bizonyítás. (0) és $q \mid \binom{q}{j}$ $1 \le j \le q-1$ miatt

$$E_q \equiv \binom{q}{q} 3^{\frac{q-1}{2}} \equiv 3^{\frac{q-1}{2}} \pmod{q}$$

valamint

$$F_q \equiv 2 \pmod{q} .$$

■

22.37. lemma. *Ha $q > 3$ prím, és $\omega(q)$ létezik, akkor*

$$\omega(q) \le q + 1 .$$

Bizonyítás. $E_1 = 1$, $E_2 = 2$. Alkalmazzuk az (1), (2) egyenlőségeket $k = q$ és $l = 1$ választással. Ekkor $2E_{q+1} = 2E_q + F_q$, $-4E_{q-1} = 2E_q - F_q$, azaz

$$-8E_{q-1}E_{q+1} = 4E_q^2 - F_{q^2} \equiv 4 \cdot 3^{q-1} - 4 \pmod{q} \equiv 0 \pmod{p} ,$$

s ezért

$$q \mid E_{q-1}E_{q+1} ,$$

amivel a lemma bizonyítását befejeztük. ■

22.38. lemma. *Ha $p \equiv 7 \pmod{12}$, akkor $\left(\frac{3}{p}\right) = -1$, és így*

$$p \mid 3^{\frac{p-1}{2}} + 1 .$$

Bizonyítás. Legyen $p \equiv 7 \pmod{12}$. Ekkor

$$\left(\frac{3}{p}\right) = (-1)^{\frac{p-1}{2} \cdot \frac{3-1}{2}} \left(\frac{p}{3}\right) = -1 \cdot \left(\frac{p}{3}\right) = -1 .$$

■

Most rátérünk a tétel *elegendőségének* bizonyítására. Legyen p páratlan prím, és $M(p) \mid s_{p-1}$. Ekkor

(9) $M(p) \mid 2^{2^{p-2}} \cdot s_{p-1}$.

Belátjuk, hogy $F_{2^n} = 2^{2^{n-1}} \cdot s_n$ $(n = 1, 2, \dots)$. Mivel $2s_1 = F_2$, ez igaz, ha $n = 1$. Tegyük fel, hogy ezt beláttuk már minden $n > 1$-re. Mivel $s_{n+1} = s_n^2 - 2$, ezért

$$2^{2^n} \cdot s_{n+1} = (2^{2^{n-1}} \cdot s_n)^2 - 2^{2^n+1} .$$

(4) miatt $k = 2^n$-re:

$$F_{2^{n+1}} = F_{2^n}^2 - 2^{2^n+1} ,$$

ezért

$$F_{2^{n+1}} = 2^{2^n} \cdot s_{n+1} \ ,$$

így $n = p-1$-re

(10) $2^{2^{p-2}} \cdot s_{p-1} = F_{2^{p-1}}$.

Ekkor (9) és (10) miatt

(11) $M(p) \mid F_{2^{p-1}}$,

ahonnan (3)-at $k = 2^{p-1}$re alkalmazva kapjuk, hogy

(12) $M(p) \mid E_{2^p}$.

Legyen most q prím, $q \mid M(n)$. Ekkor $q \neq 2, 3$, tehát $q > 3$. (12) miatt $q \mid E_{2^p}$. Ekkor 22.35. lemma miatt $\omega(q) \mid 2^p$. Ha $\omega(q) \neq 2^p$ teljesülne, akkor $\omega(q) \mid 2^{p-1}$ lenne, és így $q \mid E_{2^{p-1}}$ fennállna. Ekkor (11) és (5) miatt q valamely 2-hatvány osztója lenne, ami nem lehet. Tehát $\omega(q) = 2^p$. A 22.37. lemma miatt $2^p \leq q+1$, azaz $q \geq 2^p - 1 = M(p)$, $q \mid 2^p - 1$, és így $q = M(p)$, amivel az elégségességet beláttuk.
A tétel *szükségességének* bizonyításához legyen $p > 2$, $M(p) = q$ prím. Ekkor $8 \mid 2^p = q+1$, $q \equiv 7 \pmod 8$. Mivel p páratlan, ezért

$$q = 2^p - 1 \equiv 2 - 1 \equiv 1 \pmod 3 \ ,$$

és így $q \equiv 7 \pmod{24}$, azaz $q = 24r + 7$ alakban írható alkalmas $r \in \mathbb{N}_0$-ra. Alkalmazzuk (4)-et $k = 2^{p-1}$ választással. Ekkor

(13) $F_{2^p} = F_{2^{p-1}}^2 - 4 \cdot 2^{2^{p-1}-1}$

Mivel $q = 24r + 7 = 8 \cdot 3r + 7$, ezért

$$q \mid M\left(\frac{q-1}{2}\right) = 2^{\frac{q-1}{2}} - 1 \ .$$

Ez a

$$\left(\frac{2}{q}\right) = (-1)^{\frac{q-1}{2}} = 1$$

és a

$$2^{\frac{q-1}{2}} \equiv \left(\frac{2}{q}\right) \pmod q$$

formulából azonnal következik. Mivel $M\left(\frac{q-1}{2}\right) = M\left(2^{p-1} - 1\right)$, ezért $q \mid 2^{2^{p-1}-1} - 1$, ahonnan

(14) $q \mid F_{2^p} - F_{2^{p-1}}^2 + 4$.

(6)-ot $k = q$ és $l = 1$ választással alkalmazva, $q + 1 = 2^p$ miatt

$$2F_{2^p} = F_q F_1 + 12E_q E_1 = 2F_q + 12E_q \ .$$

Tehát

$$F_{2^p} = F_q + 6E_q = (F_q - 2) + 6(E_q + 1) - 4. \tag{15}$$

A 22.38. lemma miatt $q \mid 3^{\frac{q-1}{2}} + 1$, továbbá (7) miatt $q \mid E_q + 1$, (8) miatt $q \mid F_q - 2$. Ezért (15) miatt $q \mid F_{2^p} + 4$, és így (14) miatt $q \mid F^2_{2^{p-1}}$. (10) miatt, $q = M(p)$ páratlan volta miatt $M(p) \mid s_{p-1}$. A szükségességet beláttuk, ezáltal a tételt is. ■

Tekintsük most az algoritmust, aminek a bemenete egy $m > 2$ páratlan egész szám.

LUCAS–LEHMER-PRÍMTESZT(m)

```
1  M ← 2^m − 1
2  v ← 4
3  for i ← 1 to m − 2
4       do v ← v^2 − 2
5  if v ≡ 0 (mod M)
6     then return PRÍM
7     else return ÖSSZETETT
```

Már Euklidész is foglalkozott az úgynevezett ***tökéletes számokkal.*** Valamely n pozitív egészet tökéletesnek nevezünk, ha az osztóinak összege $2n$. Világos, hogy $n = 6$ tökéletes szám, mivel osztói az $1, 2, 3, 6$ számok, és $2 \cdot 6 = 1 + 2 + 3 + 6$. Ismeretes, hogy ha n prímtényezős felbontása $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_r^{\alpha_r}$, akkor

$$\sigma(n) = \sigma(p_1^{\alpha_1})\sigma(p_2^{\alpha_2}) \dots \sigma(p_r^{\alpha_r}) \ ,$$

ahol $\sigma(n)$ n osztóinak összegét jelenti. Tudjuk még továbbá, hogy

$$\sigma(p^\alpha) = 1 + p + p^2 + \dots + p^\alpha \ ,$$

valamely p prímszámra.

Érdekességként megemlítünk két egyszerűnek tűnő, de több ezer éve megválaszolatlan kérdést. Mivel eddig egyetlen páratlan tökéletes számot sem találtak, sejthető, hogy nincs is, de ez nem bizonyított. A következő tétel, amelyet Euler bizonyított Euklihttp://aszt.inf.elte.hu/ hunlaci/dész eredményeire támaszkodva, mutatja, hogy a Mersenne-prímek száma megegyezik a páros tökéletes számok számával. Következésképpen, a tudomány jelenlegi állása szerint, nem tudjuk megmondani, hogy létezik-e végtelen sok páros tökéles szám, avagy sem.

22.39. tétel. *Az n páros szám akkor és csak akkor tökéletes, ha felírható*

$$n = 2^{p-1} M(p)$$

alakban, ahol p és $M(p)$ prím.

Bizonyítás. Először tegyük fel, hogy $q = M(p)$ prím, $n = 2^{p-1}q$. Ekkor

$$\sigma(n) = \left(1 + 2 + \cdots + 2^{p-1}\right)(1 + q) = (2^p - 1) \cdot 2^p = 2n \ .$$

Most fordítva, tegyük fel, hogy n páros és tökéletes: $n = 2^{s-1}l$, ahol $s > 1$. Vezessük be a következő jelölést:

$$\tilde{\sigma}(n) = \sigma(n) - n = \sum_{d|n,\ d<n} d \ .$$

Ekkor

$$2n = \sigma(2^{s-1})\sigma(l) = (2^{s-1})\sigma(l)$$

miatt $2^s \parallel \sigma(l)$, továbbá

$$(2^s - 1)(l + \tilde{\sigma}(l)) = 2^s l$$

miatt $(2^s - 1)\tilde{\sigma}(l) = l$, azaz $2^s - 1 \mid l$, és $\tilde{\sigma}(l) \mid l$, és így

$$\sigma(l) \geq (2^s - 1) + \tilde{\sigma}(l) + l + 1 \ ,$$

fennáll, kivéve azt az esetet, ha $\tilde{\sigma}(l) = 1$, amikor l prím és $l = 2^s - 1$. Ekkor $s = p$ prímszám, $l = M(p)$, és így kaptuk, hogy $n = 2^{p-1}M(p)$. ■

Gyakorlatok

22.4-1. Írjunk programot, amely adott n pozitív egész számig előállítja a prímeket. Használjuk az eratosztenészi szitát, és próbáljuk meggyorsítani a „kis prímekkel” való szitálást.

22.4-2. Bizonyítsuk be, hogy végtelen sok prímszám van.

22.5. Az AKS algoritmus

Az esetleges félreértések elkerülése végett teszünk néhány megjegyzést a fejezetben használt jelölésekkel kapcsolatban. lg minden esetben kettes alapú logaritmust jelöl. A bonyolultságelméletből ismert fogalmakat a szokásos módon értelmezzük. A szakirodalomban algoritmusok vizsgálatánál általában n bites bemenetet tételeznek fel és n függvényében vizsgálják a futási időt. Prímtesztek esetében viszont, ha n pozitív egész a vizsgálandó szám, akkor ábrázolásához $\lceil \lg n \rceil$ bitre van szükség, tehát a bonyolultsági korlátok $\lceil \lg n \rceil$ függvényeiben értendőek. Használni fogjuk továbbá a $O^{\sim}(t(n))$ kifejezést $O(t(n) \cdot poly(\lg t(n)))$ jelölésére, ahol $t(n)$ tetszőleges függvénye n-nek, és $poly(\lg t(n))$ polinomiális függvénye $\lg t(n)$-nek, azaz

$$poly(\lg t(n)) = a_0 + \lg t(n) + a_1 \lg^2 t(n) + \cdots + a_r \lg^r t(n) \ ,$$

valamely r pozitív egészre, és $a_0, a_1, \ldots, a_r$ valós együtthatókra. Például, ha t éppen a lg függvénnyel egyenlő, akkor

$$O^{\sim}(\lg^k(n)) = O(\lg^k(n) \cdot poly(\lg \lg n)) = O(\lg^{k+\varepsilon} n)$$

tetszőleges pozitív ε-ra.

Adott $r \in \mathbb{N}$ és $a \in \mathbb{Z}$ esetén, ha $(a, r) = 1$, akkor azt a legkisebb k pozitív egészet, amire $a^k \equiv 1 \pmod{r}$, *a r **szerinti rendjének*** nevezzük, és $o_r(a)$-val jelöljük. Tudjuk, hogy a fenti feltételek mellett $o_r(a) \mid \varphi(r)$ teljesül, ahol φ az Euler-féle függvényt jelöli.

Agrawal, Kayal és Saxena 2002-ben közöltek egy algoritmust [8]-ben, amely nagy áttörésnek számít a prímtesztek történetében. Mondhatjuk ezt azért, mert a szerzők bizonyítják, hogy a prímszámok halmaza a P nyelvosztályba tartozik. Ez azt jelenti, hogy egy n pozitív egész szám prím mivoltának eldöntése megoldható annyi idő alatt, amely $\lceil \lg n \rceil$-nek polinomiális függvénye. Ráadásul ez az algoritmus determinisztikus, vagyis a valószínűségi tesztekkel ellentétben itt a legkisebb esélye sincs annak, hogy valamely álprímet véletlenül prímnek nyilvánítsunk.

Alapötlet

A teszt prímszámok azonosításának egy olyan ötletén alapszik, amely tulajdonképpen a kis Fermat-tétel általánosítása. A következő segédtétel állításait a szerzők sikerrel alkalmazták már polinomiális futási idejű, nem determinisztikus prímtesztek esetén.

22.40. lemma. *Legyen $a, n \in \mathbb{Z}$, melyekre $n \geq 2$ és $(a, n) = 1$ fennáll. Ekkor n akkor és csak akkor prím, ha*

$$(x + a)^n \equiv x^n + a \pmod{n} . \tag{22.17}$$

Bizonyítás. Tehát a feladatunk az $f(x) = (x+a)^n - (x^n + a)$ polinom együtthatóinak vizsgálata (mod n). Rögtön adódik, hogy $f(x)$ konstans tagja, illetve x^n együtthatója 0, továbbá $0 < i < n$ esetén minden x^i a polinomiális tétel értelmében $\binom{n}{i} a^{n-i}$-nel szorzódik. Az

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

kifejezés mindig egész számot szolgáltat, tehát $i!(n-i)!$ osztója $n!$-nak.

Most tegyük fel, hogy n prímszám. Ekkor n nem osztója $i!$ és $(n-i)!$ egyetlen tényezőjének sem, tehát $(n, i!(n-i)!) = 1$, ami azt jelenti, hogy $i!(n-i)!$ osztója $(n-1)!$-nak. Kaptuk, hogy minden $\binom{n}{i}$ kifejezés alkalmas $b \in \mathbb{Z}$ elemmel az nb alakban írható fel, ami kongruens $0 \pmod{n}$.

Most azt az esetet vizsgáljuk, amikor n összetett. Bontsuk fel n-et prímhatványok szorzatára és válasszunk a tényezők közül egy p^k számot, amire $p^k \parallel n$. Ekkor p^k relatív prím a^{n-q}-hoz és nem osztója $\binom{n}{q}$-nak. Ez azt jelenti, hogy $f(x)$-ben az x^q-os tag együtthatója nem osztható n minden faktorával, azaz n-nel sem, így

$$\binom{n}{q} a^{n-q} \not\equiv 0 \pmod{n} .$$

Tehát, ha n nem prímszám, akkor $f(x)$-nek lesz 0-tól különböző együtthatója, így nem lehet nullpolinom $\mathbb{Z}_n$ felett. ■

Prímszámok a segédtételben leírt módon való azonosíthatóságából közvetlenül adódik a prímtesztelő algoritmus alapgondolata: bemenetként adott n természetes számhoz keressünk megfelelő a egészet és ellenőrizzük, hogy teljesül-e a (22.17) kongruencia. Ez így természetesen nagyon egyszerű, de mivel mintegy n tagról kell eldönteni, hogy kongruens-e 0 (mod n), nem hatékony eljárás. A továbbiakban szeretnénk a számolásokat egy megfelelően választott relatíve kevés elemből álló véges struktúra felett végezni, azt remélve, hogy így le tudjuk csökkenteni $f(x)$ vizsgálandó együtthatóinak számát.

Az algebrában jól ismert tény, hogy tetszőleges R főideálgyűrűben $R/\langle a\rangle$ akkor és csak akkor lesz test, ha a felbonthatatlan elem R-ben. $\langle a\rangle$ itt az a elem által generált főideált jelenti. Tehát tekintsük az F_p véges test feletti $h(x)$ d-edfokú irreducibilis, azaz felbonthatatlan polinomot. Ekkor $F_p[x]/\langle h(x)\rangle$ egy p^d elemszámú testet alkot, amelynek elemei az F_p feletti legfeljebb $(d-1)$-edfokú polinomok. A továbbiakban alkalmazzuk a következő jelölést: ha $f(x)\equiv g(x)$ teljesül $\mathbb{Z}_n[x]/\langle h(x)\rangle$-ben, akkor azt írjuk, hogy

$$f(x)\equiv g(x) \pmod{h(x), n}\,.$$

Megjegyezzük, hogy $\mathbb{Z}_n$ akkor és csak akkor alkot testet, ha n prímszám, tehát a $\mathbb{Z}_n[x]/\langle h(x)\rangle$ gyűrű nem feltétlenül test, és legfeljebb $(deg(h(x))-1)$-edfokú polinomkifejezésekből áll, amelyek együtthatói $\mathbb{Z}_n$-beliek. Az AKS-algoritmusban $h(x)=x^r-1$, ahol r egy „megfelelően kicsi” pozitív egész. A továbbiakban természetesen a „megfelelően kicsi” kifejezés értelmét pontosítjuk. Tehát a prímek azonosítására használjuk (22.17) helyett a

$$(x+a)^n \equiv x^n + a \pmod{x^r-1, n}\,. \tag{22.18}$$

kongruenciát. A 22.40. lemmából közvetlenül következik, hogy ha n prímszám, akkor kielégíti a (22.18) összefüggést, minden a és r esetén. Látható továbbá, hogy ha r elég kicsi n-hez képest, akkor kevesebb együtthatót kell megvizsgálnunk. Sajnos cserébe azt az árat kell fizetnünk, hogy bizonyos a és r értékek esetén előfordulhat, hogy n összetett szám esetén is kielégíti a kongruenciát. Viszont bizonyítható, hogy megfelelően választott r esetén nem túl sok a értékre elvégezve a vizsgálatot, csak olyan összetett számok mehetnek át a teszten, amelyek prímhatványok. Az AKS-algoritmus annak köszönheti a gyorsaságát, hogy mind r értékére, mind a különböző a-k számokra olyan felső korlát adható, amely $\lceil \lg n\rceil$-nek polinomiális függvénye. Ezen tények, és az, hogy n-ről eldönteni prímhatvány mivoltát nem tart sokáig, eredményezi a $\lceil \lg n\rceil$-ben polinomiális futási időt. Ráadásul, ahogy azt a későbbiekben látni fogjuk, ennyi vizsgálat is elég ahhoz, hogy n prím voltát száz százalékos biztonsággal eldöntsük, azaz az algoritmus determinisztikus is, ami miatt kiemelkedik az eddig ismert prímtesztek közül.

Az AKS-teszt ismertetésekor feltesszük, hogy a bemenet egy egynél nagyobb n egész szám.

AKS-PRÍMTESZT(n)

```
 1  a ← 2
 2  b ← 2
 3  while n ≥ a^b
 4        do if n = a^b
 5              then return ÖSSZETETT
 6           b ← b + 1
 7           if n < a^b
 8              then b ← 2 ; a ← a + 1
 9  r ← a legkisebb olyan r amelyre o_r(n) > 4 lg^2 n
10  for a ← 2 to r
11     do if 1 < (a, n) < n
12           then return ÖSSZETETT
13  if n ≤ r
14    then return PRÍM
15  for a ← 1 to ⌊2 sqrt(φ(r)) lg n⌋
16     do if (x + a)^n ≢ x^n + a  (mod x^r − 1, n)
17           then return ÖSSZETETT
18  return PRÍM
```

22.5.1. Az algoritmus helyességének bizonyítása

22.41. tétel. *Az* AKS-PRÍMTESZT *algoritmus akkor és csak akkor tér vissza* PRÍM *értékkel, ha n prímszám.*

A további vizsgálatok áttekinthetőségének érdekében hat fő részre bontjuk az algoritmust: 1. programrész az 1–8. sor, 2. programrész a 9. sor, 3. programrész a 10–12. sor, 4. programrész a 13–14. sor, 5. programrész a 15–17. sor, 6. programrész a 18. sor.

Először azt látjuk be, hogy ha n prím, akkor az algoritmus tényleg a PRÍM értékkel tér vissza. Tegyük fel tehát, hogy n prímszám. Ekkor n nem lehet egyetlen számnak sem egynél nagyobb kitevős hatványa, tehát az 1. programrészben nem tér vissza az algoritmus az ÖSSZETETT értékkel. Ugyanígy a 3. programrészben sem, hiszen minden n-nél kisebb szám relatív prím n-hez. A 22.40. biztosítja számunkra, hogy az 5. programrészben sem lesz „kiugrás” a programból, tehát csak a 4. programrészben, vagy a 6. programrészben érhet véget az algoritmus, ahol mindkét esetben PRÍM a visszaadott érték.

Az állítás másik felének bizonyításához a továbbiakban feltesszük, hogy az algoritmus a PRÍM értékkel tér vissza. Abban az esetben, amikor a 4. programrészben lépünk ki a programból, készen is vagyunk a bizonyítással, mert ha n nem lenne prím, az a 3. programrészben kiderülne, hiszen találnánk nem triviális faktorát. Tehát a bizonyítás hátra lévő részében az általánosság megsértése nélkül feltehetjük, hogy az algoritmus a 6. programrészben ér véget PRÍM kimenettel.

Most koncentráljunk a 2. programrészre, ahol is a megfelelő r érték kiválasztása történik. Két segédtétel segítségével bizonyítjuk, hogy létezik ilyen r.

22.42. lemma. *Jelölje $LCM(m)$ az első m pozitív egész legkisebb közös többszörösét. Ekkor $m \geq 7$ esetén*

$$LCM(m) \geq 2^m \ .$$

Bizonyítás. Az állítás a prímszámok számára vonatkozó alsó és felső becslésekből rögtön következik, ugyanis

$$LCM(M) \geq \prod_{2 \leq p \leq m} p \ ,$$

így a bizonyítás kész. ■

22.43. lemma. *Ha az n egész számnak minden prímosztója nagyobb, mint $\lceil 16 \lg^5 n \rceil$, akkor létezik olyan $r \leq \lceil 16 \lg^5 n \rceil$, amelyre $o_r(n) > 4 \lg^2 n$.*

Bizonyítás. Legyen

$$T = \prod_{i=1}^{\lfloor 4 \lg^2 n \rfloor} (n^i - 1) < n^{16 \lg^4 n} \leq 2^{16 \lg^5 n} \ .$$

Ekkor

$$T < n^{16 \lg^4 n} = 2^{16 \lg^5 n}$$

fennáll $n = 2^{\lg n}$ miatt. Ha az n minden prímosztója nagyobb, mint $\lceil 16 \lg^5 n \rceil$, és minden $r \leq \lceil 16 \lg^5 n \rceil$ $o_r(n) \leq 4 \lg^2 n$ lenne, akkor $r \mid n^{o_r(n)} - 1$ miatt $r \mid T$ teljesülne. De ekkor

$$LCM\left(16 \lg^5 n\right) \mid T$$

igaz, és az előző lemma miatt

$$2^{\lceil 16 \lg^5 n \rceil} \leq T < 2^{16 \lg^5 n}$$

teljesülne, ami nyilvánvalóan ellentmondás. ■

A bizonyítás hátralévő részében rögzítjük három változónak az értékét. Az egyik a megtalált r érték, továbbá legyen $k = \lfloor 2 \sqrt{\varphi(r)} \lg n \rfloor$. A harmadik rögzített változó pedig legyen n-nek egy tetszőleges p prímosztója. Tudjuk, hogy $p > r$ és $(n, r) = 1$, különben a program megállna a 3. programrészben vagy a 4. programrészben azaz r és n is inkongruensek (mod r).

Feltételezésünk szerint az algoritmus nem lépett ki az 5. programrészben összetett értékkel, tehát tehát k darab kongruencia ellenőrzése után azt találjuk, hogy fennállnak a következő kongruenciák:

$$(x + a)^n \equiv x^n + a \pmod{x^r - 1, n} \ .$$

minden $1 \leq a \leq k$ értékre. Mivel p prímfaktora n-nek, azonnal következik, hogy

$$(x + a)^n \equiv x^n + a \pmod{x^r - 1, p} \ . \tag{22.19}$$

minden $1 \leq a \leq k$-ra. Ebből pedig 22.40. lemma implikálja, hogy minden $1 \leq a \leq k$ esetén igaz a

$$(x+a)^p \equiv x^p + a \pmod{x^r - 1, p}\,. \tag{22.20}$$

kongruencia. Úgy is megfogalmazhatnánk ezt a jelenséget, hogy n úgy „viselkedik" mint a p prímszám. Ennek a tulajdonságnak nevet is adunk, mivel a további vizsgálatokban fontos szerepet játszik.

22.44. definíció. *A fenti jelöléseket megtartva tetszőleges $f(x) \in \mathbb{Z}[x]$ polinom és $m \in \mathbb{N}$ szám esetén azt mondjuk, hogy m **önelemző** az $f(x)$ polinomra nézve, ha fennáll*

$$(f(x))^m \equiv f(x^m) \pmod{x^r - 1, p}\,.$$

(22.19)-ből és (22.20)-ból kiolvasható, hogy mind n, mind p önelemző szám az $x + a$ polinomra nézve az $1 \leq a \leq k$ esetekben. A következő állítás azt mutatja meg, hogy egy polinomra nézve önelemző számok halmaza zárt a szorzás műveletre.

22.45. lemma. *Ha m, m' önelemző számok az $f(x)$ polinomra nézve, akkor $m \cdot m'$ is az.*

Bizonyítás. Mivel m önelemző $f(x)$-re nézve, ezért kapjuk, hogy

$$(f(x))^{m\cdot m'} \equiv (f(x^m))^{m'} \pmod{x^r - 1, p}\,.$$

Mivel m' is önelemző $f(x)$-re nézve, ezért x-et x^m-nel helyettesítve adódik, hogy

$$(f(x^m))^{m'} \equiv f\left(x^{m\cdot m'}\right) \pmod{x^{r\cdot m} - 1, p}\,,$$

de $x^r - 1$ osztója a $x^{r\cdot m} - 1$ polinomnak, így

$$(f(x^m))^{m'} \equiv f\left(x^{m\cdot m'}\right) \pmod{x^r - 1, p}\,,$$

tehát az

$$(f(x))^{m\cdot m'} \equiv f\left(x^{m\cdot m'}\right) \pmod{x^r - 1, p}$$

összefüggést kaptuk, ami mutatja, hogy $m \cdot m'$ is önelemző $f(x)$-re nézve. ■

Egy m számhoz tartozó polinomokra is hasonló állítás bizonyítható, amelyekre nézve m önelemző.

22.46. lemma. *Ha m önelemző szám az $f(x)$ és $g(x)$ polinomra nézve, akkor m önelemző az $f(x) \cdot g(x)$ polinomra nézve.*

Bizonyítás. Az állítás tulajdonképpen közvetlenül adódik a következő összefüggésből:

$$(f(x)g(x))^m = (f(x))^m \cdot (g(x))^m \equiv f(x^m) \cdot g(x^m) \pmod{x^r - 1, p}\,,$$

amivel a bizonyítást be is fejeztük. ■

Az eddigi jelölések megtartásával konstruáljunk meg két halmazt, amelyek fontosak lesznek a bizonyítás további részében. Legyen $I = \left\{n^i \cdot p^j \mid i, j \geq 0\right\}$, illetve $P =$

$\left\{\prod_{a=1}^{k}(x+a)^{e_a} \mid e_a \geq 0\right\}$. Az előző két segédtétel alapján nyilvánvaló, hogy I minden eleme önelemző P minden elemére nézve. Az első csoport, a továbbiakban nevezzük G-nek, alaphalmazában tartalmazza I összes elemének osztási maradékát $\pmod r$. Mivel $(n,r) = (p,r) = 1$ és természetesen multiplikatív struktúráról van szó, $G \leq \mathbb{Z}^*$, azaz G részcsoportja $\mathbb{Z}^*$-nak. Legyen $|G| = t$. Megállapíthatjuk, hogy $t > 4\lg^2 n$, mert G n és p által generált $\pmod r$, továbbá tudjuk, hogy $o_r(n) > 4\lg^2 n$.

A második csoport definiálásához szükségünk van néhány olyan fogalomra, amelyek a véges testek elméletével kapcsolatosak. Itt használunk néhány olyan állítást, melyek részletes bizonyítását nem közöljük.

22.47. definíció. *Legyen F p karakterisztikájú véges test, r p-vel nem osztható pozitív egész, és ξ primitív r-edik egységgyök F felett. Ekkor a*

$$Q_r(x) = \prod_{lnko(s,r)=1}^{r} (x - \xi^s)$$

polinom az ***r-edik körosztási polinom K felett***

A $Q_r(x)$ körosztási polinom az $x^r - 1$ polinomot $o_r(p)$ fokú irreducibilis faktorokra bontja. Legyen $h(x)$ egy ilyen irreducibilis faktor. A második csoport alaphalmaza tartalmazza az összes P-beli polinom nemnulla osztási maradékát $\pmod{h(x), p}$. Jelölje ezt a csoportot Γ. Megállapíthatjuk, hogy Γ az $F = F_p[x]/\langle h(x)\rangle$-beli $x+1, x+2, \ldots, x+k$ polinomok által generált, és részcsoportja F multiplikatív csoportjának.

A következő két lemmával Γ elemszámát próbáljuk megbecsülni.

22.48. lemma.

$$|\Gamma| \geq \binom{t+k-2}{t-1}.$$

Bizonyítás. Elsőként megjegyezzük, hogy mivel $h(x)$ a $Q_r(x)$ körosztási polinom faktora, ezért x primitív r-edik egységgyök F-ben. Valóban, $x^r - 1 \equiv 0 \pmod{h(x)}$ miatt x r-edik egységgyök $\pmod{h(x)}$, ezért x rendje valamely l, amelyre $l \mid r$. Ha

$$x^l - 1 \equiv 0 \pmod{h(x)}, \text{ és } h(x) = \prod_{j=1}^{d}(x - \xi_j),$$

akkor $h(x) \mid x^l - 1$ miatt $\xi_j^l = 1$, másrészt $h(x) \mid Q_r(x)$ miatt $\xi_j = \zeta^s$ alkalmas s, $(s,r) = 1$ értékre, ahol ζ primitív r-edik egységgyök. Triviálisan igaz, hogy $1 = \xi_j^l = \zeta^{sl}$ nem teljesülhet, ha $l < r$, azaz x $\pmod{h(x)}$ rendje r.

Második lépésben belátjuk, hogy különböző P-beli polinomok képei különböző elemei lesznek Γ-nak is. Tegyük fel ezért indirekt módon, hogy $f(x)$ és $g(x)$ különböző polinomok P-ben, de $f(x) = g(x)$ az F testben. Ekkor tetszőleges $m \in I$-re

$$(f(x))^m = (g(x))^m$$

F-ben. Mivel m önelemző mind $f(x)$, mind $g(x)$-re nézve, továbbá $h(x)$ osztója $(x^r - 1)$-nek,

$$(f(x^m)) = (g(x^m))$$

F-ben. Ez azt jelenti, hogy x^m gyöke a $Q(y) = f(y) - g(y)$ polinomnak minden $m \in G$ esetén. Mivel $(m, r) = 1$ ($G \leq \mathbb{Z}^*$), minden ilyen x^m r-edik primitív egységgyök lesz. Tehát $Q(y)$-nak $|G| = t$ különböző gyöke lesz F-ben. Viszont f és g választása miatt $Q(y)$ fokszáma kisebb, mint t, ami ellentmondás, vagyis kaptuk, hogy $f(x) \neq g(x)$ F-ben.

Harmadszor vegyük észre, hogy az $x+1, x+2, \ldots, x+k$ polinomok F-ben mind különbözőek, hiszen

$$k = \lfloor 2\sqrt{\varphi(r)} \lg n \rfloor < \lfloor 2\sqrt{r} \lg n \rfloor < r < p \ .$$

Az természetesen előfordulhat, hogy $h(x) = x + a$ valamely $a \leq k$-ra. Ekkor $x + a = 0$ F-ben, s így nem kerül be Γ-ba, ezért legalább $k-1$ különböző elsőfokú polinom lesz Γ-ban. Ez azt jelenti, hogy legalább

$$\binom{t+k-2}{t-1}$$

különböző legfeljebb $(t-1)$-edfokú polinom lesz Γ-ban. ■

Abban az esetben, ha n nem hatványa p-nek, Γ-ra felső korlát adható.

22.49. lemma. *Ha n nem p-hatvány, akkor*

$$|\Gamma| < \frac{1}{2} n^{2\sqrt{t}} \ .$$

Bizonyítás. Tekintsük I következő részhalmazát:

$$J = \left\{ n^i \cdot p^j \mid 0 \leq i, j \leq \lfloor \sqrt{t} \rfloor \right\} \ .$$

Ha n nem hatványa p-nek, akkor a J halmaz $(\lfloor \sqrt{t} \rfloor + 1)^2 > t$ különböző elemből áll. Mivel $|G| = t$, ezért J-ben van legalább két kongruens $\pmod r$ elem. Legyen m_1, m_2 két ilyen szám. Az általánosság megsértése nélkül feltehetjük, hogy $m_1 > m_2$. Ekkor

$$x^{m_1} \equiv x^{m_2} \pmod{x^r - 1} \ .$$

Legyen $f(x) \in P$. Elvégezhetjük a következő levezetést az F testben:

$$\begin{aligned} (f(x))^{m_1} &\equiv f(x^{m_1}) \pmod{x^r - 1, p} \ , \\ (f(x))^{m_1} &\equiv f(x^{m_2}) \pmod{x^r - 1, p} \ , \\ (f(x))^{m_1} &\equiv (f(x))^{m_2} \pmod{x^r - 1, p} \ . \end{aligned}$$

Tehát az $f(x) \in \Gamma$ gyöke a $Q'(y) = y^{m_1} - y^{m_2}$ polinomnak F-ben. Mivel $f(x)$ tetszőleges eleme Γ-nak, ezért $Q'(y)$-nak legalább $|\Gamma|$ különböző gyöke van F-ben. Mivel p osztja n-et, de $p \neq n$,

$$\deg Q'(y) = m_1 \leq (np)^{\lfloor \sqrt{t} \rfloor} < \frac{1}{2} n^{2\sqrt{t}} \ .$$

Tehát megkaptuk a keresett

$$|\Gamma| < \frac{1}{2} n^{2\sqrt{t}}$$

összefüggést. ■

Tehát Γ elemszámára alsó és felső becslést is tudtunk adni. Ennek birtokában már könnyen beláthatjuk a tételt.

22.50. lemma. *Ha az* AKS-PRÍMTESZT *algoritmus a* PRÍM *értékkel tér vissza, akkor n prímszám.*

Bizonyítás. Tegyük fel, hogy a program a PRÍM értékkel tért vissza. Ekkor a 22.48. lemma értelmében $|G| = t$ és $k = \lfloor 2\sqrt{\varphi(r)} \lg n \rfloor$ esetén felírhatjuk a következő egyenlőtlenségeket, figyelembe véve, hogy $t > 2\sqrt{t} \lg n\, k \geq \lfloor 2\sqrt{t} \lg n \rfloor$ és $2\sqrt{t} \lg n \geq 3$:

$$\begin{aligned}
|\Gamma| &\geq \binom{t+k-2}{t-1}, \\
|\Gamma| &\geq \binom{k-1+\lfloor 2\sqrt{t}\lg n\rfloor}{\lfloor 2\sqrt{t}\lg n\rfloor}, \\
|\Gamma| &\geq \binom{2\lfloor 2\sqrt{t}\lg n\rfloor - 1}{\lfloor 2\sqrt{t}\lg n\rfloor}, \\
|\Gamma| &\geq 2^{\lfloor 2\sqrt{t}\lg n\rfloor}, \\
|\Gamma| &\geq \frac{1}{2} n^{2\sqrt{t}}.
\end{aligned}$$

Viszont a 22.49. lemmában láttuk, hogy ha n nem hatványa p-nek, akkor

$$|\Gamma| < \frac{1}{2} n^{2\sqrt{t}}.$$

Kaptuk, hogy $n = p^s$ valamely s pozitív egészre. Ha $k > 1$ lenne, akkor az algoritmus az 1. programrészben visszatért volna az ÖSSZETETT értékkel. Tehát $n = p$ prímszám, amivel bizonyítottuk a lemmát, és egyúttal a tételt is. ■

22.5.2. A futási idő elemzése

Ebben az alfejezetben az AKS-algoritmus időigényére fogunk felső korlátot adni. Megjegyezzük, hogy a bizonyításban szereplőnél sokkal alacsonyabb korlát is adható, de annak a bizonyítása bonyolultabb lenne, ezért közlésétől eltekintünk. Tesszük ezt annál is inkább, mert célunk a $\lg n$ függvényében polinomiális futási idő belátása. A következő alfejezetben olvashatunk az algoritmus gyorsításának elméleti és gyakorlati lehetőségeiről.

22.51. tétel. *Az* AKS-PRÍMTESZT *algoritmus aszimptotikus időigénye* $O^{\sim}(\lg^{10.5} n)$.

Bizonyítás. Vizsgálatainkhoz felhasználjuk, hogy két m bites szám szorzásának, osztásának és összeadásának időigénye $O^{\sim}(m)$, továbbá két d-edfokú legfeljebb m bites együtthatókkal rendelkező polinom esetén ugyanezen műveletek elvégezhetők $O^{\sim}(d \cdot m)$ lépésben. Az 1. programrész időigénye is jól ismert, így ezt itt nem részletezzük, csak közöljük, hogy $O^{\sim}(\lg^3 n)$.

A 2. programrészben keresünk egy r számot, amelyre $o_r(n) > 4\lg^2 n$. Ez történhet úgy is, hogy veszünk egymás utáni r értékeket és ellenőrizzük az $n^s \not\equiv 1 \pmod r$ kongruencia teljesülését minden $s \leq 4\lg^2 n$-re. Ez azt jelenti egy bizonyos r értéknél, hogy legfeljebb $O(\lg^2 n)$ szorzást kell végeznünk $\pmod r$, vagyis az időigény $O^{\sim}(\lg^2 n \lg r)$. A 22.43. lemmában láttuk, hogy a legrosszabb esetben is csak $O(\lg^5 n)$ nagyságrendű különböző r értéket kell kipróbálnunk, tehát a 2. programrész időigénye $O^{\sim}(\lg^7 n)$.

A 3. programrészben r számpár legnagyobb közös osztóját keressük. Mint azt ezen könyv első kötetében is olvashatjuk, ennek időigénye egyenként $O(\lg n)$. r darab pár esetén tehát a 3. programrész időigénye $O(r \lg n) = O(\lg^6 n)$.

Mivel a 4. programrész időigénye elhanyagolható, rögtön az 5. programrész vizsgálatára térünk. Itt $\lfloor 2\sqrt{\varphi(r)} \lg n \rfloor$ kongruencia teljesülését ellenőrzi az eljárás. Minden kongruenciánál $O(\lg n)$ darab r-edfokú polinomszorzásra van szükség, ahol az együtthatók $O(\lg n)$ nagyságrendűek. Így minden kongruencia ellenőrzésének időigénye $O^{\sim}(r \lg^2 n)$, így az 5. programrész időigényére

$$O^{\sim}(r\sqrt{\varphi(r)} \lg^3 n) = O^{\sim}(r^{\frac{3}{2}} \lg^3 n) = O^{\sim}(\lg^{10.5} n)$$

adódik, és mivel ennek a lépésnek az időigénye dominál, az egész algoritmusét meghatározza. ■

22.5.3. Az algoritmus tökéletesítése

Amint azt említettük az előző alfejezetben, az eljárás gyorsítására több lehetőség is kínálkozik. Ezek lényege, hogy az r érték becslését finomítsák. Látható, hogy ideális esetben r értéke $O(\lg^2 n)$ nagyságrendű, így az egész algoritmus időigénye $O(\lg^6 n)$ lehet. A következő két sejtés azt sugallja, hogy jó esély van a 22.43. lemmában leírtaknál kisebb r-et találni.

Az egyik az ***Artin-sejtés*** (lásd 22.19), a másik a

22.52. sejtés. *Azon $q \leq m$ prímek száma, melyekre $2q + 1$ is prím, aszimptotikusan*

$$\frac{2C_2 m}{\ln^2 m},$$

ahol C_2 az úgynevezett ikerprím konstans, amelynek közelítő értéke 0.66.

Megjegyezzük, hogy utóbbi a ***Sophie-Germain prímek*** sűrűségére vonatkozó sejtésként ismeretes a szakirodalmakban.

Az Artin-sejtés igaz volta közvetlenül maga után vonná, hogy $O(\lg^2 n)$ nagyságrendű m-re található olyan $r = O(\lg^2 n)$, amely kielégíti a kívánt feltételeket. Sajnos jelenleg az sem bizonyított, hogy minden nem négyzetszám n esetén végtelen sok q prímszámra teljesül az $o_q(n) = q - 1$ összefüggés.

Ami a Sophie-Germain prímeket illeti, ha igaz a rájuk vonatkozó sejtés, akkor biztosan

létezik legalább $\lg^2 n$ ilyen prím $\lg^2 n$ és $c\lg^2 n(\lg\lg n)^2$ között, megfelelő c konstans szorzóval. Minden ilyen q prímszám esetén vagy $o_q(n) \le 2$ áll fenn, vagy $o_q \ge (q-1)/2$. Bármely olyan q, amire $o_q(n) \le 2$ igaz, osztója $(n-1)(n^2-1)$ és így számuk $O(\lg n)$ nagyságrendű. Ez a tény implikálja olyan $r = O^{\sim}(\lg^2 n)$ létezését, ami kielégíti a $o_r(n) \ge 4\lg^2 n$ feltételt. Egy ilyen nagyságrendű r érték az AKS-algoritmusnak $O^{\sim}(\lg^6 n)$ futási időt eredményez.

Az időkorlát nagyságrendjének csökkentése az AKS-algoritmus szerzőinek konkrét eredménye is született. Jelölje $P(m)$ az m legnagyobb prímosztóját. Goldfeld megmutatta, hogy pozitív valószínűséggel fordulnak elő olyan q prímek, melyekre

$$P(q-1) > q^{\frac{1}{2}+c} ,$$

ahol $c \approx 1/12$. Ezt az eredményt Fouvry tökéletesítette, bizonyítva a következő állítást.

22.53. lemma. *Létezik olyan $c > 0$ konstans és n_0 pozitív egész, hogy minden $x \ge n_0$ esetén:*

$$\left|\left\{q \mid q \text{ prímszám, } q \le x \text{ és } P(q-1) > q^{\frac{2}{3}}\right\}\right| \ge c\frac{x}{\ln x} .$$

Jelenleg a lemmának már $q^{0.6683}$-as értékre is létezik bizonyítása. Ezen eredmények felhasználásával tudták a szerzők az AKS-algoritmus várható futási idejének felső korlátját javítani.

22.54. tétel. *Az* AKS-prímteszt *algoritmus aszimptotikus időigénye $O^{\sim}(\lg^{7.5} n)$.*

Bizonyítás. A fentiek értelmében az olyan q prímek, melyekre $P(q-1) > q^{2/3}$, nagy sűrűsége azt eredményezi, hogy a 2. programrészben az eljárás talál olyan $r = O(\lg^3 n)$ értéket, amire fennáll, hogy $o_r(n) > 4\lg^2 n$. Emiatt lehet az algoritmusnak $O^{\sim}(\lg^{7.5} n)$ nagyságrendű futási ideje. ■

Az AKS-algoritmus futásának időkorlátját leszorítani nem csak az r érték megválasztásának finomításával lehet. Tekintsük az 5. programrészt, ahol egy olyan ciklus megy 1-től $\lfloor 2\sqrt{\varphi(r)}\lg n\rfloor$-ig, amely tulajdonképpen azt ellenőrzi, hogy a Γ csoport alaphalmaza „elég” nagy-e. Ezen iterációs lépések számát csökkenthetjük oly módon, hogy találunk $(x+a)$ alakú polinomoknak egy olyan kisebb számosságú halmazát, ami generálja a kívánt elemszámú csoportot.

Biztató lehet a jövőre nézve a következő sejtés, amely ha igaz, akkor az AKS-prímteszt úgy módosítható, hogy az új algoritmus sokkal kisebb időigényt sejtet. Megjegyezzük, hogy az állítás helyességét $r \le 100$ és $n \le 10^{10}$ esetre már ellenőrizték.

22.55. sejtés. *Legyen r olyan prím, amely nem osztója n-nek. Ha fennáll az*

$$(x-1)^n \equiv x^n - 1 \pmod{x^r - 1, n} . \tag{22.21}$$

kongruencia, akkor n prím, vagy $n^2 \equiv 1 \pmod r$.

Ha a sejtés igaz, akkor az a teendőnk, hogy olyan r értéket keressünk, amely nem osztója n^2-1-nek. Ilyen r biztosan található, a $[2, 4\lg n]$ intervallumban. Ez abból következik, hogy az x-nél kisebb prímek szorzata legalább e^x. Ezután már csak (22.21) teljesülését kell ellenőrizni, amelynek az időigénye $O^{\sim}(\lg^2 n)$, vagyis az AKS-prímteszt futási ideje $O^{\sim}(\lg^3 n)$.

22.5.4. Az algoritmus megvalósíthatósága

Az AKS-algoritmus gyakorlati megvalósításával kapcsolatban fel kell hívnunk a figyelmet egy fontos dologra. Ez nevezetesen az, hogy bonyolultságelméleti szempontból a futásidő nagyon kedvezően alakul, viszont a tárigény olyan mértékűnek mutatkozik, ami viszonylag kicsi számoknál is óriási operatív memóriát feltételez. A gyorsaság érdekében az 5. programrészben az egy lépésben vizsgált polinom együtthatóit mind az operatív tárban célszerű tartani.

Egy $(x+a)^n - x^n - a$ alakú polinom osztási maradéka a $(\mathrm{mod}\ x^r - 1, n)$ modulus szerint egy legfeljebb $(r-2)$-edfokú polinom lesz, amelynek együtthatói legfeljebb az $n-1$ értéket vehetik fel. Tehát legrosszabb esetben $(r-1)$ darab együtthatót kell ellenőriznünk, amelyek tárigénye akár $(r-1)\cdot\lceil\lg(n-1)\rceil$ bit is lehet. Legyen n a legnagyobb 1000 decimális számjegyű pozitív egész. Tegyük fel, hogy r értéke eléri a bizonyításban szereplő $\lceil 16\lg^5 n\rceil$ korlátot. Ekkor egyszerű számolással belátható, hogy a tárigény túllépheti a $0.25\cdot 10^{16}$ gigabájtot. Manapság egy 1000 számjegyű egész nem számít nagynak prímtesztelés szempontjából, de ekkora operatív tár nem áll rendelkezésre.

Érdemes elvégezni ugyanezt a számolást úgy, hogy feltesszük, hogy a 22.55. sejtés igaz. Ekkor találunk olyan r-et, amelyre $r \le 4\lg n$, ami azt jelenti, hogy a tárigény 5.27 gigabájt alá csökken.

A fenti számítások azt támasztják alá, hogy nem kell teljesen elfelejteni gyakorlati alkalmazások szempontjából az AKS-PRÍMTESZT eljárást, viszont ahhoz, hogy valóban egy determinisztikus, nagy számok prímtesztelésére is alkalmas számítógépes program születhessen, még szükség van új matematikai eredményekre, elsődlegesen a 22.55. sejtés bizonyítására gondolunk, a hardverek fejlődésére, valamint ügyes programozói megoldásokra.

Gyakorlatok

22.5-1. Írjunk programot, amely adott n határig ellenőrzi a 22.55. sejtés igaz voltát.

22.5-2. Tekintsünk egy $\mathbb{F}_p$ véges prímtestet, és egy $g \in \mathbb{F}_p[x]$ irreducibilis polinomot. Lássuk be, hogy $T = \mathbb{F}_p[x]/(g)$ test, ha (g) a g által generált ideált jelöli.

22.5-3. Konkrét p-re írjuk fel az előző gyakorlatban látott T test elemeit. Adjuk meg a műveleti táblákat, és keressük meg a primitív elemeket T multiplikatív csoportjában.

22.6. Elliptikus görbék

Legyen K az $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Q}$, $\mathbb{F}_q$ ($q = p^r$ prímhatvány) testek valamelyike, ahol $p \neq 2, 3$. Legyen továbbá

$$x^3 + ax + b \in K[x]$$

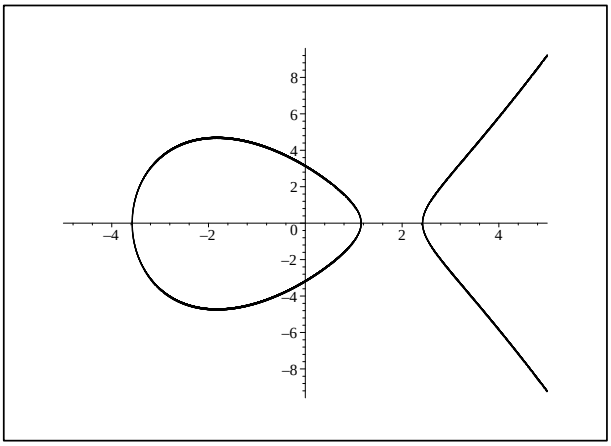
olyan harmadfokú polinom, amelynek gyökei különbözőek.

22.56. definíció. ***Elliptikus görbén*** *azoknak az $(x, y) \in K \times K$ pontoknak az $E(K)$ halmazát értjük, amelyekre*

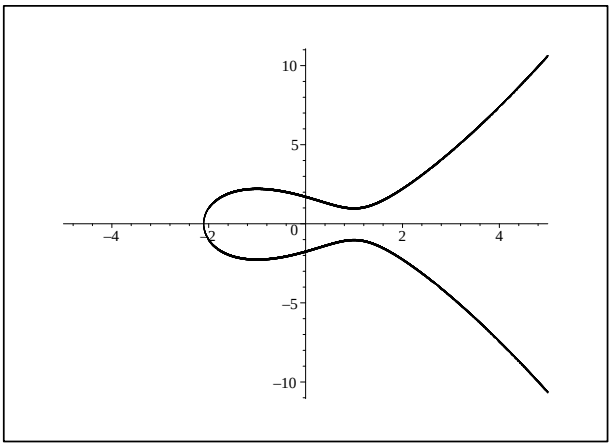
$$y^2 = x^3 + ax + b$$

teljesül, kiegészítve ezt egy úgynevezett „végtelen távoli” O elemmel.

Legyen $K = \mathbb{R}$. Ekkor $E(K)$ vagy három, vagy egy helyen metszi az x-tengelyt. Az előbbi esetre a 22.1. ábrán, utóbbira a 22.2. ábrán láthatunk példát.



22.1. ábra. Az $y^2 = x^3 - 10x + 10$ egyenlettel adott elliptikus görbe.



22.2. ábra. Az $y^2 = x^3 - 3x + 3$ egyenlettel adott elliptikus görbe.

$E(\mathbb{R})$ pontjai közt alkalmas műveletet bevezetve csoportot kapunk. Az erre vonatkozó tételt nem bizonyítjuk.

Legyen P, Q két különböző pont $E(\mathbb{R})$-ben, amelyekre P és Q nem egymás tükörképei az x-tengelyre vonatkozóan. A P, Q pontokon átmenő egyenes $E(\mathbb{R})$-et egy harmadik pontban is metszi, jelöljük ezt $P * Q$-val. $P * Q$-nak az x-tengelyre való tükörképe $P + Q$. Ha $P = Q$, akkor a P, Q pontokon átmenő egyenest a P-n átmenő érintőjével helyettesítve a további metszéspont $P * P$ és $P + P$. A $P \in E(\mathbb{R})$ pont tükörképe $-P$. A P-n és $(Q =) - P$-n átmenő egyenes merőleges az x-tengelyre. Azt mondjuk, hogy a végtelen távoli pontban, O-ban metszi a görbét, $P * (-P) = O$, és $P + (-P) = O$. Továbbá, ha $P = (x, y)$, akkor $-P = (x, -y)$.

Legyen $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $y = \alpha x + \beta$ a P, Q ponton átmenő egyenes (nem merőleges az x-tengelyre). Ekkor

$$\alpha = \frac{y_2 - y_1}{x_2 - x_1},\ \beta = y_1 - \alpha x_1\ .$$

Legyen $P * Q = (x_3, y_3)$ $(= (x_3, \alpha x_3 + \beta))$. Az

$$(\alpha x + \beta)^2 = x^3 + ax + b$$

harmadfokú egyenletnek három gyöke van: x_1, x_2, x_3. Az egyenletet

$$x^3 - \alpha^2 x^2 + (a - 2\alpha\beta)x + (b - \beta^2) = 0$$

alakban írva, s figyelembe véve, hogy a bal oldali polinom ekvivalens az $(x - x_1)(x - x_2)(x - x_3)$ polinommal,

$$x_1 + x_2 + x_3 = \alpha^2,\ x_3 = \alpha^2 - x_1 - x_2$$

adódik. Mivel $P + Q = (x_3, -y_3)$, ezért P és Q összege könnyen kiszámítható.

Legyen most $P = Q = (x_1, y_1)$, $y_1 \neq 0$. Az $E(\mathbb{R})$ görbe P pontbeli $y = \alpha x + \beta$ érintőjét az $\alpha = dy/dx$ differenciálhányados kiszámításával határozhatjuk meg. Implicit differenciálással:

$$2ydy = (3x^2 + a)dx,\ \frac{dy}{dx} = \frac{3x^2 + a}{2y}\ |_{x_1, y_1} = \frac{3x_1^2 + a}{2y_1} = \alpha\ .$$

A számolás többi része változatlan.

Bevezetjük a következő jelölést $n \in \mathbb{N}$-re:

$$nP = \underbrace{P + P + \ldots P}_{n\text{-szer}}\ ,$$

és

$$-nP = \underbrace{(-P) + (-P) + \cdots + (-P)}_{n\text{-szer}}\ .$$

Hasonló módon definiálhatjuk az $E(K)$-beli műveletet a $K = \mathbb{C}$, $K = \mathbb{Q}$, $K = \mathbb{F}_q$ testekre is. A $P \in E(K)$ pontot n-edrendűnek nevezzük, ha n az a legkisebb pozitív egész szám, amelyre $nP = O$.

22.57. tétel (Hasse). *Legyen $p \neq 2, 3$ prímszám, $\mathbb{F}_p$ a p-elemű test,*

$$E(\mathbb{F}_p):\ \ y^2 = x^3 + ax + b,\ \ (a, b \in \mathbb{F}_p)\ .$$

Tegyük fel továbbá, hogy $4a^3 + 27b^2 \neq 0$. Ekkor az $E(\mathbb{F}_p)$ pontjainak $\sharp(E(\mathbb{F}_p))$ számára

$$-2\sqrt{p} \leq \sharp(E(\mathbb{F}_p)) - (p - 1) \leq 2\sqrt{p}\ .$$

22.6.1. Az elliptikus görbék alkalmazásai

Az elliptikus görbék vizsgálata az algebrai geometria és a számelmélet fontos területe. H.W. Lenstra az előző tételre alapozva egy érdekes, sok esetben igen hatékony algoritmust dolgozott ki egész számok prímtényezős felbontására (prímfelbontásra).

Mielőtt ezt ismertetnénk, ismerjük meg a ***Pollard-féle*** $p-1$ ***algoritmussal.*** Ennek alapelve a következő.

Tegyük fel, hogy az n egésznek van olyan p prímosztója, amelyre $p-1$ minden prímosztója kicsi, $p-1 \mid k$, ahol

$$k = 2^{a_2} \cdot 3^{a_3} \cdot \dots \cdot t^{a_t}$$

az első néhány prímszám, és $a_2, a_3, \dots$ pozitív egészek. Világos, hogy $p \mid a^{p-1}-1$ és $p \mid a^k - 1$, következésképpen $p \mid (a^k - 1, n)$. $a^k \pmod n$ a ***gyors-hatványozás*** módszerével, $(a^k \pmod n, n)$ pedig az ***euklidészi algoritmussal*** hatékonyan kiszámítható. A program bemeneti értéke $n \geq 2$ összetett egész szám. Az LKKT$(a_1, \dots, a_r)$ függvény az $a_1 \dots, a_r$ elemek legkisebb közös többszörösét adó függvényt jelenti, míg (x, y) x és y legnagyobb közös osztóját. Ha n összetett, akkor az algoritmus n egy nem triviális faktorával tér vissza.

POLLARD-PMÍNUSZEGY(n)

```
 1  k ← LKKT(1, 2, ..., K)
 2  a ← RANDOM(2, n − 1)
 3  D ← (a, n)
 4  if D > 1
 5     then return D
 6  D ← (a^k − 1, n)
 7  if 1 < D < n
 8     then return D
 9  if D = 1
10     then K ← K + 1
11          goto 1
12     else goto 2
```

Mielőtt Lenstra algoritmusára rátérnénk, bizonyos előkészületeket teszünk. Legyen m egész, $x_1 = a_1/n_2$, $x_2 = a_2/n_2$ redukált alakban felírt racionális számok, amelyekre $(n_1, m) = (n_2, m) = 1$. Az $x_1 - x_2$ redukált alakja legyen $x_1 - x_2 = a/n$. Ekkor nyilvánvalóan $(n, m) = 1$, mivel $n \mid [n_1, n_2]$. Az x_1 és x_2 racionális számokat kongruenseknek mondjuk $\pmod m$, jelben $x_1 \equiv x_2 \pmod m$, ha $m \mid a$.

Könnyen beláthatjuk, hogy e kongruencia reláció ekvivalencia reláció az

$$\left\{ \frac{a}{n} \mid a \in \mathbb{Z},\ n \in \mathbb{N},\ (a,n) = 1,\ (n,m) = 1 \right\}$$

gyűrűben. Továbbá a/n-hez található olyan $b \in \{0, 1, \dots, m-1\}$ egész, amellyel $a/n \equiv b \pmod m$. Ez nyilvánvaló, mivel az $a \equiv xn \pmod m$ kongruencia $(n, m) = 1$ miatt megoldható.

Legyen $E : y^2 = x^3 + ax + b$ adott elliptikus görbe, ahol $x, y \in \mathbb{Z}$. Tegyük fel, hogy $P = (x, y)$ racionális pont a görbén. Ekkor $P \pmod n$ alatt az $(x \pmod n, y \pmod n)$ párt

értjük. Az algoritmus során hatékonyan ki kell számítanunk a $kP \pmod n$ értékeket. Erre hatékony eljárás lehet az úgynevezett ***duplázás módszere*** $P_{j+1} = 2P_j$ $P_0 = P$ $(P_j = 2^j P)$, $\ldots$, ahol $j = 0, 1, 2, \ldots$.

$E(\mathbb{Q})$ pontjait és ezek összegét mindig $\pmod n$ tekintjük. Akkor helyes a számolás, ha a számítások során előforduló valamennyi nevező relatív prím n-hez.

22.58. tétel. *Legyen*

$$E: \quad y^2 = x^3 + ax + b, \quad (a, b \in \mathbb{Z})$$

elliptikus görbe, $(4a^3 + 27b^2, n) = 1$. *Legyen* $P_1, P_2 \in E(\mathbb{Q})$ $P_1 \neq -P_2$, *és a* P_1, P_2 *pontok koordinátáinak nevezői az n-hez relatív prímek. Legyen továbbá*

$$E \pmod p: \quad y^2 = x^3 + \overline{a}x + \overline{b}\,,$$

ahol $\overline{a}, \overline{b} \in \mathbb{F}_p$, $a \equiv \overline{a} \pmod p$, $\overline{b} \equiv b \pmod p$.

Ekkor a $P_1 + P_2 \in E(\mathbb{Q})$ *pont (racionális) koordinátáiban a nevezők relatív prímek n-hez, kivéve, ha van olyan* $p \mid n$ *prímszám, amelyre*

$$P_1 \pmod p + P_2 \pmod p = O\,,$$

az $E \pmod p$ *görbén.*

Bizonyítás. Tegyük fel, hogy a $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_1 + P_2 \in E(\mathbb{Q})$ pontok koordinátáiban a nevezők n-hez relatív prímek. Legyen $p \mid n$. Meg kell mutatni, hogy

$$P_1 \pmod p + P_2 \pmod p \neq O \pmod p\,.$$

Ha $x_1 \equiv x_2 \pmod p$, akkor az $E \pmod p$ görbén vett összeadási törvény miatt $P_1 \pmod p + P_2 \pmod p$ nem a „végtelen távoli pont”.

Tegyük fel, hogy $x_1 \equiv x_2 \pmod p$. Legyen először $P_1 = P_2$.

$$2P_1 \pmod p = (x_3 \pmod p, y_3 \pmod p)\,,$$

ahol

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$

és

$$y_3 = -y_1 + \left(\frac{3x_1^2 + a}{2y_1}\right) \cdot (x_1 - x_3)\,.$$

Belátjuk, hogy $p \nmid 2y_1$. Ha ugyanis $p \mid 2y_1$, akkor amiatt, hogy x_3 nevezőjének p nem osztója, ezért $3x_1^2 + a$ számlálója p többszöröse lenne. De ekkor x_1 az $x^3 + \overline{a}x + \overline{b} \in E \pmod p$ polinomnak gyöke, továbbá x_1 gyöke a $3x^2 + \overline{a}$ polinomnak is. Tehát $x^3 + \overline{a}x + \overline{b} \in E(\mathbb{F}_p)$-nek van többszörös gyöke, ezért diszkriminánsa

$$4\overline{a}^3 + 27\overline{b} \equiv 0 \pmod p\,,$$

ami ellentmond feltételünknek.

Tegyük fel, hogy $P_1 \neq P_2$. Mivel $x_1 \equiv x_2 \pmod{p}$, $x_1 \neq x_2$, ezért $x_2 = x_1 + p^r x$, $r \geq 1$, és x számlálója is, nevezője is relatív prím n-hez. Mivel feltevésünk szerint $P_1 + P_2$ koordinátáinak nevezői nem oszthatók p-vel, ezért $P_1 * P_2 = (x_3, y_3)$ pontra az

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - (x_1 - x_2)$$

és

$$y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \cdot (x_1 - x_3)$$

formulákat alkalmazva adódik, hogy $y_2 = y_1 + p^r y$.

Másrészt

$$\begin{aligned} y_2^2 &= (x_1 + p^r x)^3 + a(x_1 + p^r x) + b = \\ &= x_1^3 + ax_1 + b + p^r x(3x_1^2 + a) = \\ &= y_1^2 + p^r x(3x_1^2 + a) \pmod{p^{r+1}} . \end{aligned}$$

Mivel $x_1 \equiv x_2 \pmod{p}$, $y_1 \equiv y_2 \pmod{p}$, ezért $P_1 \equiv P_2 \pmod{p}$, és

$$P_1 \pmod{p} + P_2 \pmod{p} = 2P_1 \pmod{p} ,$$

és ez akkor és csak akkor $O \pmod{p}$, ha $y_1 = y_2 \equiv 0 \pmod{p}$.

Ha ez teljesül, akkor $p^{r+1} \mid y_2^2 - y_1^2$, és így $p^{r+1} \mid 3x_1^2 + a$, és így $3x_1^2 + a \equiv 0 \pmod{p}$, azaz az $x^3 + \overline{a}x + \overline{b}$ polinomnak van többszörös gyöke. Tehát

$$P_1 \pmod{p} + P_2 \pmod{p} \neq O \pmod{p} .$$

Fordítva, tegyük fel, hogy minden $p \mid n$-re $P_1 \pmod{p} + P_2 \pmod{p} \neq O \pmod{p}$. meg kell mutatni, hogy $P_1 + P_2$-ben a koordináták nevezői relatív prímek n-hez, azaz nem oszthatók p-vel, minden $p \mid n$-re.

Legyen $p \mid n$. Ha $x_2 \not\equiv x_1 \pmod{p}$, akkor az összegzési képletből világos, hogy $P_1 + P_2$ koordinátáiban a nevezők relatív prímek n-hez. Tegyük fel, hogy $x_2 \equiv x_1 \pmod{p}$. Ekkor $y_2 \equiv \pm y_1 \pmod{p}$, és $P_1 \pmod{p} + P_2 \pmod{p} \neq O \pmod{p}$ miatt $y_2 \equiv y_1 \pmod{p}$, továbbá $y_2 \not\equiv 0 \pmod{p}$.

Ha $P_1 = P_2$, akkor a $2P_1$ formulára vonatkozó képletből rögtön adódik, hogy $2P_1$-ben a nevezők relatív prímek p-hez.

Ha $P_1 \neq P_2$, $x_2 = x_1 + p^r x$, x relatív prím p-hez. Az összegzési formulából kapjuk, hogy

$$\frac{y_2^2 - y_1^2}{x_2 - x_1} \equiv 3x_1^2 + a \pmod{p} .$$

Mivel $p \nmid y_2 + y_1 = 2y_1 \pmod{p}$, ezért relatív prím az

$$\frac{y_2^2 - y_1^2}{(y_2 + y_1)(x_2 - x_1)} = \frac{y_2 - y_1}{x_2 - x_1}$$

nevezőjéhez, és így az összegzési formulával az állítás adódik. ■

Könnyű belátni, hogy ha $P = (x, y)$, racionális pont az E görbén, akkor $x = p/e^2$, $y = q/e^3$, $(p, e) = (q, e) = 1$. Legyen ugyanis $x = u/M$, $y = v/N$, ahol $M, N \geq 1$, és $(u, M) = (v, N) = 1$. Ekkor

$$\frac{v^2}{N^2} = \frac{u^3}{M^3} + a \cdot \frac{u^2}{M^2} + b \cdot \frac{u}{M} + c \ ,$$

és így

$$M^3 v^2 = N^2 u^3 + aN^2 M u^2 + bN^2 M^2 u + cN^2 M^2 \ , \tag{22.22}$$

ezért $N^2 \mid M^3 v^2$, $N^2 \mid M^3$. Belátjuk, hogy $M^3 \mid N^2$. Nyilvánvaló, hogy $M \mid N^2 u^2$, és (u, M)=1 miatt $M \mid N^2$. Ezért $M^2 \mid N^2 u^2$, így $M \mid N$. Még egyszer megvizsgálva a (22.22) egyenletet, $M^3 \mid N^2 u^3$, $M^3 \mid N^2$ adódik, tehát $M^3 = N^2$.

Ezek után ismertetjük ***Lenstra elliptikus görbe algoritmusát,*** amelynek bemenete egy olyan $n \geq 3$ összetett egész szám, és $(n, 6) = 1$, kimenete pedig n-nek egy nemtriviális D osztója. Megjegyezzük még, hogy x_1, y_1 jelöli tetszőleges $P \in E(\mathbb{Q})$ pont koordinátáit, ahol $E : \ y^2 = x^3 + ax + b$ elliptikus görbe. Ahogy az előzőekben láthattuk, megfelelő d, p, q értékekkel $x_1 = p/d^2$, $y_1 = q/d^3$ írható. Feltesszük, hogy a KOORDNEV(P) függvény a d értékkel tér vissza.

ELLIPTIKUS-LENSTRA(n)

```
 1  a ← RANDOM(1, n)
 2  x_1 ← RANDOM(1, n)
 3  y_1 ← RANDOM(1, n)
 4  b ← y_1^2 − x_1^3 − a x_1  (mod n)
 5  D ← (4a^3 + 27b^2, n)
 6  if 1 < D < n
 7     then return D
 8  if D = n
 9     then goto 1
10  k ← LKKT(1, 2, ..., K)
11  d ← KOORDNEV(k · P)
12  D ← (d, n)
13  if 1 < D < n
14     then return D
15  if D = 1
16     then K ← K + 1
17          goto 10
18     else return SIKERTELEN
```

Az elliptikus görbék elméletére alapozva az elmúlt évtizedekben a gyakorlati életben talán legjobban használható prímtesztelő algoritmusokat fejlesztettek. Ezek részletes tárgyalása további fejezeteket igényelne, ezért jelen mű keretében csak vázlatos ismertetésre szorítkozunk.

Az elliptikus görbék felhasználásával történő prímtesztelésben használt algoritmusok alapötlete a következő tételre vezethető vissza, amelyet bizonyítás nélkül közlünk.

22.59. tétel. *Legyen $n \in \mathbb{N}$, $(6,n) = 1$, és E_n egy $\mathbb{Z}/n\mathbb{Z}$ feletti elliptikus görbe pontjainak a halmaza. Legyenek m, s egészek, és $s \mid m$. Tegyük fel, hogy találunk olyan $P \in E_n$ pontot, amelyre*

$$m \cdot P = O \tag{22.23}$$

és

$$\frac{m}{q} \cdot P \neq O \tag{22.24}$$

s minden q prímfaktorára fennáll. Ekkor n minden p prímosztójára

$$\left|E_p\right| \equiv 0 \pmod{s} .$$

Továbbá, ha

$$s > \left(\sqrt[4]{n} + 1\right)^2 ,$$

akkor n prím.

Először egy általános sémát vázolunk fel, amelynél a 22.59. tétel jelöléseit használjuk. A bemeneti érték $n > 1$ egész szám, amelyre $(6,n) = 1$.

ELLIPTIKUS-PRÍMTESZT(n)

1 $E_n \leftarrow$ véletlen ($\mathbb{Z}/n\mathbb{Z}$) feletti elliptikus görbe
2 $m \leftarrow |E_n|$
3 **if** $m = q \cdot s$, ahol s „valószínűleg prím” és q faktorizációja ismert
4 **then goto** 8
5 **else goto** 1
6 **if** $n_1 \leq \left(\sqrt[4]{n} + 1\right)^2$
7 **then goto** 1
8 $P \leftarrow$ véletlen pont (E_n)-ből
9 **if** qP nem definiált
10 **then return** ÖSSZETETT
11 **if** $(m/q) \cdot P = O$
12 **then goto** 8
13 **if** $m \cdot P \neq O$
14 **then return** ÖSSZETETT
15 **while** nem vagyunk biztosak
16 **do** ELLIPTIKUS-PRÍMTESZT(s)

A 13. lépésben a „nem vagyunk biztosak” azt jelenti, hogy ha s értéke túl nagy, akkor túl sok időt vesz igénybe a prímtesztje. Mivel s folyamatosan csökken minden rekurziós lépésben, minden sikeres $m = qs$ felbontásnál, ezért előbb-utóbb „elég kicsi” lesz ahhoz, hogy könnyen ellenőrizzük prím mivoltát. Ezt például egyszerűen próbaosztással is megtehetjük.

Goldwasser és Kilian javasolták ezt a gondolatmenetet. Az ő eljárásukban rögzített f értéket használtak, nevezetesen a 2-t, ami megnehezíti a megfelelő m érték megtalálását.

Ez amúgy is nehéz feladat, annak ellenére, hogy létezik m meghatározására polinomiális idejű algoritmus. Feltesszük, hogy a RANDOM-GÖRBE(S) eljárás adott S algebrai struktúrához véletlenszerűen szolgáltat egy $E(S)$ elliptikus görbét. A RANDOM-PONT(E_n) függvény egy véletlen $P \in E_n$ pontot ad visszatérési értékként.

GOLDWASSER–KILIAN-PRÍMTESZT(n)

```
 1  E_n ← RANDOM-GÖRBE(ℤ/nℤ)
 2  m ← |E_n|
 3  if m = 2n_1 nem áll elő úgy, hogy n_1 „valószínűleg prím”
 4     then goto 1
 5  P ← RANDOM-PONT(E_n)
 6  if 2P nem definiált
 7     then return ÖSSZETETT
 8  if 2 · P = O
 9     then goto 5
10  if mP ≠ O
11     then return ÖSSZETETT
12  if n_1 biztosan prím
13     then return PRÍM
14     else GOLDWASSER-KILIAN-PRÍMTESZT(n_1)
```

Az ***Atkin-teszt*** kiváló példa egy olyan pontra, ahol a számelmélet három területe, a prímtesztelés, az elliptikus görbék és a kvadratikus testek elmélete ***összetalálkozik.*** Itt az alapgondolat az, hogy először nem az elliptikus görbét választjuk ki, hanem a rendjét. Ez úgy történik, hogy a $\mathbb{Q}(\sqrt{(D)})$ képzetes kvadratikus test algebrai egészei közt keresgélünk, és ha találunk olyan ν-t, amelyre $|\nu|^2 = n$, akkor ***érdemes*** olyan m számmal kísérletezni, amelyre $m = |\nu \pm 1|^2$, azért, mert ha van ilyen és megfelelő módon faktorizálni is lehet, akkor a megfelelő $E(\mathbb{Z}/n\mathbb{Z})$ elliptikus görbe viszonylag könnyen megtalálható. Megfelelő alatt azt értjük, hogy a csoport rendje m, azaz $|E(\mathbb{Z}/n\mathbb{Z})| = m$.

A felhasznált kvadratikus testben nem mindegy, hogy hogyan választjuk meg az úgynevezett D ***alapdiszkriminánst.*** D negatív egésznek rendelkeznie kell a következő tulajdonságokkal: $D \equiv 0 \pmod 4$, vagy $D \equiv 1 \pmod 4$, minden $k > 1$ egészre D/k^2 nem alapdiszkrimináns, $D < -7$. A NEXTD() eljárás egy megfelelő D értéket szolgáltat az alapdiszkriminánsok halmazából.

A gyakorlat azt mutatja, hogy a fenti feltételekkel rendelkező alapdiszkriminánsok közül nem mindegyik „viselkedik” egyformán, ezért nem véletlenszerűen választunk. A választás stratégiáját most nem részletezzük, csak megemlítjük, hogy például a csupa kis faktorból álló alapdiszkriminánsok választása az átlagosnál nagyobb sikerrel kecsegtet, azaz kisebb eséllyel tér vissza az algoritmus az első lépéshez új D értéket választani.

Adott n esetén és a hozzá megtalált D értékkel kiszámíthatjuk az úgynevezett ***Hilbert-polinomot,*** amelynek tetszőleges $\pmod n$ vett gyökének segítségével megadhatunk két elliptikus görbét, amelyek rendje $m = |\nu \pm 1|^2$. Legyen HILBERT(n, D) az a függvény, amely adott n-hez és D-hez szolgáltatja a Hilbert polinom egy gyökét.

Felhasználjuk továbbá a Goldwasser–Kilian teszt ellenőrző részét, de nem korlátozzuk f értékét. A BIZONYÍTÉK függvény bemeneti értékei: E_n, m, f ahol természetesen m már keresztülment a megfelelő ellenőrzéseken, azaz felírható $m = f \cdot n_1$ alakban, ahol f fak-

torizációja ismert, és n_1 valószínűleg prím. A kimeneti érték lehet összetett, ha biztosan megkapjuk, hogy n összetett. Lehet nem bizonyíték, ha a 7. lépésben tér vissza a program. Ez azt jelenti, hogy n vagy összetett, vagy a lehetséges kettőből a másik elliptikus görbét kellett volna választanunk. Végül lehet bizonyíték a kimenet, ekkor következhet a rekurzió következő lépése.

Bizonyíték(E_n, m, f)

```
1  P ← Random(E_n)
2  if f · P nem definiált
3     then return összetett
4  if f · P = O
5     then goto 1
6  if mP ≠ O
7     then return nem
8  return igen
```

Most már nekifoghatunk az Atkin-teszt ismertetésének.

Atkin-prímteszt(n)

```
1  D ← NextD()
2  ω ← (D + √D)/2
3  if ∃ x, y ∈ ℤ : 4n = (2x + yD)² − y²D
4     then ν ← x + yω
5     else goto 1
6  m ← |ν + 1|²
7  if m = f · n_1, ahol n_1 „valószínűleg prím” és n_1 > (∜n + 1)²
8     then goto 12
9  m ← |ν − 1|²
10 if m = f · n_1 nem áll elő úgy, hogy n_1 „valószínűleg prím”) és n_1 > (∜n + 1)²
11    then goto 1
12 x_0 ← Hilbert(n, D)
13 c ← tetszőleges egész, amire (c/n) = −1
14 k ← tetszőleges egész, amire k ≡ x_0/(1728 − x_0)  (mod n)
15 E_n ← {(x, y) | y² = x³ + 3kx + 2k}
16 if Bizonyíték(E_n, m, f) = összetett
17    then return összetett
18    else if Bizonyíték(E_n, m, f) = igen
19            then goto 23
20         E_n ← {(x, y) | y² = x³ + 3kc²x + 2kc³}
21 if Bizonyíték(E_n, m, f) = összetett vagy Bizonyíték(E_n, m, f) = nem
22    then return összetett
23 if n_1 biztosan prím
24    then return prím
25    else Atkin-prímteszt(n_1)
```

Gyakorlatok

22.6-1. Legyen $E: y^2 = x^2 + 2$ elliptikus görbe $\mathbb{F}_7$ felett. Adjuk meg $E(\mathbb{F}_7)$ elemeit.

22.6-2. Legyen $E: y^2 = x^3 + 1$ elliptikus görbe $\mathbb{F}_2$ felett. $E(\mathbb{F}_2)$ ciklikus csoport?

22.6-3. Írjunk programot, amely megvalósítja a POLLARD-PMÍNUSZEGY algoritmust.

Feladatok

22-1. Prímszámok vizsgálata

Írjunk programot, amely adott n pozitív egész számig ellenőrzi a páros Goldbach-sejtést, továbbá statisztikát készít a hézagokról: tárolja első előfordulásukat és előfordulásaik számát. Ehhez szükség van nagy számú prímszám előállítására. Oldjuk meg ezek tárolását a lehető legkevesebb memória felhasználásával.

22-2. A Brun-konstans kiszámolása

Írjunk olyan programot, amely a lehető legpontosabban adja meg a Brun-konstans értékét. Természetesen minél több ikerprímet találunk, annál pontosabb számolást végezhetünk. Érdekes programozói feladat annak megoldása, hogy hogyan tudunk nagy pontossággal lebegőpontos számokat ábrázolni.

22-3. Az* AKS-PRÍMTESZT *megvalósítása

Írjunk prímtesztelő programot az AKS-algoritmus felhasználásával. Tételezzünk fel 10^{10}-nél kisebb bemeneti értéket. Készítsük el az eljárás egy módosított változatát is, felhasználva a 22.55. sejtésben látottakat. Hasonlítsuk össze a programok futási idejét és tárigényét.

Megjegyzések a fejezethez

A tárgyalt témakörök megértéséhez szükséges alapvető számelméleti fogalmak találhatók Járai Antal [184] könyvében.

A körosztási polinommal, illetve egyéb véges testekkel kapcsolatos fogalmakról és tételekről Liedl és Niederreiter [227] könyvében találhatunk részletes információkat.

Goldfeld bizonyítását, miszerint pozitív valószínűséggel fordulnak elő olyan q prímek, melyekre

$$P(q-1) > q^{\frac{1}{2}+c} ,$$

ahol $c \approx 1/12$ a [137], míg ennek Fouvry által adott tökéletesítéséről a [118] publikációban olvashatunk.

A [33] dolgozatban találjuk a 22.53. lemma bizonyítását a $q^{0.6683}$ értékig.

Az AKS-algoritmus futási idejének elemzéséhez használt számítások, mint például számok és polinomok szorzásának, osztásának és összeadásának időigénye részletesen tanulmányozható von zur Gathen és Gerhard [128] könyvében. Ugyanitt találjuk az 1. programrész időigényének részletes vizsgálatához szükséges információkat.

A 22.55. sejtés részletes elemzésével [43] foglalkozik, míg helyességének ellenőrzése $r \leq 100$ és $n \leq 10^{10}$ esetre [194]-ben található.

Az állítás bizonyítása, miszerint az x-nél kisebb prímek szorzata legalább e^x, Apostol könyvében [21] olvasható.

További érdekes számelméleti algoritmusok találhatók Járai Antal digitális jegyzetében [183].

23. Osztott algoritmusok

Osztott rendszernek nevezzük az egymással kommunikáló önálló számító eszközöket. Ez a meghatározás nagyon tág, magában foglalja a VLSI áramköröket, a többprocesszoros rendszereket, a helyi hálózattal klaszterbe kötött munkaállomásokat és az Internetet. Itt most a lazábban összekötött rendszerekre koncentrálunk. Úgy tekintjük, hogy egy osztott rendszerben az egyes processzorok lényegében függetlenek, de bizonyos okokból – ilyenek például az erőforrás-megosztás, a hozzáférhetőség, a hibatűrés – összhangba kell hozniuk a tevékenységüket.

Bár hatalmas igény mutatkozik az osztott rendszerekre, közismerten nehéz olyan hatékony osztott algoritmusokat készíteni, melyek jól teljesítenek valódi rendszereken. A nehézségek nem csak gyakorlati természetűek, hanem elméletiek is. Nevezetesen, a következő három tényező sok problémát okoz: aszinkronitás, korlátozott helyi információk, meghibásodások. Az aszinkronitás azt jelenti, hogy nem feltétlenül áll rendelkezésre egy globális idő, valamint, hogy az egyes számító eszközökön bekövetkező események abszolút és relatív időpontjai gyakran nem pontosan ismertek. Továbbá, az egyes számító eszközök csak a kapott információkat ismerik, így csak helyi nézetük van a rendszer globális állapotáról. Végül, a számító eszközök és a hálózati komponensek egymástól függetlenül meghibásodhatnak, azaz némelyek működőképesek maradnak, míg mások nem.

Az osztott rendszerek elemzésére használt modellek leírását a számítás üzenetátadási modelljével kezdjük. Bemutatunk és elemzünk néhány, ezeken a modelleken alapuló, osztott algoritmust. Tárgyaljuk a hibatűrést az osztott rendszerekben és megvizsgálunk néhány olyan algoritmust, mely megoldást jelent az üzenetátadási modellekben a hibák kezelésére. Mivel az osztott rendszerekben a globális idő gyakran nem elérhető, bemutatunk néhány megközelítést az ún. logikai idő meghatározására, mely lehetővé teszi az okozati viszony és az ellentmondásmentes állapotok megállapítását. Ezután haladóbb témákat veszünk górcső alá. Bemutatjuk azokat az üzenetszóró szolgáltatásokat, melyeket az osztott rendszerek gyakran használnak, és bemutatjuk azokat az algoritmusokat, melyek ezeket a szolgáltatásokat megvalósítják. Bemutatunk néhány információgyűjtő algoritmust is. Végül az osztott számítások közös memóriát használó modelljének kölcsönös kizárási problémáját tárgyaljuk.

23.1. Üzenetküldő rendszerek és algoritmusok

Az osztott számítás első modelljeként az üzenetküldő rendszerek meghibásodás nélküli üzenetküldési modelljét tárgyaljuk. Figyelembe vesszük mind a szinkron, mind az aszinkron rendszereket, és bemutatunk néhány kiválasztott algoritmust, amelyek tetszőleges hálózati topológiájú üzenetküldő rendszerekben alkalmazhatók – mind szinkron, mind pedig aszinkron esetekben.

23.1.1. Üzenetküldő rendszerek modellezése

Egy üzenetküldő rendszerben a processzorok[1] kommunikációs csatornákon keresztül küldött üzenetekkel kommunikálnak, ahol az egyes csatornák kétirányú kapcsolatot biztosítanak a két kérdéses processzor között. A csatornákkal meghatározott kapcsolódási mintát a rendszer *topológiájának* nevezzük. Ezt a topológiát egy irányítatlan gráffal ábrázoljuk, ahol az egyes csúcsok egy processzort jelentenek, és akkor és csak akkor van egy él két csúcspont között, ha van egy csatorna a csúcspontokkal ábrázolt processzorok között. A csatornák összességét *hálózatnak* is nevezzük. Egy adott topológiájú üzenettovábbító rendszer algoritmusa a rendszer egyes processzoraira vonatkozó helyi programból áll. Ez a helyi program lehetővé teszi, hogy a processzor helyi számításokat végezzen és az adott topológia melletti szomszédainak üzeneteket küldjön, illetve azoktól üzeneteket kapjon.

A rendszer egyes processzorait egy-egy olyan automatával modellezzük, amely végtelen is lehet. ***Konfigurációnak*** nevezünk egy $C = (q_0, \ldots, q_{n-1})$ vektort, ahol a q_i-k, az egyes P_i processzorok állapotai. A rendszerben zajló tevékenységeket oszthatatlan rendszerműveleteket leíró ***eseményeknek*** (vagy ***műveleteknek***) nevezzük. Esemény például a helyi számítási esemény vagy az olyan kézbesítési esemény, ahol egy processzor egy üzenetet kap. A rendszer időbeli működését egy ún. ***végrehajtási sorozattal*** (röviden: végrehajtás) modellezzük, ami C_i konfigurációkat és a_i eseményeket váltakozva tartalmazó, véges vagy végtelen sorozat: $C_0, a_1, C_1, a_2, C_2, \ldots$. A rendszer modelljétől függően, a végrehajtásnak meg kell felelnie számos, helyességi tulajdonságokat leíró feltételnek. E feltételeket vagy az ún. *biztonsági* vagy az ún. *élőségi* osztályokba sorolhatjuk. Egy rendszer *biztonsági* feltétele egy olyan feltétel, melynek igaznak kell lennie a rendszer bármely eseménye előtt. Informálisan ez azt jelenti, hogy még semmi *rossz* nem történt. Egy *létezési* feltétel egy olyan feltétel, melynek igaznak kell lennie számos (valószínűleg végtelen) alkalommal. Informálisan ez azt jelenti, hogy végül is valami *jó* be fog következni. Egy fontos létezési feltétel a ***tisztaság,*** mely megköveteli, hogy egy (végtelen) végrehajtás végtelen sok processzorműveletet tartalmazzon, hacsak néhány konfiguráció után az adott processzoron nincs engedélyezett művelet.

23.1.2. Aszinkron rendszerek

Egy rendszerre akkor mondjuk, hogy *aszinkron,* ha egy üzenet kézbesítési idejének vagy egy processzor egymás után tett lépései közti időnek nincs rögzített felső korlátja. Az aszinkron rendszer nyilvánvaló példája az Internet. Egy osztott rendszer megvalósításában gyakran létezik felső korlát az üzenetek kézbesítési idejére és a processzorlépések idejére. Mivel ezek

[1] A rendszer elemeit a továbbiakban processzoroknak tekintjük. *A lektor.*

a felső korlátok gyakran nagyon nagyok és időben változnak, gyakran kívánatos egy olyan algoritmus kifejlesztése, mely független az időzítési paraméterektől, vagyis aszinkron.

Az aszinkron modellben egy végrehajtás ***elfogadható,*** ha az egyes processzorok végtelen számú számítási eseménnyel rendelkeznek, és az összes küldött üzenet végül kézbesítődik. Az első követelmény modellezi azt a tényt, hogy a processzorok nem hibásodnak meg. (Ez nem azt jelenti, hogy a processzor helyi programja végtelen ciklust tartalmaz. Egy algoritmus még mindig megállhat, ha olyan az átmenetfüggvénye, amely egy bizonyos pont után nem változtatja meg a processzor állapotát.)

Feltesszük, hogy minden processzor állapothalmaza tartalmaz egy olyan részhalmazt, mely a *megállt* állapotokat tartalmazza. Ha egy processzor egy ilyen állapotba kerül, akkor abban is marad. Azt mondjuk, hogy az algoritmus *megállt,* ha az összes processzor megállt állapotban van, és nincs kézbesítés alatt álló üzenet.

Az aszinkron modellben egy algoritmus ***üzenetszáma*** az elfogadható végrehajtási sorozatokban csúcsponttól csúcsponthoz küldött üzenetek számának maximuma.

Az ***időzített végrehajtás*** olyan végrehajtás, melynél minden eseményhez egy nemnegatív valós számot rendelünk, az esemény bekövetkezésének *időpontját.* Egy aszinkron algoritmus *futási idejének* méréséhez először feltesszük, hogy bármely végrehajtás esetén az üzenetek kézbesítési ideje egy időegység. Így a ***futási idő*** az a maximális idő, míg az összes olyan időzített elfogadható végrehajtás megáll, ahol a üzenetek kézbesítési ideje legfeljebb egy. Intuitív módon ezt úgy tekinthetjük, hogy vesszük az algoritmus bármelyik végrehajtását, és úgy normalizáljuk, hogy a leghosszabb kézbesítési időt tekintjük egy időegységnek.

23.1.3. Szinkron rendszerek

A szinkron modellben a processzorok zárt lépéseket végeznek. A végrehajtást olyan menetekre bontjuk, ahol az egyes processzorok egy-egy üzenetet küldhetnek szomszédaiknak, az üzenetek megérkeznek, és minden processzor az épp kapott üzeneteket alapul véve számol. Ez a modell nagyon kényelmes az algoritmusok tervezése során. Az ebben a modellben tervezett algoritmusok sok esetben könnyen szimulálhatók úgy, hogy más, a valóságot jobban tükröző időzítési modellben is működjenek.

A szinkron modellben azt mondjuk, hogy egy végrehajtás elfogadható, ha az végtelen. A menet alapú struktúrából az következik, hogy minden processzor végtelen sok számítási lépést tesz, és minden egyes üzenet valamikor megérkezik. Így egy hiba nélküli szinkron rendszerben ha a (determinisztikus) algoritmus rögzítésre került, az egyetlen megváltoztatható szempont egy végrehajtás meghatározásánál a kezdőkonfiguráció. Másrészt egy aszinkron rendszerben ugyanannak az algoritmusnak több, különböző végrehajtása lehet még azonos kezdőkonfiguráció és hibamentesség esetén is, mivel itt a processzorlépések egymásutánisága és az üzenetek késése nem rögzített.

A *megállt állapot* fogalma és az algoritmus *megállása* eltér az aszinkron modellétől.

A szinkron modellben az algoritmus ***üzenetszáma*** ugyanúgy az összes küldött üzenet számának maximuma az algoritmus összes elfogadható végrehajtása esetén, mint az aszinkron modellben.

Szinkron esetben az idő méréséhez egyszerűen összeszámoljuk a megállásig megtett menetek számát. Így a szinkron modellben egy algoritmus *futási ideje* az algoritmus összes elfogadható végrehajtásainak megállásig megtett menetei számának maximuma.

23.2. Alapvető algoritmusok

Az üzenetküldő modell néhány egyszerű algoritmusával kezdjük.

23.2.1. Üzenetszórás

Egy, az (együzenetes) üzenetszórási problémára vonatkozó egyszerű algoritmussal kezdjük. Feltesszük, hogy az n csúcsú hálózatgráf feszítőfája már adott. Később majd eltekintünk ettől a feltevéstől. A P_r processzor egy M üzenetet kíván küldeni az összes többi processzornak. A P_r gyökerű feszítőfa osztott módon van karbantartva: Minden egyes processzor rendelkezik egy megkülönböztetett csatornával a fabeli *szülője* felé, illetve csatornák egy halmazával a fabeli *gyerekei* felé. A P_r gyökér az M üzenetet elküldi a gyerekei felé vezető összes csatornán. Amikor egy processzor megkapja az M üzenetet egy csatornán a szülőjétől, akkor azt elküldi az összes gyerekének.

FESZÍTŐFA-ÜZENETSZÓRÓ

```
    Kezdetben M útban van P_r-től az összes feszítőfabeli gyereke felé.
    Kód P_r esetén:
1       üres üzenet fogadásakor: // P_r első számítási eseménye
2           leállás

    Kód P_j, 0 ≤ j ≤ n − 1, j ≠ r esetén:
3       M szülőtől való fogadásakor:
4           M küldése az összes gyerekhez
5           leállás
```

A FESZÍTŐFA-ÜZENETSZÓRÓ algoritmus helyes akár szinkron akár aszinkron rendszerről van szó. Továbbá, az üzenetszámok és a futási idők is azonosak mindkét modellben.

Egyszerű induktív bizonyítással először bebizonyítunk egy lemmát, mely szerint a t-edik menet végén az M eléri az összes, a feszítőfában P_r-től t (vagy rövidebb) távolságra lévő processzort.

23.1. lemma. *A szinkron modellben az üzenetszóró algoritmus minden elfogadható végrehajtási sorozatában az összes, a feszítőfában P_r-től t távolságra lévő processzor megkapja az M üzenetet a t-edik menetben.*

Bizonyítás. A tetszőleges P_r processzortól számított t távolság szerinti indukciós feltétellel fogunk haladni. Először legyen $t = 1$. Az algoritmusból következik, hogy P_r minden egyes gyereke megkapja az üzenetet az első menetben.

Tegyük fel, hogy minden $t - 1$ távolságra lévő processzor megkapta az M üzenetet a $(t - 1)$-edik menetben. Meg kell mutatnunk, hogy minden P_t processzor, mely t távolságra van, megkapja az üzenetet a t-edik menetben. Legyen P_s P_t szülője a feszítőfában. Mivel P_s $t - 1$ távolságra van P_r-től, az indukciós feltétel szerint P_s megkapta az M üzenetet a $(t - 1)$-edik menetben. Az algoritmus alapján így P_t megkapja M-et a t-edik menetben. ■

A 23.1. lemma alapján az üzenetszóró algoritmus futási ideje d, ahol d a feszítőfa magassága. Mivel d legfeljebb $n-1$ (amikor a feszítőfa egy lánc), azt kapjuk, hogy:

23.2. tétel. *Ha egy d magasságú gyökeres feszítőfa előre ismert, n processzor esetén létezik egy szinkron üzenetszóró algoritmus, melyre az üzenetszám $n-1$ és a futási idő d.*

Most megvizsgáljuk az aszinkron rendszert és egy hasonló elemzést végzünk.

23.3. lemma. *Az aszinkron modellben az üzenetszóró algoritmus minden egyes elfogadható végrehajtása esetén minden – P_r-től a feszítőfában t távolságra lévő – processzor megkapja az M üzenetet a t időpontig.*

Bizonyítás. A P_r processzortól számított t távolság szerinti indukcióval fogunk eljárni. Az algoritmusból következik, hogy M kezdetben útban van P_r-től 1 távolságra lévő P_i processzorok felé. Az aszinkron modell futási idejének meghatározása szerint P_i megkapja az M üzenetet az 1 időpontig.

Tegyük fel, hogy minden, $t-1$ távolságra lévő processzor megkapta az üzenetet a $t-1$ időpillanatban. Meg kell mutatnunk, hogy minden t távolságra lévő P_t processzor megkapja az üzenetet a t időpontig. Legyen P_s P_t szülője a feszítőfában. Mivel P_s $t-1$ távolságra van P_r-től, az indukciós feltevés miatt, P_s M-et továbbküldi P_t-nek, amikor a $t-1$ időpontban megkapja azt. Az algoritmus szerint P_t így megkapja M-et a t időpontig. ■

Ebből közvetlenül kapjuk a következő tételt:

23.4. tétel. *Ha egy d magasságú gyökeres feszítőfa előre ismert, n processzor esetén létezik egy aszinkron üzenetszóró algoritmus, melyre az üzenetszám $n-1$ és a futási idő d.*

23.2.2. A feszítőfa megkonstruálása

A következőkben tárgyalt ELÁRASZTÁS nevű aszinkron algoritmus felépíti a kijelölt P_r csúcsnál lévő gyökerű feszítőfát. Az algoritmus hasonlít a MÉLYSÉGI-KERESÉS (MK) algoritmushoz. Az MK algoritmustól eltérően azonban, ahol csak egyetlen processzor rendelkezik „globális ismerettel” a fáról, az ELÁRASZTÁS algoritmusban minden egyes processzornak „helyi ismerete” van a gráfról, a processzorok a munkájukat üzenetküldéssel koordinálják és a processzorok, valamint az üzenetek tetszőlegesen sokat késhetnek. Mindezek miatt az ELÁRASZTÁS algoritmus megtervezése és elemzése igazi kihívás, mivel meg kell mutatnunk, hogy az algoritmus ténylegesen felépít egy feszítőfát a késések kedvezőtlen összejátszása esetén is.

Az algoritmus leírása

Minden egyes processzor négy helyi változóval rendelkezik.

Az egy processzorhoz kapcsolódó összeköttetéseket 1-től kezdődő egyedi számokkal azonosítjuk, és egy *szomszédok*-nak nevezett helyi változóban tároljuk. Azt mondjuk, hogy a *feszítőfát felépítettük,* ha a *szülő* változó a processzor feszítőfabeli szülőjére mutató összeköttetés azonosítószámát tartalmazza, kivéve, ha a P_r kijelölt processzorról van szó. Ez utóbbi esetben a *szülő* értéke NINCS. A *gyerekek* változó a fában a gyerekekre mutató élek

azonosítóinak halmazát, az *egyéb* változó pedig az összes többi összeköttetés azonosítóinak halmazát tárolja. Így a feszítőfa ismeretét „oszthatjuk” a processzorok között.

Az egyes processzorokra vonatkozó kód szegmensekből áll össze. Az első szegmens (1–4. sorok) leírja, a processzorok helyi változóinak kezdőértékadását. Emlékezzük rá, hogy a helyi változók a nulladik időpillanat előtt kapnak kezdőértéket. A következő három szegmens (5–10., 11–14. és 15–18. sorok) leírja azokat a műveleteket, melyeket az egyes processzoroknak kell végrehajtani egy üzenet fogadásakor: <*kijelöl*>, <*jóváhagyva*> vagy <*visszautasítva*> . Az utolsó szegmens (19–21. sorok) csak a P_r processzor kódjára vonatkozik. Ez a szegmens csak akkor hajtódik végre, ha a helyi *szülő* változó értéke a P_r processzor esetén NIL. Valamely időpillanatban megtörténhet, hogy egy processzornak egynél több szegmenst kell végrehajtania (például akkor, ha a processzor két másik processzortól kapott <*kijelöl*> üzenetet). Ekkor a processzor a szegmenseket egymás után egyesével hajtja végre (egy processzoron futó szegmensek sohasem hajtódnak végre egyszerre). Azonban egy másik processzor műveletei tetszőlegesen hajtódhatnak végre a végrehajtás során. A feldolgozható összes üzenet végül feldolgozásra kerül és a végrehajtható szegmensek végül végrehajtódnak (tisztaság).

ELÁRASZTÁS

```
    Kód a P_k, 1 ≤ k ≤ n processzorok esetén
 1  kezdőértékek beállítása
 2      szülő ← NIL
 3      gyerekek ← ∅
 4      egyéb ← ∅

 5  üzenet feldolgozása, <kijelöl> érkezett a j élen
 6      if szülő = NIL
 7        then szülő ← j
 8             <jóváhagyva> küldése a j élre
 9             <kijelöl> az összes szomszédok \{j} halmazbeli összeköttetésre
10        else <visszautasítva> küldése a j összeköttetésre

11  üzenet feldolgozása <jóváhagyva> érkezett a j élen
12      gyerekek ← gyerekek ∪ {j}
13      if gyerekek ∪ egyéb = szomszédok \{szülő}
14        then terminate

15  üzenet feldolgozása < visszautasítva> érkezett a j élen
16      egyéb ← egyéb ∪ {j}
17      if gyerekek ∪ egyéb = szomszédok \{szülő}
18        then terminate

    Extra kód a kijelölt P_r processzorra
19  if szülő = NIL
20    then szülő ← NINCS
21         <kijelöl> küldése az összes szomszédok-beli élre
```

Az alábbiakban körvonalazzuk, hogyan működik az algoritmus. A kijelölt processzor egy <*kijelöl*> üzenetet küld az összes szomszédjának, és a NINCS értéket rendeli a *szülő* változóhoz (a NIL és a NINCS eltérő értékek, és eltérnek bármely természetes számtól), úgy, hogy sohasem küldi az üzenetet újra egyetlen szomszédjának sem.

Amikor egy processzor először feldolgoz egy <*kijelöl*> üzenetet, a saját *szülő* változóhoz rendeli annak az összeköttetésnek az azonosítóját, melyen az üzenet érkezett, és ugyanazon az összeköttetésen válaszol egy <*jóváhagyva*> üzenettel, valamin továbbítja a <*kijelöl*> üzenetet az összes többi összeköttetésen. Azonban, amikor egy processzor újra feldolgoz egy <*kijelöl*> üzenetet, a processzor a <*visszautasítva*> üzenettel válaszol, mert a *szülő* változó már nem NIL értékű.

Amikor egy processzor feldolgoz egy <*jóváhagyva*> üzenetet, a saját *gyerekek* halmazához adja annak az összeköttetésnek az azonosítóját, melyen az üzenet érkezett. Előfordulhat, hogy a *gyerekek* halmaz és az *egyéb* halmaz együtt tartalmazzák a processzorból induló összes összeköttetés azonosítóját a *szülő* változóban tárolt azonosítót kivéve. Ebben az esetben a processzor egy megállási állapotba kerül.

Amikor egy processzor feldolgoz egy <*visszautasítva*> üzenetet, a saját *egyéb* halmazához adja annak az összeköttetésnek az azonosítóját, melyen az üzenet érkezett. Ha a *gyerekek* és az *egyéb* halmazok uniója elég nagy, akkor a processzor szintén egy megállási állapotba kerül.

Helyesség bizonyítása

Most bebizonyítjuk, hogy az ELÁRASZTÁS algoritmus felépíti a feszítőfát. Az algoritmus végrehajtásának kulcspillanatai azok, amikor az egyes processzorok értéket rendelnek a *szülő* változóhoz. Ezek a hozzárendelések határozzák meg a feszítőfa „alakját” Azok a tények, hogy bármely processzor valamikor végül végrehajt egy utasítást, valamint hogy minden üzenet valamikor végül megérkezik, és hogy minden üzenet feldolgozásra kerül, biztosítják, hogy ezek a hozzárendelések végigfutnak a szomszédokon. Így az algoritmus kiterjed a gráf egy részfájára, jóllehet a terjedés meglehetősen lassú lehet. Végül a feszítőfa létrejön. Ha a feszítőfa létrejött, végül az egyes processzorok megállnak, bár néhány processzor még a fa létrejötte előtt megáll.

23.5. lemma. *Bármely $1 \leq k \leq n$ esetén létezik olyan t_k időpont, hogy az az első olyan pillanat, amikor pontosan k processzor esetén nem* NIL *a szülő változójának értéke, és ezek a processzorok és a szülő változók egy P_r gyökerű fát alkotnak.*

Bizonyítás. A bizonyítást k szerinti indukcióval végezzük. Az alapesetben tegyük fel, hogy $k = 1$. Figyeljük meg, hogy a P_r processzor valamikor *nincs* értéket rendel a *szülő* változójához. Legyen t_1 az a pillanat, amikor ez bekövetkezik. Ekkor a P_r kivételével az összes processzor esetén a *szülő* változó értéke még mindig NIL, mivel még <*kijelöl*> üzenet nem került elküldésre. A P_r processzor és a *szülő* változó egy egy csúcsú, él nélküli fát alkot. Így ezek egy gyökeres fát alkotnak. Ezzel beláttuk, hogy az induktív feltevés igaz a $k = 1$ esetben.

Induktív lépésként tegyük fel, hogy $1 \leq k < n$ és azt, hogy az induktív feltétel igaz k-ra. Tekintsük azt a t_k pillanatot, amikor pontosan k olyan processzor van, melynél a *szülő* változó értéke nem NIL. Mivel $k < n$, létezik egy olyan processzor, mely nincs a fában. Mivel a G hálózatgráfgráf összefüggő, létezik egy olyan – nem a fában lévő – processzor, mely szomszédos a fával. (A processzorok bármely T részhalmaza esetén egy P_i T-vel akkor és

csak akkor *szomszédos,* ha létezik egy olyan él a G gráfban, amely P_i-től egy T-beli processzorba vezet.) Emlékezzünk rá, hogy a meghatározás szerint egy ilyen processzor *szülő* változójának értéke NIL. Az induktív feltétel szerint a k processzoroknak a kódjuk 7. sorát már végre kellett hajtaniuk, így már küldtek, vagy valamikor a jövőben küldenek egy <*kijelöl*> üzenetet az egyes szomszédaiknak a *szülő* változóban meghatározott összeköttetés kivételével az összes összeköttetésen. Így a fával szomszédos nem fabeli processzorok már kaptak vagy valamikor a jövőben kapni fognak egy <*kijelöl*> üzenetet. Emiatt végül az összes szomszédos processzor egy NIL-től eltérő értéket fog rendelni a *szülő* változójához. Legyen $t_{k+1} > t_k$ az első pillanat, amikor valamely processzor végrehajt egy ilyen hozzárendelést, és jelöljük ezt a processzort P_i-vel. Ez nem lehet egy fabeli processzor, mivel a fabeli processzorok még egyszer már nem rendelnek értéket a *szülő* változójukhoz. Lehet P_i olyan processzor, mely nem része a fának, de nem is szomszédos vele? Nem, mert egy ilyen processzor nem rendelkezik a fával való közvetlen összeköttetéssel, így nem kaphat <*kijelöl*> üzenetet közvetlenül a fából, és így ez azt jelentené, hogy valamely t_k és t_{k+1} közti t' időpontban valamely nem fabeli P_j <*kijelöl*> üzenetet küldött P_i-nek, s így P_j-nek NIL-től eltérő értéket kellett rendelnie a *szülő* változójához valamikor t_k után, de t_{k+1} előtt, ami ellentmond annak a ténynek, hogy t_{k+1} az első ilyen pillanat. Következésképp P_i egy olyan nem fabeli processzor, mely szomszédos a fával, s mint ilyen a t_{k+1} pillanatban P_i a *szülő* változójához rendeli egy olyan összeköttetés azonosítószámát, mely egy fabeli processzorhoz kapcsolja. Tehát a t_{k+1} az első olyan pillanat, amikor pontosan $k+1$ olyan processzor van, melynél a *szülő* változó értéke nem NIL, és ekkor ezek a processzorok és a *szülő* változóik egy P_r gyökerű fát alkotnak. Ezzel teljes az induktív lépés és a lemma bizonyítása. ■

23.6. tétel. *Valamikor a jövőben minden egyes processzor megáll, és amikor már minden processzorok megállt, a szülő változók által indukált részgráf egy P_r gyökerű feszítőfa lesz.*

Bizonyítás. A 23.5. lemma alapján tudjuk, hogy létezik egy t_n pillanat, amikor az összes processzor és a *szülő* változóik értéke egy feszítőfát alkotnak.

Előfordulhat-e, hogy minden processzor megáll t_n előtt? A kód vizsgálatával az kapjuk, hogy egy processzor csak akkor áll meg, ha <*visszautasítva*> vagy <*jóváhagyva*> üzenetet kapott az összes szomszédjától a *szülő* változó által mutatott szomszédja kivételével. Egy processzor ilyen üzenetet csak a processzor által küldött <*kijelöl*> üzenetre adott válaszként kaphat. A t_n időpillanatban van egy olyan processzor, mely még nem küldött <*kijelöl*> üzenetet. Emiatt nem igaz az, hogy minden processzor már megállt a t_n pillanatig.

Megáll-e minden processzor végül? Megjegyezzük, hogy a t_n pillanatig minden egyes processzor vagy már küldött, vagy végül küld egy <*kijelöl*> üzenetet a *szülő* változó által mutatott szomszéd kivételével az összes szomszédnak.. Amikor egy processzor megkap egy <*kijelöl*> üzenetet, a processzor válaszol egy <*visszautasítva*> vagy egy <*jóváhagyva*> üzenettel, még akkor is, amikor a processzor leállt. Így minden egyes processzor valamikor végül kap egy <*visszautasítva*> vagy egy <*jóváhagyva*> üzenetet az összes olyan összeköttetésen, melyen a processzor egy <*kijelöl*> üzenetet küldött. Így végül minden processzor leáll. ■

Figyeljük meg, hogy az a tény, hogy egy processzor leállt, nem jelenti azt, hogy a feszítőfa elkészült. Valójában előfordulhat, hogy a hálózat egyik részében lévő processzorok

megállnak akkor, amikor a hálózat egy másik részének processzorai még egy üzenet sem kaptak. processzor megállt.

23.7. tétel. *Az* ELÁRASZTÁS *algoritmus üzenetszáma* $O(e)$, *ahol* e *a* G *gráfban lévő élek száma.*

A tétel bizonyítását meghagyjuk feladatnak (lásd 23-1. feladat).

Gyakorlatok

23.2-1. Előfordulhat, hogy egy processzor már leállt, bár még nem kapott egyetlen üzenetet sem. Egy egyszerű hálózatpéldán mutassuk be, hogyan lehet az üzenettovábbítást és a processzor számítását úgy késleltetni, hogy ez valójában megtörténjen.

23.2-2. Előfordulhat, hogy egy processzor már leállt, de még nem válaszolt egy üzenetre sem. Egy egyszerű hálózatpéldán mutassuk be, hogyan lehet az üzenettovábbítást és a processzor számítását úgy késleltetni, hogy ez valójában megtörténjen.

23.3. Gyűrűs algoritmusok

Egy osztott rendszerben gyakran kell koordinálni a processzorok tevékenységét. Ez gyakran egyszerűsíthető, ha van egy olyan processzor, mely koordinátorként működik. Előfordulhat, hogy kezdetben nincs a rendszernek koordinátora vagy egy meglévő koordinátor meghibásodik, és egy másikat kell választani. Ez felveti azt a problémát, ahol a processzoroknak ki kell választaniuk pontosan egyet maguk közül: egy ***vezetőt***. Ebben a szakaszban speciális típusú hálózatokban vizsgáljuk ezt a problémát: gyűrűkben. A probléma megoldására kifejlesztünk egy aszinkron algoritmust. Mint ahogy demonstrálni fogjuk, az algoritmus aszimptotikusan optimális üzenetszámmal rendelkezik. Ebben a szakaszban megismerjük a soros algoritmusokban gyakran használt, a futási időt alacsonyan tartó, jól ismert oszd-meg-és-uralkodj technikának az osztott analógját. Az osztott rendszereknél ez a technika segít csökkenteni az üzenetszámot.

23.3.1. A vezetőválasztási probléma

A vezetőválasztási probléma az, hogy választani kell egy vezetőt egy processzorhalmazban. Formálisan minden processzor rendelkezik egy *vezető* nevű változóval, melynek kezdőértéke NIL. Egy algoritmusról azt mondjuk, hogy *megoldja a vezetőválasztás problémáját*, ha kielégíti a következő feltételeket:

1. Bármely végrehajtásban pontosan egy processzor valamikor IGAZ értéket rendel a *vezető* változójához, és
2. bármely végrehajtásban ha egy processzor egyszer már értéket rendelt a *vezető* változójához, akkor a változó értéke már nem fog változni.

Gyűrű modell

A vezetőválasztás problémáját egy speciális hálózatban, a gyűrűben fogjuk vizsgálni. Formálisan az n csúcsú osztott rendszert modellező G gráf egy egyszerű kört alkot, más él nincs

a gráfban. A processzorhoz kapcsolódó két összeköttetést ÓMJ (óramutató járása szerinti), illetve ÓMJE (óramutató járásával ellentétes) címkével látjuk el. A processzorok megegyeznek a gyűrű irányultságán, azaz ha egy üzenet ÓMJ irányban halad tovább n lépésben, akkor ezzel bejárja az összes csúcspontot és visszakerül az kezdeti küldőhöz. Ugyanez igaz az ÓMJE irányra is. Minden processzor rendelkezik egy egyedi *azonosítóval*, mely egy természetes szám. Az egyes processzorok azonosítója tehát különbözik bármely más processzorétól. Az azonosító értékeknek nem kell egymás után következő $1, \ldots, n$ számoknak lenniük. Kezdetben egyetlen processzor sem ismeri egyetlen más processzor azonosítóját sem. A processzorok szintén nem ismerik a gyűrű n hosszát.

23.3.2. A vezetőválasztó algoritmus

A BULLY algoritmus megválaszt egy vezetőt a $P_1, \ldots, P_n$ processzorok közül. A processzorazonosítókat az algoritmus kritikus módon használja. Röviden: mindegyik processzor megpróbál vezető lenni, a legnagyobb azonosítójú processzor blokkolja a többiek kísérletét, magát vezetőnek deklarálja, és kényszeríti a többieket, hogy ne legyenek vezetők.

Az algoritmus néhány ötletének szemléltetéséhez kezdjük az algoritmus egy egyszerűbb változatával. Tegyük fel, hogy minden egyes processzor végigküld egy üzenetet a gyűrűben, mely tartalmazza a processzor azonosítóját. Bármely processzor *csak* akkor küld tovább egy ilyen üzenetet, ha az üzenetben lévő azonosító nagyobb, mint a processzor azonosítója. Így a gyűrűben a legnagyobb azonosítójú processzor által küldött üzenet mindig továbbítódik, végül körbeutazik a gyűrűn, és visszatér ahhoz a processzorhoz, mely eredetileg küldte. A processzor észlelni tudja, hogy egy ilyen üzenet visszaérkezett, mivel más processzor nem küld üzenetet ezzel az azonosítóval (az azonosítók egyediek). Vegyük észre, hogy más üzenet nem utazik körbe a gyűrűn, mivel a legnagyobb azonosítójú nem fogja továbbadni. Azt mondhatjuk, hogy a legnagyobb azonosítójú processzor „lenyeli” azokat az üzeneteket, melyek kisebb azonosítót szállítanak. Ezután a processzor vezető lesz, és végigküld egy speciális üzenetet a gyűrűn, amellyel arra veszi rá a többieket, hogy ne döntsenek úgy, hogy vezetők lesznek. Az algoritmus $\Theta(n^2)$ üzenetszámú, mivel az egyes processzorok legfeljebb n üzenetet küldenek, és a vezető küld még n további üzenetet, másrészt lehet úgy azonosítót rendelni a processzorokhoz és késleltetni a processzorokat és az üzeneteket, hogy az üzeneteket az n processzor egy konstans hányada küldi az n összeköttetés egy konstans hányadán. Az algoritmus javítható úgy, hogy az üzenetszám $O(n \lg n)$-re csökkenjen. Egy ilyen javított algoritmust mutatunk be ennek a pontnak a hátralévő részében.

A BULLY algoritmus kulcsgondolata az, hogy kevés üzenet utazik hosszan, így biztosítva az $O(n \lg n)$ üzenetszámot. Speciálisan, a processzorok tevékenységét fázisokra bontjuk. Egy fázis elején egy processzor küld egy „próba” üzenetet mind az ÓMJ, mind az ÓMJE irányba. Ezek az üzenetek a küldő azonosítóját tartalmazzák és egy bizonyos „élettartam” értéket, mely korlátozza, hogy hány lépést tehetnek az egyes üzenetek. A próba üzenetet akkor adhatja tovább egy processzor, ha a benne szereplő processzorazonosító nagyobb, mint a saját azonosítója. Amikor az üzenet eléri a korlátját, és nem nyelték le, akkor „visszapattan”. Így, ha az eredeti küldő két visszapattant üzenetet kap a két különböző irányból, akkor a processzor biztos benne, hogy a korlátnyi távolságra nincs nagyobb azonosítójú processzor sem az ÓMJ, sem az ÓMJE irányban, mivel akkor az lenyelte volna a próbaüzenetet. Csak ezután lép a processzor a következő fázisba, amiben egy nagyobb élettartam korlátú próbaüzenetet küld, hogy kitalálja, van-e nagyobb azonosítójú processzor egy kétszer nagyobb sugarú

környezetben. Ennek eredményeképpen egy processzor által küldött próbaüzenet csak akkor ugrik sokat, ha nincs nagyobb azonosítójú processzor a processzor egy nagy sugarú környezetében. Így tehát egyre kevesebb processzor küld üzeneteket, melyek egyre távolabb jutnak. Következésképp, ahogy nemsokára bebizonyítjuk, az algoritmus üzenetszáma $O(n \lg n)$ lesz.

A következőkben részletezzük a BULLY algoritmust. Minden egyes processzornak öt helyi változója van. Az *id* változó tárolja a processzor egyedi azonosítóját. A *vezető* változó IGAZ értéket vesz fel, amikor a processzor úgy dönt, hogy ő a vezető, egyébként a HAMIS értéket veszi fel. A maradék három változó az állapotkövetést szolgálja. Az *alszik* azt határozza meg, hogy a processzor küldött-e olyan <*próba,id,*0,0> üzenetet, mely a processzor *id*-jét hordozza. Bármely processzor küldhet <*próba,id,fázis,*$2^{fázis} - 1$> üzenetet bármely irányban (ÓMJ, ÓMJE) – különböző *fázis* különböző értékei mellett. Bármely üzenetre a processzor egy <*válasz,id,fázis*> üzenettel válaszolhat. Az *ÓMJválaszolt* és a *ÓMJEválaszolt* változók annak a nyilvántartására szolgálnak, hogy a választ feldolgozta-e a processzor.

Az egyes processzorokra vonatkozó kód öt részből áll. Az első rész (1–5. sorok) beállítja a processzor helyi változóinak kezdeti értékét. A második részt (6–8. sorok) csak akkor szabad végrehajtani, ha az *alszik* helyi változó értéke IGAZ. A fennmaradó három rész (9–17., 18–26. és 27–31. sorok) írja le azokat a lépéseket, melyeket a processzor a három különböző üzenet, nevezetesen a <*próba,ids,fázis,élettartam*>, <*válasz, ids, fázis*> és a <*megállás*> hatására végez. Az üzenetek az *ids fázis* és az *élettartam* paramétereket használják, melyek természetes számok.

Most bemutatjuk, hogyan működik az algoritmus. Emlékezzünk vissza, hogy az egyes processzorok helyi változói a globális idő szerinti 0 időpillanat előtt kaptak kezdőértéket. Minden egyes processzor valamikor a jövőben küld egy <*próba, id, 0, 0*> üzenetet, mely a processzor *id* azonosítóját tartalmazza. Ekkor azt mondjuk, hogy a processzor a 0 fázisba *lépett*. Általában, amikor a processzor egy <*próba,id,fázis,*$2^{fázis} - 1$> üzenetet küld, azt mondjuk, hogy a processzor a *fázis* sorszámú fázisba *lépett*. A <*próba, id, 0, 0*> üzenet sohasem kerül újraküldésre, mivel a 7. sorban az *alszik* változó a HAMIS értéket veszi fel. Előfordulhat, hogy mire ez az üzenet küldésre kerül, a processzor már más üzeneteket is feldolgozott.

Amikor egy processzor feldolgoz egy, az ÓMJ (vagyis az óramutató járásával megegyező irányhoz tartozó) kapcsolaton érkező <*próba, ids, fázis, élettartam*> formátumú üzenetet, akkor az *ids* paramétertől és a processzor *id* azonosítójától függően tevékenykedik. Ha az *ids* kisebb, mint az *id*, a processzor nem csinál semmit (azaz lenyeli az üzenetet). Ha az *ids* egyenlő az *id*-del, és a processzor még nem döntött, akkor, mint majd látjuk, a processzor által küldött *próba*-üzenet körbement a teljes gyűrűn. Ekkor a processzor küld egy <*megállás*> üzenetet, magát vezetőnek jelöli meg, és leáll (de a leállás után még dolgozhat fel üzeneteket). Ha az *ids* nagyobb, mint az *id*, akkor a tevékenység az *élettartam* paraméter értékétől függ. Ha ez az érték nagyobb, mint nulla, a processzor csökkenti az *élettartam* paraméter értékét, és így adja tovább az üzenetet. Ha az *élettartam* értéke nulla, akkor a processzor visszaküld (ÓMJ irányban) egy válaszüzenetet. Hasonlóak a műveletek, abban az értelemben, hogy az üzenetküldési irányok felcserélődnek, amikor a <*próba, ids, fázis, élettartam*> üzenet az ÓMJE kapcsolaton érkezik. A részleteket lásd a programkódban.

BULLY

Kód a P_k, $1 \leq k \leq n$ processzorok esetén:

```
1  kezdőértékek beállítása
2      alszik ← IGAZ
3      ÓMJválaszolt ← HAMIS
4      ÓMJEválaszolt ← HAMIS
5      vezető ← NIL

6  if alszik then
7      alszik ← HAMIS
8      <próba, id, 0, 0 > küldése az ÓMJ és az ÓMJE élekre

9  üzenet feldolgozása <próba, ids, fázis, élettartam> érkezett az ÓMJ (ill. ÓMJE) élre
10     if id = ids és vezető = NIL then
11         <megállás> küldése az ÓMJ élre
12         vezető ← IGAZ
13         terminate
14     if ids > id és élettartamttl > 0 then
15         <próba, ids, fázis, élettartam-1> küldése az ÓMJE (ill. ÓMJ) élre
16     if ids > id és élettartam = 0 then
17         <válasz, ids, fázis> küldése az ÓMJ (ill. ÓMJE) kapcsolatra

18 üzenet feldolgozása <válasz, ids, fázis> érkezett az ÓMJ (ill. ÓMJE) élen
19     if id ≠ ids then
20         <válasz, ids, fázis> küldése az ÓMJE (ill. ÓMJ) élre
21     else
22         ÓMJválaszolt ← IGAZ (ill. ÓMJEválaszolt)
23         if ÓMJválaszolt és ÓMJEválaszolt then
24             ÓMJválaszolt ← HAMIS
25             ÓMJEválaszolt ← HAMIS
26             <próba, id, fázis+1, 2^(fázis+1) − 1> küldése az ÓMJ és ÓMJE élekre

27 üzenet feldolgozása <megállás> érkezett az ÓMJ élen
28     if vezető = NIL then
29         <megállás> küldése az ÓMJE élre
30         vezető ← HAMIS
31         terminate
```

Amikor egy processzor az ÓMJ kapcsolaton érkező *<válasz, ids, fázis>* üzenetet kap, először ellenőrzi, hogy a *ids* értéke eltér-e a saját *id* értékétől. Ha igen, a processzor egyszerűen továbbadja az üzenetet. Ha azonban $ids = id$, akkor a processzor feljegyzi, hogy egy válaszüzenet érkezett az ÓMJ kapcsolaton úgy, hogy az *ÓMJválaszolt* változóhoz IGAZ értéket rendel. Ezután a processzor ellenőrzi, hogy az *ÓMJválaszolt* és az *ÓMJEválaszolt* változók értéke IGAZ-e. Ha igen, akkor már mindkét irányból érkezett válasz. Ekkor a processzor

mindkét változóhoz a HAMIS értéket rendeli, majd egy *próba* üzenetet küld. Ez az üzenet a processzor *id* azonosítójából, a következő fázis *fázis*+1 számából, és a $(2^{fázis+1} - 1)$-re növelt élettartamértékből áll. Hasonló tevékenységet kell végezni, ha a <*válasz, ids, fázis*> üzenet az ÓMJE kapcsolatra érkezik.

Az utolsó, a processzor által feldolgozandó üzenet a <*megállás*> . A processzor ellenőrzi, hogy már döntött-e arról, hogy ő-e a vezető. Ha még nem volt döntés, továbbadja a <*megállás*> üzenetet, és úgy dönt, hogy nem ő a vezető. Ez az üzenet egyszer majd egy olyan processzorhoz jut, mely már döntött, és így az nem kerül továbbadásra.

23.3.3. A vezetőválasztási algoritmus elemzése

Az elemzést azzal kezdjük, hogy belátjuk, a BULLY algoritmus megoldja a vezetőválasztási problémát.

Helyesség bizonyítása
Először a BULLY algoritmus helyességét látjuk be.

23.8. tétel. *A* BULLY *algoritmus megoldja a vezetőválasztási problémát bármely gyűrűben aszinkron processzorok esetén.*

Bizonyítás. Azt kell belátnunk, hogy az alfejezet elején említett két feltétel megáll. A bizonyítás kulcsa, mely egyszerűsíti az érvelést, hogy egy processzorra koncentrálunk. Tegyük fel, hogy a P_i processzor rendelkezik a gyűrűben a legnagyobb *id* azonosítóval. Ez a processzor valamikor feltétlenül végrehajtja a 6 – –8. lépéseket. Ekkor a processzor küld egy <*próba, id, 0, 0* > üzenetet az ÓMJ és az ÓMJE kapcsolatokon. Jegyezzük meg, hogy amikor a processzor egy <*próba, id, fázis,* $2^{fázis} - 1$> üzenetet küld, az ilyen üzeneteket a többi processzor mindig továbbadja, amíg az *élettartam* értéke nullára nem esik. Ekkor a processzor visszaküld egy választ a P_i-nek. Ekkor a P_i végső soron minden irányból megkapja a <*válasz, id, fázis*> üzenetet, és a *fázis*+1 fázisba lép úgy, hogy <*próba, id, fázis+1,* $2^{fázis+1} - 1$> üzenetet küld mindkét irányba. Ezek az üzenetek magasabb *élettartam* értékkel rendelkeznek, mint az előző, *fázis* sorszámú fázis élettartamértéke. Mivel a gyűrű véges, az *élettartam* olyan nagy lesz, hogy a P_i processzor kap egy olyan üzenetet, melyben a P_i az azonosító. Jegyezzük meg, hogy P_i két ilyen üzenetet is kap. Az első alkalommal, amikor a P_i egy ilyen üzenetet feldolgoz, a processzor küld egy <*megállás*> üzenetet, és leáll, mint vezető. A második alkalommal, amikor a P_i egy ilyen üzenetet feldolgoz, a 11–13. sorok nem kerülnek végrehajtásra, mivel a *vezető* változó értéke már nem NIL. Megjegyezzük, hogy más P_j processzor nem hajthatja végre a 11–13. sorokat, mert egy, a P_j-ból küldött *próba* üzenet nem mehet végig az egész gyűrűn, mivel az útjában lévő P_i lenyeli azt. Ugyanakkor mivel az azonosítók egyediek, más processzor nem küldhet *próba* üzenetet P_i azonosítóval. Így a P_i-n kívül más processzor nem rendelhet IGAZ értéket a *vezető* változójához. Bármely a P_i-től különböző processzor, mely a <*megállás*> üzenetet megkapja, HAMIS értéket rendel a *vezető* változójához, és továbbadja ez üzenetet. Végül, a <*megállás*> üzenet megérkezik P_i-hez, és az nem adja tovább. A fenti érvelés azt bizonyítja, hogy végül pontosan egy processzor rendeli a IGAZ értéket a *vezető* változójához, és az összes többi HAMIS értéket rendel ehhez a változójához. Ha egy processzor már értéket rendelt a *vezető* változójához, az már változatlan marad. ■

A következő feladatunk, hogy felső korlátot találjunk az algoritmus által küldött üzenetek számára. A következő lemma megmutatja, hogy az egy fázisba belépő processzorok számára a fázis sorszámának növekedésével exponenciálisan csökkenő korlát adható.

23.9. lemma. *Adott egy n csúcsú gyűrű. Az $i \geq 0$ fázisba belépő processzorok k száma legfeljebb $n/2^{i-1}$.*

Bizonyítás. Pontosan n processzor lép az $i = 0$ fázisba, mivel minden egyes processzor végső soron elküldi a <*próba, id, 0, 0*> üzenetet. A lemmában említett korlát azt állítja, hogy a 0 fázisba lépő processzorok száma legfeljebb $2n$, így a korlát triviálisan igaz $i = 0$-ra. Vizsgáljuk a többi esetet, azaz tegyük fel, hogy $i \geq 1$. Tegyük fel továbbá, hogy a P_j processzor belép az i-edik fázisba. Így az a definíció alapján küld egy <*próba*,$id, i, 2^i - 1$> üzenetet. Ahhoz, hogy a processzor egy ilyen üzenetet küldjön, a processzor által a megelőző fázisban a két irányba küldött két <*próba*, $id, i-1, 2^{i-1}-1$> üzenetnek 2^{i-1} ugrást kellett végrehajtania, mindig elérve egy, a P_j azonosítónál kisebb azonosítójú processzort. (Máskülönben ha egy *próba* üzenet egy, az üzenetben szereplőnél nagyobb vagy egyenlő azonosítójú processzorhoz érkezik, az lenyeli az üzenetet, és nem generál válaszüzenetet. Következésképp P_j nem léphetne az i-edik fázisba.) Ennek eredményeképp, ha egy processzor belép az i-edik fázisba, nincs más olyan processzor tőle mindkét irányban 2^{i-1} ugrásnyi távolságra, mely valaha is ebbe a fázisba lépne. Tegyük fel, hogy $k \geq 1$ processzor van az i-edik fázisban. Minden egyes ilyen p_j processzorhoz hozzárendelhetjük a tőle ÓMJ irányba lévő 2^{i-1} következő processzort. Ezért legalább $k + k \cdot 2^{i-1}$ különböző processzor van a gyűrűben. Így $k(1 + 2^{i-1}) \leq n$. Gyengíthetjük a korlátot az 1 elhagyásával, így a kívánt $k \cdot 2^{i-1} \leq n$ eredményt kapjuk. ■

23.10. tétel. *A* Bully *algoritmus üzenetszáma $O(n \lg n)$, ahol n a gyűrű mérete.*

Bizonyítás. Megjegyezzük, hogy bármely, az i-edik fázisban lévő processzor, a két (ÓMJ és ÓMJE) irányba, 2^i távolságra küld üzeneteket. Ez az i-edik fázisba lépő processzoronként legfeljebb $4 \cdot 2^i$ üzenetet jelent. Az üzenetek száma kisebb is lehet, mint $4 \cdot 2^i$, ha egy próbaüzenetet lenyelnek út közben. A 23.9. lemma felső korlátot ad a k-adik fázisba lépő processzorok darabszámára. Mi az a legmagasabb fázis, amelybe egy processzor még beléphet? Az i-edik fázisban lévő processzorok k száma legfeljebb $n/2^{i-1}$. Így, ha $n/2^{i-1} < 1$, nem lehet olyan processzor, mely az i-edik fázisba lép. Tehát egyik processzor sem léphet magasabb fázisba, mint $h = 1 + \lceil \lg n \rceil$, mivel $n < 2^{(h+1)-1}$. Végül, egyetlen processzor küld egy megállási üzenetet, mely egyszer körbemegy a gyűrűn. Így az algoritmus által küldött összes üzenetek számára a

$$n + \sum_{i=0}^{1+\lceil \lg n \rceil} \left(n/2^{i-1} \cdot 4 \cdot 2^i \right) = n + \sum_{i=0}^{1+\lceil \lg n \rceil} 8n = O(n \lg n)$$

felső korlátot kapjuk. ■

Burns azt is megmutatta, hogy a Bully algoritmus üzenetszáma aszimptotikusan optimális, mivel bármely, a processzorok számát nem ismerő vezetőválasztó algoritmus egy aszinkron gyűrűben $\Omega(n \lg n)$ üzenetet küld.

23.11. tétel. *Bármely, a processzorok számát nem ismerő vezetőválasztó algoritmus egy aszinkron gyűrűben* $\Omega(n \lg n)$ *üzenetet küld.*

Gyakorlatok

23.3-1. Bizonyítsuk be, hogy az egyszerűsített BULLY algoritmus üzenetszáma $\Omega(n^2)$, ha az n méretű gyűrűben megfelelően rendeljük az azonosítókat a processzorokhoz, és meghatározzuk, hogy hogyan késleltessük a processzorokat és az üzeneteket.

23.3-2. Mutassuk meg, hogy a BULLY algoritmus üzenetszáma $\Omega(n \lg n)$.

23.4. Hibatűrő egyetértés

Az eddig bemutatott algoritmusok azon a feltételezésen alapulnak, hogy megbízható rendszeren futnak. Most bemutatunk néhány kiválasztott algoritmust nem megbízható rendszerek esetén, ahol az aktív (vagy helyes) processzoroknak koordinálniuk kell a tevékenységüket, azaz közös döntéseket kell hozniuk.

A probléma természeténél fogva nehéz olyan processzoroknak megegyezni, melyek egy hibáknak kitett osztott rendszerben vannak. Tekintsük például azt a megtévesztően egyszerű példát, amikor két hibamentes processzor megpróbál megegyezni egy közös bitről, miközben olyan kommunikációs médiát használ, ahol az üzenetek elveszhetnek. Ez a probléma a ***két tábornok problémája*** néven ismert. Ebben két tábornoknak kell koordinálnia egy támadást olyan futárok segítségével, akiket elfoghat az ellenség. Az derül ki, hogy véges számú üzenetküldéssel nem sikerül megoldani a problémát. Ezt ellentmondásra jutással bizonyítjuk. Tegyük fel, hogy van olyan, az A és B processzorok között használt protokoll, mely véges számú üzenetküldésből áll. Tekintsük azt az ilyen protokollt, mely a legkevesebb – mondjuk, k darab – üzenetet használja. Az általánosság megszorítása nélkül feltételezhetjük, hogy az utolsó, k-adik üzenetet A küldi B-nek. Mivel ezt az utolsó üzenetet B nem nyugtázza, A-nak attól függetlenül kell meghoznia döntését, hogy B megkapta-e ezt az üzenetet. Mivel az üzenet elveszhet, B-nek az utolsó üzenettől függetlenül kell döntenie. Ez alapján azonban A és B a k-adik üzenet felhasználása nélkül döntött, azaz létezik egy olyan protokoll, mely csak $k - 1$ üzenetet használ a probléma megoldására. Ez ellentmond annak a feltételnek, hogy a probléma megoldásához k darab üzenetre van szükség.

Az alfejezet hátralévő részében azokat a problémákat vizsgáljuk, ahol a kommunikációs média megbízható, de a processzoroknál a következő kétfajta meghibásodás léphet fel: ***megállás*** (vagy összeomlás), amikor a processzor megáll, és nem hajt végre további lépéseket és a ***bizánci hiba,*** ahol a processzor a hibánál fogva tetszőleges (akár rosszindulatú) tevékenységet végezhet.

A bemutatott algoritmusok az úgynevezett ***egyetértési problémát*** (vagy konszenzus problémát) oldják meg, amely egy alapvető koordinációs probléma: azt kívánja a processzoroktól, hogy megegyezzenek egy közös kimenetben – annak ellenére, hogy a bemenetük különböző lehet.

23.4.1. Az egyetértési probléma

Tekintsünk egy olyan rendszert, amelyben minden egyes P_i processzor rendelkezik egy speciális x_i állapotkomponenssel, amit *bemenetnek* nevezünk, és rendelkezik egy y_i kom-

ponenssel, melyet *kimenetnek* (más néven *döntésnek*) nevezünk. Az x_i változó kezdetben a lehetséges bemenetek valamely jól rendezett halmazából származó értéket vesz fel, az y_i értéke nem definiált. Ha egyszer már rendeltek értéket az y_i változóhoz, az később már nem változtatható meg. Az egyetértési probléma bármely megoldásának a következőket kell garantálnia:

- **Megállás:** Minden megengedett végrehajtás és minden hibamentes P_i processzor esetén y_i kap értéket.
- **Megegyezés:** Minden végrehajtás és minden hibamentes P_i és P_j processzor esetén, ha y_i és y_j már kapott értéket, akkor $y_i = y_j$, azaz a hibamentes processzorok nem eredményeznek különböző kimeneteket.
- **Érvényesség:** Minden végrehajtásban, ha valamely v értékre $x_i = v$ minden P_i processzor esetén, azaz, ha minden processzor ugyanazt a bemenőértéket kapja, és y_i értéket kap valamely hibamentes P_i processzor esetén, akkor az eldöntött érték a közös bemenet lesz.

Megjegyezzük, hogy a megállási hibák esetén ez az érvényességi feltétel gyengébb annál a követelménynél, hogy a hiba nélkül működő processzorok valamely processzor bemenetét határozzák meg kimenetnek. Ha egy processzor megáll, az nem érdekli az algoritmust, ezért nincs követelmény a meghibásodott processzor kimenetére sem.

Egy egyszerű algoritmussal kezdjük egy megállási hibát megengedő szinkron üzenetküldő rendszerben.

23.4.2. Egyetértés megállási hibák esetén

Mivel a rendszer szinkron, a végrehajtás menetek sorozatából áll. Minden egyes menet az összes üzenet kézbesítéséből, és az azt követő egyetlen számítási lépésből áll az összes processzorra. A meghibásodott processzorok a különböző végrehajtásokban különbözők lehetnek, azaz előre nem ismertek, számuk legfeljebb f. Legyen F a hibás processzorok halmaza. Minden egyes menet pontosan egyetlen számítási lépést tartalmaz azon processzorok esetén, melyek nincsenek F-ben, és legfeljebb egy lépést az F-ben lévő processzorok esetén. Továbbá, ha egy processzor F-ben van, és valamely menetben nem hajt végre számítási lépést, a további menetekben már nem hajt végre ilyen lépést. Az utolsó olyan menetben, melyben egy meghibásodott processzor számítási lépést hajt végre, a kimenő üzeneteinek tetszőleges részhalmaza kerül kézbesítésre.

EGYETÉRTÉS-MEGÁLLÁSI-HIBÁKNÁL

Kód a P_i, $0 \leq i \leq n-1$ processzorok esetén.
Kezdetben $V = \{x\}$
k-adik menet, $1 \leq k \leq f+1$
1 $\{v \in V : p_i$ még nem küldte el v-t$\}$ küldése az összes processzorhoz
2 S_j fogadása P_j-től, $0 \leq j \leq n-1$, $j \neq i$
3 $V \leftarrow V \cup \bigcup_{j=0}^{n-1} S_j$
4 **if** $k = f+1$ **then** $y \leftarrow min(V)$

Ebben az algoritmusban miden egyes processzor fenntart egy halmazt a rendszerben ismert értékek tárolására. Ez kezdetben csak a saját bemenetet tartalmazza. A későbbi menetekben a processzor ezt frissíti a már processzoroktól kapott halmazokkal. Ezután az új megismert étékeket közvetíti az összes processzor felé. Ez $f+1$ menetig tart, ahol f a meghibásodható processzorok számának maximuma. Ezen a ponton a processzor meghatározza, hogy melyik a legkisebb érték az értékhalmazában.

Az algoritmus helyességének bizonyításához először azt kell észrevenni, hogy az algoritmus pontosan $f + 1$ menetet igényel. Ez maga után vonja a megállást. Továbbá, az érvényességi feltétel nyilvánvalóan megáll, mivel a meghatározott érték valamely processzor bemenő értéke. Már csak azt kell megmutatni, hogy a megegyezési feltétel teljesül. A következő lemmát fogjuk bizonyítani:

23.12. lemma. *Minden egyes végrehajtásban az $(f + 1)$-edik menet végén $V_i = V_j$, minden hibamentes P_i és P_j processzorpárra.*

Bizonyítás. Az állítást azzal bizonyítjuk, hogy megmutatjuk: ha $x \in V_i$ az $(f+1)$-edik menet végén, akkor $x \in V_j$ is az $(f + 1)$-edik menet végén.

Legyen r az első olyan menet, amelyben x-et V_i-hez adták valamely p_i hibamentes processzor esetében. Ha x kezdetben V_i-ben van, legyen $r = 0$. Ha $r \leq f$, akkor az $r + 1 \leq f + 1$ menetben P_i elküldi x-et az összes P_j-nek. Ezáltal P_j x-et V_j-hez adja, ha már nem lenne benne.

Egyébként tegyük fel, hogy $r = f + 1$, és legyen P_j egy hibamentes processzor, mely x-et először az $f + 1$-edik menetben kapja meg. Ekkor kell lenni egy olyan $f + 1$ elemből álló $P_{i_1}, \ldots, P_{i_{f+1}}$ processzorláncnak, melyből az x érték P_j-hez érkezik. Így P_{i_1} elküldi x-et P_{i_2}-hez az első menetben stb., amíg $P_{i_{f+1}}$ el nem küldi x-et P_j-hez az $(f + 1)$-edik menetben. Ugyanakkor $P_{i_1}, \ldots, P_{i_{f+1}}$, egy $f + 1$ processzorból álló lánc. Így legalább az egyik processzor a láncból, mondjuk P_{i_k}, hibamentes. Ezáltal P_{i_k} felveszi x-et a $k - 1 < r$ menetben, ami ellentmond annak, hogy r minimális. ■

Ez a lemma a korábban említett megfigyeléssel együtt bizonyítja a következő tételt.

23.13. tétel. *Az* EGYETÉRTÉS-MEGÁLLÁSI-HIBÁKNÁL *algoritmus megoldja az egyetértési problémát legfeljebb f megállási hiba esetén egy $f + 1$ menetből álló üzenetküldő rendszerben.*

A következő tétel megmutatja, hogy az előző algoritmus az adott modell mellett optimális.

23.14. tétel. *Nincs olyan algoritmus, mely az egyetértés problémát kevesebb, mint $f + 1$ menetben, f megállási meghibásodás mellett, ha $n \geq f + 2$.*

Mi van, ha meghibásodások nem jóindulatúak? Azaz, megoldható-e az egyetértési probléma *bizánci* típusú meghibásodások mellett? Ha igen, hogyan?

23.4.3. Egyetértés bizánci típusú meghibásodások mellett

A bizánci modellben egy hibás processzor egy számítási lépés után ismeretlen állapotba kerül, és az általa küldött üzenet tetszőleges. Mint a megbízható esetben, minden egyes processzor egy számítási lépést tesz minden menetben, és minden, általa küldött üzenet kézbesítésre kerül az adott menetben. Így a meghibásodott processzor tetszőlegesen, sőt

rosszindulatúan viselkedhet. Például, különböző üzeneteket küldhet különböző processzorokhoz. Még az is előfordulhat, hogy a meghibásodott processzorok együttműködnek. Egy hibás processzor még egy összeomlott processzor működését is utánozhatja azzal, hogy egy adott pont után nem küld üzeneteket.

Ebben az esetben a konszenzus probléma definíciója ugyanaz, mint az összeomlásos hibák melletti üzenetküldési modellben. Az érvényességi feltétel ebben a modellben azonban nem ekvivalens annak megkövetelésével, hogy a hibamentes processzorok kimeneti értéke megfelel valamely processzor bemenetének. Mint a megállásos esetben, nincs feltétel a meghibásodott processzorok kimenetére.

23.4.4. Alsó korlát a hibás processzorok arányára

Pease, Shostak és Lamport bizonyította először a következő tételt:

23.15. tétel. *Ha $n \leq 3f$, akkor nincs olyan algoritmus, amely egy n processzorból és f bizánci processzorból álló rendszerben megoldja az egyetértési problémát.*

23.4.5. Egy polinomiális algoritmus

A következő algoritmus konstans méretű üzeneteket használ, $2(f+1)$ menetet igényel, és feltételezi, hogy $n > 4f$. Az algoritmust Berman és Garay mutatta be.

Ez a bizánci típusú meghibásodások melletti egyetértési algoritmus $f+1$ fázisból áll, minden fázis két menetet tartalmaz. Minden egyes processzor minden egyes fázisban rendelkezik egy kedvezményezett döntéssel, mely kezdetben a bemenő értéke. Minden egyes fázis első menetében a processzorok elküldik ezt a kedvezményezett adatot egymásnak. Legyen v_i^k a leggyakoribb érték, melyet a P_i processzor kapott a k fázis első menetében. Ha nincs ilyen érték, a $v_\perp$ érték lesz a használatos. A fázis második menetében a menet *királyának* nevezett P_k processzor elküldi a v_k^k többségi értékét az összes processzornak. Ha P_i (a fázis első menetében) $(n/2+f)$-nél több példányt kap v_i^k-ból, akkor a következő fázisban v_i^k lesz a kedvezményezett értéke. Egyébként a kedvezményezett értéke a fázis királyának v_k^k kedvezményezettje lesz, melyet a fázis második menetében kapott. $f+1$ fázis után a processzor dönt a kedvezményezettje felől. Minden processzor fenntart egy $pref[0\,..\,n-1]$ helyi tömböt.

A helyességet a következő lemmákkal bizonyítjuk. A megállás közvetlen. A következőkben a döntés állandóságát látjuk be.

23.16. lemma. *Ha a k-adik fázis elején az összes hibamentes processzor kedvezményezettje v, akkor a k-adik fázis végén is v lesz a kedvezményezett érték minden processzor és minden k, $1 \leq k \leq f+1$ esetén.*

Bizonyítás. Mivel az összes hibamentes processzor kedvezményezettje a k fázis elején v, ezért mind legalább $n-f$ példányt kapnak (a sajátjukkal együtt) v-ből a k fázis első menetében. Mivel $n > 4f$ és $n - f > n/2 + f$, ezért az összes hibamentes processzor kedvezményezettje v lesz a k-adik fázis végén. ■

EGYETÉRTÉS-BIZÁNCI-HIBÁKNÁL

Kód a P_i, $0 \leq i \leq n-1$ processzorok esetén:
Kezdetben $pref[j] = v_\perp$, minden $j \neq i$ esetén
$2k-1$, $1 \leq k \leq f+1$ menet
1 $\langle pref[i] \rangle$ küldése az összes processzornak
2 $\langle v_j \rangle$ fogadása P_j-től, és hozzárendelés $pref[j]$-hez minden $0 \leq j \leq n-1$, $j \neq i$ esetén
3 legyen *maj* a többségben lévő érték a $pref[0], \ldots, pref[n-1]$ tömbelemekben,
illetve $v_\perp$, ha nincs ilyen
4 legyen *mult maj* multiplicitása

$2k$, $1 \leq k \leq f+1$ menet
5 **if** $i = k$ **then** $\langle maj \rangle$ küldése az összes processzorhoz
6 $\langle$*király-maj*$\rangle$ fogadása p_k-tól, ($v_\perp$ ha nincs)
7 **if** $mult > n/2 + f$
8 **then** $pref[i] \leftarrow maj$
9 **else** $pref[i] \leftarrow$ *király-maj*
10 **if** $k = f+1$ **then** $y \leftarrow pref[i]$

Mindez maga után vonja az érvényességi feltétel teljesülését: Ha az összes processzor a v bemenettel indul, akkor a továbbiakban is v lesz a kedvezményezettjük, és az $(f+1)$-edik fázisban a v lesz a eldöntött érték. A megegyezés meglétét a király biztosítja. Mivel minden fázisban más a király és $f+1$ fázis van, legalább egy menet rendelkezik hibamentes királlyal.

23.17. lemma. *Legyen g egy olyan fázis, amelyben a P_g király hibamentes. Ekkor az összes hibamentes processzor befejezi a g fázist, és a kedvezményezettje ugyanaz lesz.*

Bizonyítás. Tegyük fel, hogy az összes hibamentes processzor a királytól kapott többségi értéket veszi kedvezményezettnek. Mivel a király hibamentes, ugyanazt az üzenetet küldi, így az összes hibamentes processzornál a kedvezményezett ugyanaz.

Tegyük fel, hogy egy P_i hibamentes processzor a saját v többségi értékét tekinti kedvezményezettnek. Így P_i a g fázis első menetében v-re $(n/2 + f)$-nél több üzenetet kap. Ezáltal az összes processzor, P_g-t is beleértve, a g fázis első menetében több mint $n/2$ üzenetet kap v-re, és a többségi értékét v-re állítja. Így minden hibamentes processzor v-t választja kedvezményezettnek. ■

Ezáltal a $(g+1)$-edik fázisban az összes processzor ugyanazzal a kedvezményezettel rendelkezik, és a 23.16. lemma alapján ugyanazt az értéket határozzák meg kedvezményezettnek. Tehát az algoritmus rendelkezik a megegyezés tulajdonsággal, és megoldja az egyetértési problémát.

23.18. tétel. *Ha $n > 4f$, akkor létezik olyan algoritmus, mely n processzor és f bizánci típusú meghibásodás esetén megoldja az egyetértési problémát $2(f+1)$ menetben konstans méretű üzenetekkel.*

23.4.6. Lehetetlenség az aszinkron rendszerekben

Mint ahogy korábban bemutattuk, az egyetértési probléma megoldható szinkron rendszerekben mind megállás (jóindulatú), mind bizánci (súlyos) típusú meghibásodások mellett. Mi a helyzet az aszinkron rendszerekkel? Azon feltételezés mellett, hogy a kommunikációs rendszer teljesen megbízható, és a meghibásodásokat csak a megbízhatatlan processzorok okozzák, belátható, hogy ha a rendszer teljesen aszinkron, nincs egyetértési algoritmus még akkor sem, ha csak egyetlen processzor hibásodhat meg. Ez akkor is igaz, ha a processzorok csak megállás típusú hibának vannak kitéve. A lehetetlenség bizonyítása főleg a rendszer aszinkron voltán nyugszik.

A lehetetlenség fennáll mind a csak az olvasás/írás regisztereket használó közös memóriájú, mind az üzenetküldő rendszerekre. Az állítás először a közös memóriájú rendszerekre vonatkozik. Az üzenetküldő rendszerekre az eredmény szimulációval vihető át.

23.19. tétel. *Nincs olyan egyetértési algoritmus egy olvasás/írás aszinkron közös memóriájú rendszerre, mely akár egyetlen megállás típusú meghibásodást tolerálni tudna.*

Szimulációval a következő is belátható:

23.20. tétel. *Nincs algoritmus, mely egy n processzorból álló aszinkron üzenetküldő rendszerben megoldja az egyetértés problémát, ha akár csak egy processzor is megállási hibás.*

Megjegyezzük, hogy ezek az eredmények nem jelentik azt, hogy az egyetértés megoldhatatlan az aszinkron rendszerek esetén. Az eredmények inkább azt jelentik, hogy nincs olyan algoritmus, mely garantálná a megállást, a megegyezést és az érvényességet az összes végrehajtásban. Ésszerű azt feltételezni, hogy a megegyezés és az érvényesség szükséges, azaz ha egy konszenzus algoritmus megáll, akkor a megegyezési és az érvényességi követelmények garantáltak. Valójában vannak olyan hatékony és hasznos algoritmusok az egyetértés problémára, melyek nem garantálják a megállást minden végrehajtás esetén. A gyakorlatban ez elégséges lehet, mivel a nem-megállást okozó speciális feltételek meglehetősen ritkák. Továbbá, mivel sok valóságos rendszerben az ember időzítési feltételekkel élhet, lehet, hogy nem szükséges megoldást adni az aszinkron egyetértésre.

qvspace-1mm

Gyakorlatok

23.4-1. Bizonyítsuk be a Egyetértés-megállási-hibáknál algoritmus helyességét.
23.4-2. Bizonyítsuk be a Egyetértés-bizánci-hibáknál algoritmus helyességét.
23.4-3. Bizonyítsuk be a 23.20. tételt.

23.5. Logikai idő, okság és konzisztens állapot

Egy osztott rendszerben gyakran hasznos megállapítani az összes processzorok állapotaiból álló globális állapotot. Ha hozzáférünk a globális állapothoz, megállapításokat tehetünk az összes processzortól függő rendszertulajdonságokról, például észlelhetünk egy holtpontot. Az egyik lehetőség a globális állapot meghatározására az összes processzor megállítása, és az állapotainak összegyűjtése egy központi helyre. Egy ilyen módszer megfelel a legtöbb olyan osztott rendszerben, mely örökké számol. Ez az alfejezet arról szól, hogyan lehet

megállapítani a globális állapotot, ami meglehetősen intuitív, ugyanakkor konzisztens egy pontos értelemben.

Először egy olyan osztott algoritmust vizsgálunk, mely processzorok által végrehajtott utasítások egy globális sorrendjét határozza meg. Ez az algoritmus egy olyan illúziót kelt, mintha a processzorok számára egy globális óra állna rendelkezésre. Ezután bevezetjük a más utasításra hatással lévő utasítás fogalmát, és egy olyan algoritmust, mely kiszámítja, hogy mely utasítás van hatással mely más utasításra. Erről a fogalomról kiderül, hogy nagyon hatásos egy osztott rendszer konzisztens globális állapotának maghatározásában. Az alfejezetet olyan osztott algoritmusokkal zártjuk, melyek egy osztott rendszer egy konzisztens globális állapotát határozzák meg.

23.5.1. Logikai idő

Osztott algoritmusok tervezése könnyebb, ha a processzorok hozzáférnek egy (newtoni) globális órához, mivel az osztott rendszerben előforduló események az óra állásával megcímezhetők, a processzorok megegyezhetnek bármely események sorrendjéről, és ezt az egyetértést az algoritmusok felhasználják döntések meghozatalára. Ugyanakkor egy globális óra elkészítése nehéz. Vannak algoritmusok, melyek közelítik az ideális globális órát helyi hardverórák periodikus szinkronizálásával. Ugyanakkor lehetséges az eseményeket hardver órák használata nélkül teljesen rendezni. Ezt az fogalmat ***logikai órának*** nevezik.

Emlékezzünk, vissza, hogy egy végrehajtás n program utasításainak egymásba fonása. Minden egyes utasítás egy processzor egy számítási lépése, üzenetküldése vagy üzenetfogadása lehet. Bármely utasítást a globális idő egy meghatározott pontján hajtanak végre. Ugyanakkor a globális óra olvasása nem lehetséges a processzorok számára. A célunk az, hogy a logikai óra állását rendeljük az egyes utasításokhoz úgy, hogy ezek az értékek a globális óra értékeinek tűnjenek. Azaz az utasítások végrehajtásának pillanatait előre vagy hátra mozgathatjuk úgy, hogy minden egyes x utasítás, melyhez a lokális óra t_x időt rendeli, pontosan a globális óra t_x időpillanatában hajtódik végre, és a létrejövő végrehajtás érvényes abban az értelemben, hogy ténylegesen előfordulhat, amikor az algoritmust késleltetve futtatják.

A LOGIKAI-ÓRA nevű algoritmus logikai időt rendel minden egyes utasításhoz. Az egyes processzorok rendelkeznek egy *számláló*-nak nevezett helyi változóval. Ennek a változónak az értéke kezdetben nulla, és a processzor minden utasításvégrehajtása után növekszik. Amikor egy processzor üzenetküldéstől és üzenetfogadástól eltérő utasítást hajt végre, a *számláló* értéke pontosan eggyel növekszik. Amikor egy processzor üzenetet küld, a változót eggyel növeli, és az eredményt csatolja az üzenethez. Amikor egy processzor üzenetet fogad, akkor az üzenethez csatolt értéket beolvassa, meghatározza a *számláló* aktuális értékének és a kapott értéknek a maximumát, növeli ezt a maximumértéket eggyel, és hozzárendeli a *számláló* változóhoz. Megjegyezzük, hogy minden utasításvégrehajtáskor a *számláló* változó értéke legalább eggyel növekszik, és tovább nő, amíg a processzor utasításokat hajt végre. Az x utasításhoz rendelt logikai időt a *(számláló,id)* pár határozza meg, ahol a *számláló* a *számláló* változó értéke közvetlenül az utasítás végrehajtása után, az *id* pedig a processzor azonosítója. A logikai idő értékei egy teljes rendezést alkotnak, ahol a párokat lexikografikusan hasonlítjuk össze. Ezt a logikai időt Lamport-időnek is nevezik. t_x-et a *számláló* $+1/(id+1)$ hányadosként határozzuk meg, ami a pár reprezentálásának egy ekvivalens módja.

23.21. állítás. *Bármely végrehajtás esetén a logikai idő kielégíti az alábbi három feltételt:*

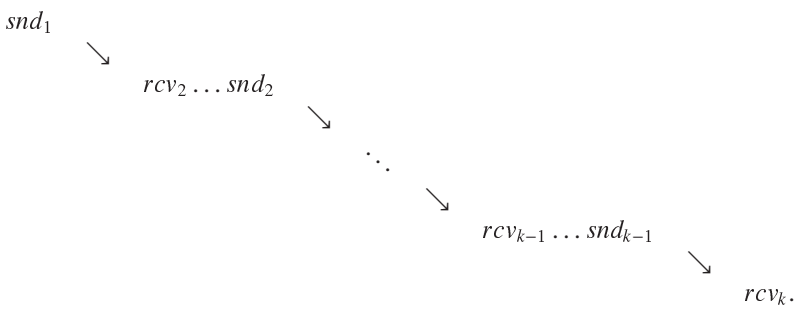
1. *ha egy x utasítást egy processzor egy y utasítás előtt hajt végre, akkor x logikai ideje kisebb, mint y-é.*
2. *bármely két processzor bármely két különböző utasításához különböző logikai időt kell rendelni.*
3. *Ha egy x utasítás egy üzenetet küld, és y fogadja ezt az üzenetet, akkor x logikai ideje kisebb, mint y-é.*

Most az a célunk, hogy belássuk, hogy a logikai óra a processzorok számára a globális óra illúzóját kelti. Intuitív módon egy ilyen illúzió létrehozhatóságának az oka, hogy vehetjük egy determinisztikus algoritmus bármely végrehajtását, kiszámolhatjuk a t_x logikai időt az összes x utasítás esetén, majd újrafuttathatjuk a végrehajtást a processzorok és üzenetek olyan lassításával, illetve felgyorsításával, hogy az x utasítások a globális óra t_x időpillanatában kerülnek végrehajtásra. Így a hardverórához vagy más, a modellünkben nem ismertetett, külső mérőeszközhöz való hozzáférés nélkül a processzorok nem tudják megkülönböztetni a logikai és a globális órák által mutatott időt. Hogy miért érvényes formálisan az újraidőzített végrehajtás, vagyis miért megkülönböztethetetlen az eredeti végrehajtástól, azt az alábbi következményben összegezzük, mely a 23.21. állításból közvetlenül következik.

23.22. következmény. *Bármely α végrehajtás esetén legyen T az utasításokhoz történő logikai idő hozzárendelése, és legyen β az α-beli utasítások logikai idő szerint rendezett sorozata. Ekkor minden processzor esetén, a processzor által α-ban végrehajtott utasítások részsorozata azonos a β-beli részsorozattal. Továbbá, minden egyes, β-ban fogadott üzenet a β-beli elküldés után kerül fogadásra.*

23.5.2. Okság

Egy rendszervégrehajtásban egy utasítás hatással lehet egy másik utasításra úgy, hogy megváltoztatja annak a számításnak az állapotát, melyet a másik utasítás végrehajt. Azt mondjuk, hogy egy utasítás ***oksági hatással*** (más néven befolyással) van egy másikra, ha az első utasítás által létrehozott információ átadható a másik utasításának. Emlékezzünk vissza, hogy az osztott rendszerünk modelljében minden utasítást a globális idő egy jól meghatározott pontján hajtunk végre, de a processzorok nem férnek hozzá egy globális órához. Illusztráljuk az okságot. Ha két utasítást ugyanaz a processzor hajt végre, akkor azt mondhatjuk, hogy a korábban végrehajtott utasítás okozati módon befolyásolja a később végrehajtott utasítást, mivel lehetséges, hogy a korábban végrehajtott utasítás eredményét a később végrehajtott utasítást felhasználja. A lehetséges szót hangsúlyoznunk kell, mivel valójában a későbbi utasítás lehet, hogy nem használja a korábbi által előállított információt. Azonban az okság meghatározásánál egyszerűsítjük azt a problémát, hogy hogyan befolyásolnak az egyes processzorok más processzorokat, és csak arra koncentrálunk, hogy mi lehetséges. Ha az x és y utasítást különböző processzorokon hajtjuk végre, azt mondhatjuk, hogy az x utasítás oksági hatással van az y utasításra, amikor az x-et végrehajtó processzor x végrehajtása közben vagy után üzenetet küld, és az üzenet megérkezett y másik processzoron történő végrehajtása előtt vagy közben. Az is lehetséges, hogy a befolyást más processzorok közvetítik vagy a processzorok több utasítást hajtanak végre a másik processzor elérése előtt.



23.1. ábra. Utasítás- és üzenetláncolat.

Azt az intuíciót, hogy egy utasítás oksági hatással van egy másikra, formálisan az utasításpárokhoz kapcsolódó ***korábban történt*** reláció segítségével határozzuk meg. A relációt egy adott végrehajtásra határozzuk meg, vagyis az algoritmus által végrehajtott utasítások sorozatát és az utasítások végrehajtása alatti globális idő állásait rögzítjük, majd meghatározzuk, hogy mely utasításpárok elégítik ki a *korábban történt* relációt. A relációt két lépesben vezetjük be. Ha az x és y utasításokat ugyanaz a processzor hajtja végre, úgy pontosan akkor mondjuk azt, hogy x ***korábban történt*** y-nál, ha x y előtt hajtódik végre. Ha x és y különböző processzorokon hajtódik végre, úgy pontosan akkor mondjuk azt, hogy x *korábban történt* y-nál, ha létezik olyan utasítás- és üzenetláncolat $k \geq 2$ esetén, hogy snd_1 vagy egyenlő x-szel vagy ugyanazon processzoron x után hajtódik végre; rcv_k vagy egyenlő y-nal vagy y előtt hajtódik végre ugyanazon a processzoron, mint y, rcv_h snd_h előtt hajtódik végre ugyanazon a processzoron, $2 \leq h < k$, és snd_h egy olyan üzenetet küld, melyet rcv_{h+1} fogad, $1 \leq h < k$ esetén (lásd a 23.1 alfejezetet).

Megjegyezzük, hogy egyetlen utasítás sem hajtódik végre önmaga előtt. Azt, hogy x y előtt történik, úgy jelöljük, hogy $x <_{HB} y$. Arra a végrehajtásra vonatkozó hivatkozást, melyben a reláció definiálva van, kihagyjuk, mivel a környezetből egyértelmű, hogy mely végrehajtást tekintjük. Azt mondjuk, hogy az x és y utasítások ***konkurensek,*** ha sem $x <_{HB} y$, sem pedig $y <_{HB} x$.

Felmerül a kérdés, hogy a processzorok a meghatározás alapján egy adott végrehajtás mellett hogyan tudják eldönteni, hogy egy utasítás egy másik előtt történik. Ez a korábban bemutatott LOGIKAI-ÓRA algoritmus általánosításával válaszolhatjuk meg. Az általánosítást VEKTORÓRA-nak nevezzük.

A VEKTORÓRA algoritmus lehetővé teszi a processzorok számára, hogy utasításokat kössenek össze, és ez az összekötés pontosan a *korábban történik* relációt adja. Minden P_i processzor fenntart egy n egészből álló V_i vektort. A vektor j-edik koordinátáját $V_i[j]$-vel jelöljük. A vektort kezdeti értéke a $(0, \ldots, 0)$ nullvektor. A vektor mindig módosul, ha egy processzor egy utasítást hajt végre úgy, ahogy a *számláló* módosult a LOGIKAI-ÓRA algoritmusban. Speciálisan, amikor egy P_i processzor végrehajt egy üzenetküldéstől vagy egy üzenetfogadástól eltérő utasítást, a $V_i[j]$ érték eggyel növekszik, míg a más koordinátán lévő értékek változatlanul maradnak. Amikor a processzor üzenetet küld, $V_i[j]$ értékét növeli eggyel, és az eredményül létrejövő V_i vektort csatolja az üzenethez. Amikor a P_j processzor üzenetet kap, beolvassa a csatolt V vektort, és koordinántánként kiszámítja a V és a

V_j maximumát a $V_j[j]$ koordináta kivételével, mely utóbbi értékét növeli eggyel. Ezután a keletkező eredményt hozzárendeli a V_j vektorhoz.

$V_j[j] \leftarrow V_j[j] + 1$
minden $k \in [n] \setminus \{j\}$ értékre
  $V_j[k] \leftarrow \max\{V_j[k], V[k]\}$

A P_i processzor által végrehajtott minden x utasítást a V_i vektor utasítás végrehajtása utáni értékével megcímkézzük. A címkét $VT(x)$-vel jelöljük, és *vektor időbélyegnek* nevezzük. Intuitív módon $VT(x)$ a P_i processzor ismereteit reprezentálja arról, hogy az egyes processzorok hány utasítást hajtottak végre abban a pillanatban, amikor P_i az x utasítást hajtotta végre. Ez a tudás elavult lehet.

A vektor-időbélyegek felhasználhatók a végrehajtott utasítások sorrendbe rendezéséhez. Specifikusan, ha adott két utasítás x és y, és a megfelelő vektor-időbélyegük $VT(x)$ és $VT(y)$, azt írjuk, hogy $x \leq_{VT} y$, amikor a $VT(x)$-t majorálja a $VT(y)$ vektor, azaz minden k-ra a $VT(x)[k]$ koordináta értéke legfeljebb a megfelelő $VT(y)[k]$ értéke. Ha $x \leq_{VT} y$, de $VT(x) \neq VT(y)$, akkor azt írjuk, hogy $x <_{VT} y$.

A következő tétel megmutatja, hogy a Vektoróra algoritmus valóban megvalósítja a *korábban történt* relációt, mivel két utasításról eldönthetjük, hogy egyik a másik előtt hajtódott-e végre vagy sem, egyszerűen úgy, hogy összehasonlítjuk az utasítások időbélyegeit.

23.23. tétel. *Bármely végrehajtás és bármely két x és y utasítás esetén $x <_{HB} y$ akkor és csak akkor, ha $x <_{VT} y$.*

Bizonyítás. Először az előreirányuló következményt látjuk be. Tegyük fel, hogy $x <_{HB} y$. Így x és y két különböző utasítás. Ha a két utasítást ugyanazon a processzoron hajtják végre, akkor x-et y előtt kell, hogy végrehajtsák. y végrehajtásának idejére csak véges számú utasítást hajtottak végre. A Vektoróra algoritmus eggyel növel egy koordinátát és kiszámolja az x és y közötti utasítások vektor-időbélyegeit az x-hez és y-hoz tartozót is beleértve, és egyetlenegy koordinátaérték sem kerül csökkentésre. Emiatt $x <_{VT} y$. Ha az x és y utasítást különböző processzorokon hajtják végre, akkor a *korábban történik* reláció meghatározása szerint létezik egy x-ből y-ba vezető utasítás- és üzenetlánc. De a vektoróra algoritmus szerint egy vektor-időbélyeg koordinátájának értéke minden lépésben növekszik, ahogy a láncban előre haladunk, és így szintén $x <_{VT} y$.

Most belátjuk a visszafelé irányuló következményt is. Tegyük fel, hogy nem igaz, hogy $x <_{HB} y$. Tekintsük azt a néhány alesetet, amelynél nem mindig igaz a $x <_{VT} y$ konklúzió. Először, előfordulhat, hogy x és y ugyanazon utasítás. De ekkor az x-hez és az y-hoz rendelt vektorórák is nyilvánvalóan azonosak, tehát nem igaz az, hogy $x <_{VT} y$. Tegyük fel tehát, hogy x és y különböző utasítások. Ha ugyanazon a processzoron hajtják végre őket, akkor x-et nem lehet y előtt végrehajtani, így x y után kerül végrehajtásra. Így, az időbélyegek monotonitása miatt $y <_{VT} x$, és nem igaz, hogy $x <_{VT} y$. Az utolsó aleset az, amikor x-et és y-t két különböző processzor hajtja végre – P_i és P_j. Koncentráljunk a P_i processzor V_i vektorórájának i-edik komponensére közvetlenül az x végrehajtása után. Legyen ez az érték k. Emlékezzünk vissza, hogy a processzorok az i-edik komponensük értékét csak úgy tudják növelni, ha átvesznek egy más processzor által küldött értéket. Így ahhoz, hogy a P_j pro-

cesszor i-edik komponensének értéke k vagy több legyen, y végrehajtásának pillanatában léteznie kell egy olyan, P_i-ből induló, utasításokból és üzenetekből álló láncnak, mely legalább k nagyságú értéket küld. Ez a lánc az x vagy a P_i által x után közvetlenül végrehajtott utasítással kezdődik. De egy ilyen lánc létezése maga után vonja, hogy x y előtt történik, amiről feltettük, hogy nem igaz. Így a $VT(y)$ vektoróra i-edik komponense kisebb, mint a $VT(x)$ vektoróra i-edik komponense. Ezért nem lehet igaz az, hogy $x <_{VT} y$. ■

A tétel szerint úgy dönthetjük el, hogy a különböző x és y utasítások konkurensek-e, hogy ellenőrizzük, hogy $VT(x) < VT(y)$ és $VT(x) > VT(y)$ igaz-e.

23.5.3. Konzisztens állapot

A korábban történik reláció felhasználható egy osztott rendszer egy globális állapotának megállapítására úgy, hogy ez az állapot valamilyen értelemben konzisztens. Rövidesen formálisan meg fogjuk határozni a konzisztencia fogalmát. Minden processzor utasításokat hajt végre. A K ***vágatot*** úgy definiáljuk, mint a nemnegatív egészekből álló cut $K = (k_1, \ldots, k_n)$ vektort. Intuitív módon, a K a processzorok állapotait jelöli. Formálisan k_i azon utasítások számát tartalmazza, melyeket a P_i processzor végrehajtott. Nem minden vágat felel meg osztott processzorok természetesnek vagy konzisztensnek tekintett állapotgyűjteményének. Például, ha a P_i processzor üzenetet kapott P_j-től, és P_i állapotát rögzítjük a vágatban azzal, hogy k_i-t megfelelően nagyra választjuk az üzenet fogadása után, de k_j-t olyan kicsire választjuk, hogy a vágat a küldő azon állapotát tartalmazza, mielőtt az üzenetet elküldte, akkor azt mondhatjuk, hogy az ilyen vágat nem természetes: olyan utasítások vannak a vágatban rögzítve, melyekre a vágatban nem rögzített utasítások oksági hatással vannak. Az ilyen vágatokat nem tekintjük konzisztensnek, s így nem kívánatosak. Formálisan egy $K = (k_1, \ldots, k_n)$ vágat ***inkonzisztens,*** ha léteznek olyan P_i és P_j processzorok, hogy a P_i k_i számú utasítására oksági hatással van a P_j processzor egy k_j utáni utasítása. Így egy inkonzisztens vágatban van olyan üzenet, mely visszafelé „szeli át” a vágatot. A nem inkonzisztens vágatokat *konzisztens vágatnak* nevezzük.

A KONZISZTENS-VÁGAT algoritmus vektor-időbélyegeket használ egy konzisztens vágat megkeresésére. Feltesszük, hogy minden processzor megkapja ugyanazt a $K = (k_1, \ldots, k_n)$ vágatot, mint bemenetet. Ezután a processzoroknak meg kell határozniuk egy olyan K' konzisztens vágatot, melyet K majorál. Minden P_i processzor rendelkezik egy $VT_i[0, 1, 2, \ldots]$ végtelen vektortáblával. A processzor utasításokat hajt végre, és tárolja a vektor-időbélyegeket a tábla egymásutáni bejegyzéseiként. Specifikusan, a táblázat m bejegyzése a processzor m-edik végrehajtott utasításához tartozó $VT_i[m]$ vektor-időbélyeg. $VT_i[0]$-t nullvektornak határozzuk meg. A P_i processzor elkezdi egy vágat kiszámítását, miután végrehajtotta a k_i utasítást. A processzor meghatározza azt a legnagyobb $k'_i \geq 0$ számot, amelyik legfeljebb k_i, és amelyre a K majorálja a $VT_i[k'_i]$ vektort. A processzorok által együtt megtalált $K' = (k'_1, \ldots, k'_n)$ vektorról kiderül, hogy egy konzisztens vágat.

23.24. tétel. *Bármely K vágat esetén a* KONZISZTENS-VÁGAT *algoritmus által megtalált K' vágat egy, a K által majorált konzisztens vágat.*

Bizonyítás. Figyeljük meg először, hogy a k_i-n túli VT_i bejegyzésekre nincs szükség. Ezeket a bejegyzéseken nem majorálja K, mivel e vektorok i-edik koordinátája kifejezetten nagyobb, mint k_i. Így valójában a keresésben VT_i első k_i bejegyzésére koncentrálhatunk.

Legyen $k'_i \geq 0$ a legnagyobb olyan bejegyzés, hogy a $VT_i[k'_i]$ vektort a K vektor majorálja. Tudjuk, hogy létezik ilyen vektor, mivel $VT_i[0]$ nullvektor, és azt bármilyen K vágat majorálja.

Azt, hogy $(k'_1, \ldots, k'_n)$ konzisztens vágat, ellentmondásra jutással bizonyítjuk. Tegyük fel, hogy a $(k'_1, \ldots, k'_n)$ vektor inkonzisztens vágat. Ekkor definíció szerint vannak olyan P_i és P_j processzorok, melyekre létezik a P_i processzornak egy olyan x utasítása, mely a k'_i számú utasítás után következik, és a végrehajtása a P_j processzor k'_j utasításánál korábban történik. Emlékezzünk vissza, hogy k'_i a legtávolabbi olyan VT_i-beli érték, melyet K majorál. Így a k'_i+1 értéket nem majorálja K, és mivel az összes következő, ideértve az x utasítást is, bejegyzés csak nagyobb koordinátával rendelkezhet, K ezeket sem majorálja. De mivel x végrehajtása a k'_j számú utasítás előtt történik, így a k'_j bejegyzés csak az x-hez tartozó bejegyzés megfelelő koordinátáinál magasabb koordinátával rendelkezhet, s így K nem majorálhatja $VT_j[k'_j]$-t sem. Ez ellentmond annak a feltételnek, hogy K majorálja $VT_j[k'_j]$-t. Tehát $(k'_1, \ldots, k'_n)$-nak konzisztens vágatnak kell lennie. ■

A konzisztens vágat megtalálására van egy triviális algoritmus. Az algoritmus a $K' = (0, \ldots, 0)$ vágatot választja ki. Azonban a KONZISZTENS-VÁGAT algoritmus jobb abban az értelemben, hogy a talált konzisztens vágat maximális. Annak belátását, hogy ez valóban igaz, feladatnak hagyjuk.

A konzisztens vágat megtalálására van egy másik mód is. A KONZISZTENS-VÁGAT algoritmus megköveteli, hogy az üzenetekhez vektor-időbélyegeket csatoljunk, és emlékezzünk azon A algoritmus által eddig végrehajtott összes utasításhoz tartozó vektor-időbélyegekre, melyekhez tartozó konzisztens vágatt ki akarjuk számolni. Ez túl költséges lehet. Az OSZTOTT-PILLANATKÉP algoritmus elkerüli ezeket a költségeket. Az algoritmusban egy processzor úgy kezdeményezi a konzisztens vágat kiszámítását, hogy elárasztja a hálózatot egy olyan speciális üzenettel, mely úgy viselkedik, mint egy, az A algoritmust konzisztensen vágó kard. Ahhoz, hogy belássuk, a vágat tényleg konzisztens, megköveteljük, hogy az üzenetek fogadása azok küldésének sorrendjében történjen. Egy ilyen rendezés sorozatszámokkal megvalósítható.

Az OSZTOTT-PILLANATKÉP algoritmusban minden P_i processzor rendelkezik egy *számláló*-nak nevezett változóval, mely megszámolja az A algoritmusbeli, a processzor által eddig végrehajtott utasításokat. Továbbá, a processzor rendelkezik egy k_i változóval, mely a vágat i-edik koordinátáját tárolja. Ennek a változónak a kezdőértéke $\perp$. Mivel a *számláló* csak az A algoritmus utasításait számlálja, az OSZTOTT-PILLANATKÉP algoritmus utasításai nincsenek hatással a *számláló* változókra. Valamilyen értelemben az OSZTOTT-PILLANATKÉP algoritmus a „háttérben" fut. Tegyük fel, hogy létezik pontosan egy olyan processzor, mely úgy dönthet, pillanatképeket készít az osztott rendszerről. Ha így dönt, a processzor „elárasztja" a a hálózatot egy speciális, <*pillanatkép*> üzenettel. Specifikusan, a processzor az összes szomszédjának elküldi ezt az üzenetet, és *számláló* változóhoz a k_i értéket rendeli. Amikor a P_j processzor megkapja az üzenetet, és a k_j változó értéke még mindig $\perp$, akkor a processzor az összes szomszédjának elküldi a <*pillanatkép*> üzenetet és k_j-hoz az *aktuális* számláló értéket rendeli. A <*pillanatkép*> üzenet küldése és az értékadás közben a processzor nem hajt végre utasítást A-ból. (Az OSZTOTT-PILLANATKÉP algoritmusra úgy tekinthetünk, mint egy „megszakítás"-ra.) Az algoritmus egy konzisztens vágatot számít ki.

23.25. tétel. *Bármely két P_i és P_j processzorra a P_i-ből P_j-be küldött üzenetek érkezzenek meg a küldés sorrendjében. Az* OSZTOTT-PILLANATKÉP *algoritmus végül megtalál egy $(k_1, \ldots, k_n)$ konzisztens vágatot. Az algoritmus $O(e)$ darab üzenetet küld, ahol e a gráf éleinek száma.*

Bizonyítás. Az a tény, hogy a k_i változó valamikor el fog térni $\perp$-től, a modellből következik, mivel feltételezzük, hogy az utasítások valamikor végrehajtódnak és az üzenetek kézbesítésre kerülnek, így a <*pillanatkép*> üzenet valamikor minden csúcshoz megérkezik.

Tegyük fel, hogy $(k_1, \ldots, k_n)$ nem konzisztens vágat. Ekkor létezik egy olyan P_j processzor, hogy a $(k_j + 1)$-edik vagy későbbi utasítás küld egy, a <*pillanatkép*> üzenettől eltérő, $< M >$ üzenetet, és az üzenetet megkapják, mielőtt egy P_i processzor a k_i-edik utasítást hajtja végre, vagy éppen aközben. Így az $< M >$ üzenetet azután kellett elküldeni, miután a P_j elküldte a <*pillanatkép*> üzenetet a P_i-hez. De mivel az üzeneteket a küldés sorrendjében kapják meg, a P_i a <*pillanatkép*> üzenetet $< M >$ előtt dolgozza fel. De ekkor az $< M >$ azután érkezik, hogy pillanatképet vettek P_i-nél. Ez a kívánt ellentmondás. ■

Gyakorlatok

23.5-1. Mutassuk meg, hogy a logikai idő megőrzi a *korábban történik* relációt, azaz mutassa meg, hogy ha $x <_{HB} y$, akkor $LT(x) < LT(y)$, ahol LT a logikai időt jelöli.

23.5-2. Mutassuk meg, hogy bármely n processzor közti konkurenciát rögzítő vektoróra legalább n koordinátájú.

23.5-3. Mutassuk meg, hogy a KONZISZTENS-VÁGAT algoritmus által kiszámolt K' vektor valójában egy maximális konzisztens vágat, melyet K majorál, azaz nincs olyan K'-tól eltérő K'', mely majorálja K'-t miközben K majorálja K''-t.

23.6. Kommunikációs szolgáltatások

Az üzenetküldéssel kommunikáló processzorokból álló osztott rendszerek alapvető problémái közé tartozik az információk terjesztésének és összegyűjtésének feladata. Sok kommunikációs hálózatra készített osztott algoritmus felépíthető üzenetszórási és többes üzenetszórási szolgáltatások implementálásával. Ebben az alfejezetben bemutatunk néhány alapvető kommunikációs szolgáltatást az üzenetküldő modellben. Az ilyen szolgáltatások tipikusan valamilyen szolgáltatásminőségi követelményeket kell, hogy kielégítsenek, melyek az üzenetsorrendre és a megbízhatóságra vonatkoznak. Először az üzenetszórási szolgáltatásokra koncentrálunk, majd az általánosabb többes szórási szolgáltatásokat tárgyaljuk.

23.6.1. Az üzenetszóró szolgáltatások tulajdonságai

Az üzenetszórási problémában egy választott P_i processzor, melyet ***forrásnak*** vagy küldőnek neveznek, egy m üzenetet akar a rendszerben lévő összes processzorhoz (beleértve a saját magát is) elküldeni. Az üzenetszóró szolgáltatás interfészét a következőképpen határozzák meg:

bc-send$_i(m, qos)$: a P_i processzor egy eseménye, mely egy m üzenetet küld az összes processzornak.

bc-recv$_i(m, j, qos)$: a P_i processzor egy eseménye, mely egy – a P_j processzor által küldött – m üzenetet fogad.

A fenti meghatározásokban a *qos* a rendszer által nyújtott ***szolgáltatás minőségét*** (**Q**uality **O**f **S**ervice) jelöli. Kétféle szolgáltatásminőséget fogunk vizsgálni:

Sorrend: hogyan függ az üzenetek fogadásának sorrendje az a forrás által küldött üzenetek sorrendjétől?

Megbízhatóság: hogyan függ a megkapott üzenetek halmaza a rendszerben előforduló hibáktól?

Az üzenetküldésen alapuló osztott rendszerek alapmodellje nem garantál semmit az üzenetek rendezésére vagy a megbízhatóságára nézve. Az alapmodellben csak azt tesszük fel, hogy az összes processzorpárt egy-egy vonal köti össze, és az üzenetkézbesítés független az egyes vonalakon. Az üzenetek megkapásának sorrendje nem áll összefüggésben az üzenetek elküldésének sorrendjével, az üzenetek elveszhetnek a küldők vagy a fogadók megállása miatt.

Bemutatunk néhány hasznos követelményt az üzenetszóró szolgáltatások rendezésére és megbízhatóságára nézve. A fő kérdés az, hogy az alap rendszermodellből kiindulva hogyan implementáljunk egy erősebb szolgáltatást egy gyengébb szolgáltatás tetején.

A rendezésre vonatkozó követelmények változatai

A ***korábban történt*** definícióját üzenetekre alkalmazva azt mondjuk, hogy az m üzenet m'-nél korábban történt, ha vagy m-et és m'-t ugyanaz a processzor küldte, és m-t korábban küldte, mint m'-t, vagy az m-re vonatkozó bc-recv esemény korábban következik be, mint az m'-re vonatkozó bc-send esemény (a *korábban történt* kifejezés utasításokra vonatkozó definícióját a 23.5.2 pontban adtuk meg).

Az üzenetküldések sorrendje szempontjából az általános üzenetszórási szolgáltatásokat négyféleképpen osztályozhatjuk:

Alap üzenetszórás: az üzenetek sorrendje nem garantált.

Egyforrású FIFO (*először be, először ki*): az egy processzor által küldött üzeneteket az egyes processzorok a küldés sorrendjében kapják meg. Pontosabban, minden P_i, P_j processzorpárra és m, m' üzenetekre, ha a P_i processzor m-et m' előtt küldi, akkor a P_j nem kapja meg hamarabb m'-t, mint m-et.

Oksági sorrrend: az üzenetek előfordulásuk sorrendjében érkeznek meg. Pontosabban, minden m, m' üzenetpárra és minden P_i processzorra, ha m m' előtt történik, akkor P_i nem kapja meg m'-t m előtt.

Teljes sorrend: az összes processzor a kapott üzeneteknek azonos sorrendjét biztosítja. Pontosabban, minden P_i, P_j processzor-, és minden m, m' üzenetpárra, ha a P_i processzor az m üzenetet m' előtt kapja meg, akkor a P_j processzor sem kapja meg m'-t m előtt.

Könnyű belátni, hogy az oksági sorrend maga után vonja az egyforrású FIFO követelményeket (mivel a *korábban történik* reláció az üzenetek esetén magában foglalja az egyetlen processzor által küldött üzenetsorrendet), valamint, hogy az összes felsorolt szolgáltatás magától értődően maga után vonja az alap üzenetszórást. A négy szolgáltatás közt nincs további reláció. Például, vannak olyan végrehajtások, melyek maguk után vonják az egyetlen

forrás FIFO tulajdonságot, de az oksági sorrendet nem. Tekintsük a P_0 és P_1 processzorokat. Az első eseményben a P_0 üzenetszórással elküldi az m üzenetet, majd a P_1 processzor megkapja m'-t. Ebből az következik, hogy m m' előtt történik. Ugyanakkor, ha a P_0 processzor m'-t kapja meg m előtt, ami meg is történhet, akkor ez a végrehajtás megsérti az oksági sorrend követelményét. Megjegyezzük, hogy az egyetlen forrás FIFO követelmény triviálisan megáll, hiszen az egyes processzorok csak egy üzenetet küldenek.

Az alapvető üzenetszóró szolgáltatást ***bb***-vel, az egyetlen forrás FIFO-t ***ssf***-fel, az oksági sorrendet ***co***-val, a teljes sorrendet pedig ***to***-val jelöljük.

Megbízhatósági követelmények
A meghibásodás nélküli modellben az üzenetszóró szolgáltatások következő tulajdonságait szeretnénk garantálni:

Integritás: a bc-recv eseményben kapott minden egyes m üzenetet valamilyen bc-send eseményben küldték.

Üzenetek duplikálásának elkerülése: egyetlen processzor sem kap meg egy üzenetet többször.

Létezés: az összes küldött üzenetet megkapja az összes processzor.

A hibák melletti modellben definiáljuk a ***megbízható*** üzenetszóró szolgáltatást, mely kielégíti az integritási, a kettősségmentességi és a kétféle létezési tulajdonságot:

Hibamentes létezés: bármely, egy hibamentes P_i processzor által küldött m üzenetet meg kell, hogy kapjon minden hibamentes processzor.

Meghibásodás melletti létezés: bármely, egy meghibásodott processzor által küldött üzenetet meg kell, hogy kapjon minden hibamentes processzor vagy egyik ilyen processzor sem kaphatja meg az üzenetet.

A megbízható alap üzenetszóró szolgáltatást ***rbb***-vel, a megbízható egyetlen forrás FIFO-t ***rssf***-fel, a megbízható oksági sorrendet ***rco***-val, a megbízható teljes sorrendet pedig ***rto***-val jelöljük.

23.6.2. Rendezett üzenetszóró szolgáltatások

A továbbiakban leírjuk a különböző üzenetszóró szolgáltatások algoritmusainak implementációit.

Alap üzenetszórás megvalósítása aszinkron pont-pont üzenetküldésre épülve
A bb szolgáltatást a következőképpen valósítják meg. A bc-send$_i(m, bb)$ esemény előfordulásakor a P_i processzor az összes vonalon elküldi az m üzenetet P_i-ből P_j-be, ahol $0 \leq i \leq n-1$. Ha egy m üzenet megérkezik a P_j processzorhoz, akkor az engedélyezi a bc-recv$_j(m, i, bb)$ eseményt.

A megbízhatóság biztosításához a következőt tesszük. Létrehozunk egy megbízható üzenetszórás szolgáltatást az alap üzenetszórás szolgáltatásra épülve. Amikor a bc-send$_i(m, rbb)$ esemény bekövetkezik, a processzor engedélyezi a bc-send$_i(\langle m, i\rangle, bb)$ eseményt. Ha a bc-recv$_j(\langle m, i\rangle, k, bb)$ esemény bekövetkezik, és az m üzenetkoordináta először jelenik meg, akkor a P_j processzor először engedélyezi a bc-send$_j(\langle m, i\rangle, bb)$ eseményt (hogy informálja a többi hibamentes processzort az m üzenetről abban az esetben, ha a P_i processzor hibás), majd engedélyezi a bc-recv$_j(m, i, rbb)$ eseményt.

Bebizonyítjuk, hogy a fenti algoritmus megbízhatóságot biztosít az alap üzenetszórás szolgáltatás számára. Először figyeljük meg, hogy az integritás és a kettősségmentesség tulajdonságok közvetlenül következnek abból a tényből, hogy az összes P_j processzor csak akkor engedélyezi bc-recv$_j(m, i, rbb)$-t, ha az m üzenetkoordináta először érkezett. A hibamentes létezés megőrződik, hiszen a hibamentes processzorok közti vonalak helyesen engedélyezik a bc-recv$_j(\cdot, \cdot, bb)$ eseményt.

A meghibásodás melletti létezést az a tény garantálja, hogy ha van egy olyan P_j hibamentes processzor, mely a meghibásodott P_i-től egy m üzenetet kap, akkor a bc-recv$_j(m, i, rbb)$ engedélyezése előtt a P_j processzor a bc-send$_j$ esemény segítségével elküldi az m üzenetet. Mivel P_j hibamentes, az összes hibamentes p_k processzor megkapja az m üzenetet valamely bc-recv$_k(\langle m, i\rangle, \cdot, bb)$ eseményben, majd elfogadja azt (a bc-recv$_k(m, i, rbb)$ esemény engedélyezésével) az első ilyen esemény során.

Egyetlen forrás FIFO megvalósítása az alap üzenetszóró szolgáltatásra épülve
Minden egyes P_i processzor rendelkezik egy számlálóval (időbélyeg), melynek kezdőértéke 0. Ha bekövetkezik a bc-send$_i(m, ssf)$ esemény, akkor a P_i processzor úgy küldi el az m üzenetet, hogy csatolja az aktuális időbélyeget a bc-send$_i(< m, időbélyeg >, bb)$ segítségével. Ha egy bc-recv$_j(< m, t >, i, bb)$ esemény bekövetkezik, akkor a P_j processzor engedélyezi a bc-recv$_j(m, i, ssf)$ eseményt éppen a bc-recv$_j(m_0, i, ssf)$, ..., bc-recv$_j(m_{t-1}, i, ssf)$ események engedélyezése után, ahol $m_0, \ldots, m_{t-1}$ azok az üzenetek, melyeknél az bc-recv$_j(< m_0, 0 >, i, bb)$,...,bc-recv$_j(< m_{t-1}, t-1 >, i, bb)$ események engedélyezve vannak.

Megjegyzzük, hogy ha háttérszolgáltatásként a megbízható alap üzenetszórást alkalmazzuk az alap üzenetszórás helyett, az egyetlen forrás FIFO fenti implementációja megbízható egyetlen forrás FIFO szolgáltatást eredményez. A bizonyítást gyakorlatként az Olvasóra bízzuk.

Oksági sorrend és teljes sorrend implementálás az egyetlen forrás FIFO szolgáltatásra épülve
Bemutatunk egy rendezett üzenetszórási algoritmust, mely az egyetlen forrás FIFO üzenetszóró rendszert biztosító aszinkron üzenetküldő rendszerre épül. Ez is időbélyegeket használ, de kifinomultabb módon, mint az *ssf*. Az oksági és teljes rendezési feltételeknek megfelelő szolgáltatást *cto*-val jelöljük.

Minden egyes P_i processzor egy helyi T tömbben tartja nyilván a növekvő számlálóját (időbélyeg) és a többi processzor becsült értékeit. Az időbélyegek arra szolgálnak, hogy küldés előtt ezekkel címkézik meg az üzeneteket: Ha P_i üzenetszórással akar küldeni egy üzenetet, növeli az időbélyegét, és ezzel címkézi meg a küldeni kívánt üzenetet (11–13. sorok). A végrehajtás során a P_i processzor megbecsüli a többi processzor időbélyegeinek értékét a T helyi vektorban. Ha a P_i processzor egy t (P_j időbélyege) címkével címkézett üzenetet kap P_j-től, elhelyezi t-t a $T[j]$-ben (23. és 32. sor). A P_i processzor úgy állítja be az aktuális időbélyegét, hogy a T vektorban becsült időbélyegek maximumához hozzáad egyet (24–26. sorok) Az időbélyeg frissítése után a processzor egy frissítés üzenetet küld. Egy processzor akkor fogad el egy t időbélyeggel címkézett m üzenetet a P_j processzortól, ha a (t, j) pár a legkisebb a többi fogadott üzenet között (42. sor) és minden egyes processzor legalább akkora nagy időbélyeggel rendelkezik, mint amit a P_i ismer (43. sor). A részletek az alábbi kódban találhatók.

RENDEZETT-ÜZENETSZÓRÁS

```
     Kód a P_i processzorokra, 0 ≤ i ≤ n − 1
 1   kezdőértékek beállítása
 2       T[j] ← 0 minden 0 ≤ j ≤ n − 1 esetén

11   if bc-send_i(m, cto) előfordul then
12       T[i] ← T[i] + 1
13       enable bc-send_i(< m, T[i] >, ssf)

21   if bc-recv_i(< m, t >, j, ssf) előfordul then
22       add (m, t, j) hármas hozzáadása a függőkben levőkhöz
23       T[j] ← t
24       if t > T[i] then
25           T[i] ← t
26           enable bc-send_i(< update, T[i] >, ssf)

31   if bc-recv_i(< update, t >, j, ssf) előfordul then
32       T[j] ← t

41   if
42       (m, t, j) függő hármas, melyre (t, j) a legkisebb és
43       t ≤ T[k] minden 0 ≤ k ≤ n − 1 esetén
44   then
45       enable bc-recv_i(m, j, cto)
46       remove (m, t, j) hármas eltávolítása a függők listájáról
```

A RENDEZETT-ÜZENETSZÓRÁS algoritmus kielégíti az oksági sorrend követelményt. A bizonyítást az Olvasóra hagyjuk gyakorlatként (a későbbiekben bemutatjuk, hogy hogyan lehet elérni az erősebb megbízható rendezett oksági sorrend szolgáltatást, és a bizonyítást adunk arra az erősebb esetre).

23.26. tétel. *A* RENDEZETT-ÜZENETSZÓRÁS *algoritmus kielégíti a teljes sorrend feltételt.*

Bizonyítás. Az integritás abból a tényből következik, hogy az egyes processzorok csak akkor engedélyezhetik a bc-recv$_i(m, j, cto)$ eseményt, ha a (m, t, j) hármas függő (41–45. sorok), ami csak a P_j processzortól érkező m üzenet fogadása után következhet be. A kettősségmentesség tulajdonságot az a tény garantálja, hogy legfeljebb egy olyan hármas van, amelyet a P_j processzor küldött és tartalmazza az m üzenetet (13. és 21–22. sorok).

A létezési feltétel kielégítése abból a tényből következik, hogy minden egyes függő hármas a végrehajtás valamely pillanatában kielégíti a 42–43. sorokban lévő feltételt.

E tény bizonyítása a végrehajtásban szereplő eseményekre vonatkozó indukción alapul. Tegyük fel, ellentmondó módon, hogy a (m, t, j) hármas az, amelyre a (t, j) a legkisebb és a végrehajtás során sohasem elégíti ki a 42–43. sorokban lévő feltételeket. Ebből az következik, hogy van olyan pillanat, mikor a (m, t, j) a legkisebb (t, j) koordinátákkal rendelkezik a P_i processzor függő hármasai között. Emiatt, ettől a pillanattól fogva valamely k-ra meg kell sértenie a 43. sorban lévő feltételt. Megjegyezzük, hogy a 23–25. sorok frissítési szabályai miatt $k \neq i, j$. Ebből az következik, hogy a P_i processzor sohasem kap olyan üzenetet P_k-tól,

mely $(t-1)$-nél nagyobb időbélyeggel van címkézve. Ez a 24–26. sorok frissítési szabályai miatt azt jelenti, hogy a P_k processzor sohasem kapja meg P_j-től az $< m, t >$ üzenetet, ami ellentmond a *ssf* szolgáltatás létezési feltételének.

A teljes rendezés tulajdonság bizonyításához elég azt belátni, hogy minden P_i processzor és minden, a P_k, P_l processzorok által t, t' időbélyeggel küldött megfelelő m, m' üzenet esetén a (m, t, k), (m', t', l) hármasokat $(t, k), (t', l)$ lexikografikus sorrendjében elfogadják. Két eset van.

1. eset. A végrehajtás valamely pillanatában mindkét hármas függő lesz a P_i processzornál. Ekkor a 42. sorban lévő feltétel garantálja a $(t, k), (t', l)$ sorrendben történő elfogadást.

2. eset. Az általánosság megszorítása nélkül feltehetjük, hogy a P_i processzor elfogadja a (m, t, k) hármast mielőtt a (m', t', l) hármas függő lesz. Ha $(t, k) < (t', l)$, akkor az elfogadás még mindig a $(t, k), (t', l)$ sorrendnek felel meg. Máskülönben $(t, k) > (t', l)$, és a 43. sor feltétele miatt azt kapjuk, hogy $t \leq T[l]$ és következményképpen $t' \leq T[l]$. Ez nem fordulhat elő az *ssf* követelmény és amiatt a feltételezés miatt, hogy a P_i processzor még nem kapta meg az $< m', t' >$ üzenetet l-től az *ssf* üzenetszóró szolgáltatáson keresztül. ■

Most az oksági rendezés és a teljes rendezés szolgáltatások megbízható változatait vizsgáljuk. A megbízható oksági rendezés követelményei a processzormegállás melletti aszinkron üzenetküldő rendszerben a következő algoritmussal valósíthatók meg a megbízható alap üzenetszórásra épülve. Ugyanazok az adatszerkezetek, mint a korábbi rendezett üzenetszóró algoritmusnál. A MEGBÍZHATÓ-OKSÁGI-RENDEZÉS algoritmus és a RENDEZETT-ÜZENETSZÓRÁS algoritmus között fő különbségek a következőek: Az egészből álló időbélyegek helyett vektoros T időbélyeget használ és nem becsüli más processzorok időbélyegeit, csak lexikografikusan összehasonlítja a saját (vektoros) időbélyegeit a kapottakkal. A P_i processzor vektoros időbélyegei mögötti intuíció az, hogy az tárolja azt az információt, hogy hány üzenetet küldött P_i és hány üzenetet fogadott el a P_i az egyes P_k-któl, ahol $k \neq i$.

Az algoritmus végrehajtása folyamán a P_i processzor azelőtt növeli a megfelelő i pozíciót a T időbélyeg-vektorában, mielőtt egy új üzenetet küld (12. sor), valamint növeli a vektor-időbélyeg j-edik pozícióját, miután egy új üzenetet fogadott el P_j-től (38. sor). Egy új, a P_j-től származó, $\hat{T}$ vektor-időbélyeggel rendelkező, üzenet fogadása után P_i felveszi a $(m, \hat{T}, j)$ hármast a függők listájára, és akkor fogadja el ezt a hármast, ha az a P_j-től származó első, el nem fogadott üzenet (feltétel a 33. sorban), és a P_j-től származó elfogadott üzenetek száma (a $P_k \neq P_i$ processzorokra) m küldésének pillanatában nem nagyobb, mint a P_i aktuális időpillanatában (feltétel a 34. sorban). A következőkben bemutatjuk az algoritmus részletes kódját.

MEGBÍZHATÓ-OKSÁGI-RENDEZÉSŰ-ÜZENETSZÓRÁS

```
     Kód a P_i processzorokra, 0 ≤ i ≤ n − 1 esetén
 1   kezdőértékek beállítása
 2       T[j] ← 0 minden 0 ≤ j ≤ n − 1 esetén
 3       a függőek listája üres

11   if bc-send_i(m, rco) előfordul then
12       T[i] ← T[i] + 1
13       enable bc-send_i(< m, T >, rbb)
```

21 **if** bc-recv$_i$(< $m, \hat{T}$ >, j, rbb) előfordul **then**
22 **add** $(m, \hat{T}, j)$ hármas felvétele a függőek listájába

31 **if**
32 $(m, \hat{T}, j)$ függő hármas és
33 $\hat{T}[j] = T[j] + 1$, és
34 $\hat{T}[k] \leq T[k]$ minden $k \neq i$ esetén
35 **then**
36 **enable** bc-recv$_i(m, j, rco)$
37 **remove** $(m, \hat{T}, j)$ hármas eltávolítása a függőek listájából
38 $T[j] \leftarrow T[j] + 1$

Most bebizonyítjuk, hogy a MEGBÍZHATÓ-OKSÁGI-RENDEZÉSŰ-ÜZENETSZÓRÁS algoritmus megbízható oksági üzenetszórási szolgáltatást nyújt a megbízható alap üzenetszórásra épülő rendszerben. Az integritás és a kettősségmentesség tulajdonságokat az *rbb* üzenetszóró szolgáltatás és azok a tények garantálják, hogy minden egyes üzenet legalább egyszer felkerül a függők listájára, és a nem fogadott üzenetek sohosem kerülnek fel erre a listára. A hibamentesség és a hiba melletti létezési tulajdonságokat a végrehajtáson vett indukcióval lehet bizonyítani azon tények felhasználásával, hogy a hibamentes processzorok az összes küldött üzenetet megkapják, melyek garantálják, hogy a 33–34. sorokban szereplő feltételek valamikor végül teljesülnek. Az oksági sorrend feltétel teljesül, mivel ha egy m üzenet m' előtt történik, akkor minden egyes P_i processzor az m, illetve m' üzenetekhez tartozó vektorok, azaz $\hat{T}$ és $\hat{T}'$ lexikografikus sorrendjének megfelelően fogadja el az m, m' üzeneteket, valamint ebben az esetben e vektortömbök összehasonlíthatóak. A bizonyítás részleteit az Olvasóra bízzuk (lásd 23.6-6. gyakorlat).

Megjegyezzük, hogy a megbízható teljes rendezéses üzenetszórás szolgáltatás nem valósítható meg a processzor-meghibásodások melletti általános aszinkron beállítások mellett, mivel ez megoldaná az egyetértési problémát ebben a modellben. Az először elfogadott üzenet határozná meg a döntési értéket (ami ellentmond annak a ténynek, hogy az egyetértés nem oldható meg az általános modellben – lásd a 23.4.6. pontban).

23.6.3. Többes üzenetküldő szolgáltatások

A többes üzenetküldő szolgáltatások hasonlóak az üzenetszóró alkalmazásokhoz azzal a különbséggel, hogy az egyes többesküldésű üzenetek címzettje a processzorok halmazának egy adott részhalmaza. A többes üzenetküldő szolgáltatásokban kétfajta esemény van, ahol a qos a megkívánt szolgáltatásminőséget jelöli:

mc-send$_i(m, D, qos)$: a P_i processzor egy eseménye, mely az m üzenetet küldi az azonosítókkal együtt a $D \subseteq \{0, \ldots, n-1\}$ célhalmazban szereplő processzorokhoz.

mc-recv$_i(m, j, qos)$: a P_i processzor egy eseménye, mely a P_j processzor által küldött m üzenetet fogadja.

Megjegyezzük, hogy az mc-recv esemény hasonló a bc-recv eseményhez.

Az üzenetszórásos szolgáltatás esetén is hasznos rendezési és megbízhatósági tulajdonságokat szeretnénk nyújtani a többes küldés szolgáltatások számára. A rendezési követelményeket átvehetjük az üzenetszóró szolgáltatásoktól. Az alap többes üzenetküldés nem igé-

nyel rendezési tulajdonságokat. Az Egyforrású FIFO megköveteli, hogy ha egy processzor többes küldést végez (valószínűleg különböző célhalmazokba), akkor az egyes processzorok által fogadott üzeneteket (ha léteznek) ugyanabban a sorrendben kell fogadni, amelyben a forrás azokat küldte. Az oksági sorrend meghatározása ugyanaz marad. A teljes sorrend helyett, melyet a célhalmazok különbözősége miatt nehéz elérni, egy másik rendezési tulajdonságot definiálunk:

Részben teljes rendezés: az összes processzor által fogadott üzenetek sorrendje kiterjeszthető az üzenetek teljes rendezésére. Pontosabban, bármely m, m' üzenet, és P_i, P_j processzorpárra ha P_i és P_j is fogadta az m, m' üzeneteket, akkor azokat P_i és P_j is ugyanabban a sorrendben fogadta.

A többes üzenetküldés megbízhatósági tulajdonságai valamelyest eltérnek a megbízható üzenetszórásra vonatkozó feltételektől.

Integritás: minden, mc-recv$_i$ esemény során fogadott m üzenetet olyan mc-send esemény küldött, melyben a P_i processzor szerepelt a célhalmazban.

Üzenetek duplikálásának elkerülése: egyik processzor sem kap meg egy üzenetet többször.

Hibamentes létezés: minden, a P_i hibamentes processzor által küldött m üzenetet megkap minden, a célhalmazban szereplő, processzor.

Meghibásodás melletti létezés: bármilyen üzenetet, melyet egy meghibásodott processzor küldött, vagy megkap minden, a célhalmazban szereplő, hibamentes processzor, vagy közülük egyik sem kapja meg.

A rendezett és megbízható többes küldés implementálásának egyik módja a megfelelő üzenetszórási szolgáltatás felhasználása. (A Részben teljes rendezés esetén megfelelő üzenetszórási követelmény a teljes rendezés.) Pontosabban, az mc-send$_i(m, D, qos)$ esemény előfordulásakor a P_i processzor engedélyezi a bc-send$_i(< m, D >, qos)$ eseményt. Egy bc-recv$_j(< m, D >, i, qos)$ esemény előfordulásakor a P_j processzor engedélyezi az mc-recv$_j(m, i, qos)$ eseményt, ha $P_j \in D$, különben figyelmen kívül hagyja az eseményt. Annak bizonyítását, hogy egy ilyen módszer biztosítja a kívánt szolgáltatásminőségi követelményt, gyakorlatként az Olvasóra bízzuk.

Gyakorlatok

23.6-1. Tervezzünk futtatást azt bemutatandó, hogy nincs kapcsolat az Oksági sorrend és a Teljes sorrend között, valamint az Egyforrású FIFO és a Teljes sorrendű üzenetszóró szolgáltatások között. Az egyszerűség kedvéért tekintsünk két processzort és két elküldött üzenetet.

23.6-2. Az Egyforrású FIFO-t és az Oksági sorrend követelményeit kielégítő üzenetszóró szolgáltatás kielégíti a Teljes sorrend tulajdonságot? Az Egyforrású FIFO-t és a Teljes sorrend követelményeit kielégítő üzenetszóró szolgáltatás kielégíti az Oksági sorrend tulajdonságot? Ha igen, adjunk rá bizonyítást, ha nem, mutassunk be egy ellenpéldát.

23.6-3. Mutassuk meg, hogy Egyforrású FIFO szolgáltatás megvalósításában ALAP-ÜZENETSZÓRÁS helyett MEGBÍZHATÓ-ALAP-ÜZENETSZÓRÁS-t alkalmazva megbízható Egyforrású FIFO üzenetszórást kapunk.

23.6-4. Bizonyítsuk be, hogy RENDEZETT-ÜZENETSZÓRÁS algoritmus az Egyforrású FIFO szolgáltatáson alapulva oksági sorrendű szolgáltatást valósít meg.

23.6-5. Mennyi a RENDEZETT-ÜZENETSZÓRÓ-SZOLGÁLTATÁS algoritmusban csúcsról csúcsra küldött üzenetek teljes száma k számú üzenetszórás esetén?
23.6-6. Bizonyítsuk be részletesen, hogy a MEGBÍZHATÓ-OKSÁGI-RENDEZÉSŰ-ÜZENETSZÓRÁS algoritmus megbízható oksági rendezésű üzentszórást biztosít a megbízhtaó alap üzenetszórásra épülő rendszerben.
23.6-7. Becsüljük meg a MEGBÍZHATÓ-OKSÁGI-SORRENDŰ-ÜZENETSZÓRÁS végrehajtása közben csúcsról csúcsra küldött üzenetek teljes számát, ha az k számú üzenetszórást hajt végre, és a végrehajtás során $f < n$ processzor omlik össze.
23.6-8. Mutassuk be a MEGBÍZHATÓ-OKSÁGI-SORRENDŰ-ÜZENETSZÓRÁS algoritmus egy olyan végrehajtási sorozatát, amely megsérti a Teljes sorrend követelményt.
23.6-9. Írjunk kódot megbízható részleges sorrendű többes üzenetküldés szolgáltatás megvalósítására.
23.6-10. Mutassuk meg, hogy a megfelelő üzenetszolgáltatásra épülő többes üzenetküldés megvalósításának leírt módszere helyes.

23.7. Szóbeszédgyűjtő algoritmusok

A fejlettebb kommunikációs problémák algoritmusának összeállításánál a megbízható többes üzenetküldő szolgáltatások felhasználhatók építő blokként. Ebben a fejezetben ezt a módszert mutatjuk be a szóbeszédek megállásra hajlamos szinkron processzorokkal történő gyűjtésének problematikájára vonatkozóan. (Mivel csak tiszta végrehajtással foglalkozunk, feltételezzük, hogy legalább egy processzor működőképes marad a számítások végéig.)

23.7.1. Szóbeszédgyűjtési (pletyka) probléma és követelményei

A ***szóbeszédgyűjtés*** vagy *pletyka* klasszikus problémája a következőképpen határozható meg: Kezdetben minden processzor a saját részinformációjával rendelkezik, nevezzük ezt ***szóbeszédnek.*** A cél az, hogy minden egyes processzorral tudassuk az összes szóbeszédet.

A processzor megállásos modellben mindemellett szükségünk van a pletyka probléma átfogalmazására úgy, hogy tekintetbe vegyük a processzorok megállását. Mind az integritás, mind a duplikált üzenetek elkerülése tulajdonság ugyanaz itt is, mint a megbízható üzenetszóró szolgáltatásban, az egyetlen különbség (ami következik a pletyka probléma specifikációjából) a létezés követelményei között van:

Hibamentes létezés: Minden egyes hibamentes processzornak ismernie kell minden hibamentes processzor szóbeszédét.

Meghibásodás melletti létezés: Ha a P_i processzor végrehajtás közben összeomlott, valamennyi hibamentes processzor vagy tudja a P_i szóbeszédét, vagy azt tudja, hogy a P_i összeomlott.

A *pletyka* algoritmusok hatékonyságát a futási idővel és az üzenetszámmal jellemezzük. A futási idő méri a (szinkron) lépések számát a kezdettől a befejezésig. Az üzenetszám méri a csúcsról csúcsra küldött üzenetek teljes számát (pontosabban ha egy processzor három további processzornak küld el egy üzenetet, ez az üzenetszám szempontjából háromnak számít).

A következő egyszerű algoritmus mindössze egy szinkron lépésben végzi el a pletyka funkciót: minden egyes processzor elküldi a saját szóbeszédét az összes processzornak. Az algoritmus kifogástalan, mivel minden egyes megkapott üzenet tartalmaz egy szóbeszédet, és egy meg nem érkezett üzenet a küldő hibáját jelenti. Egy ilyen megoldásnak az a hátránya, hogy négyzetes számú üzenetelküldést kíván, ami igen rossz hatékonyságot jelent.

Mi úgy szeretnénk lefolytatni a pletykát, hogy ne csak gyors legyen, hanem kevesebb üzenetet kelljen csúcsról csúcsra küldeni. Van egy magától értetődő kompromisszum az idő és a kommunikáció között. Figyeljük meg, hogy egy processzormegállás-mentes rendszerben egy ilyen kompromisszum elérhető például az üzeneteknek egy (majdnem) teljes bináris fán való küldésével, ekkor a futási idő $O(\lg n)$, míg az üzenetszám $O(n \lg n)$. Így tehát a futási idő kismértékű növelésével elérhetünk egy közel lineáris növekedést az üzenetszámot illetően.

Ha az alap kommunikációs hálózat komponensei meghibásodhatnak, akkor a szabálytalan hibaminták zavarják az információáramlást, ezzel jóval hosszabbá téve a pletykázás folyamatát. Ebben a fejezetben azzal a kérdéssel foglalkozunk, hogy mi a legmegfelelőbb kompromisszum a futási idő és az üzenetszám között egy processzor-megállást megengedő modellben.

23.7.2. Hatékony pletyka algoritmus

Ebben a részben a *pletyka* algoritmusoknak a családját írjuk le, amely algoritmusok között találhatunk néhány hatékonyat is. Mindegyikük ugyanazon az általános kódon alapszik, hatékonyságuk pedig az általános algoritmusba beépített két adatstruktúra minőségétől függ. A célunk annak bebizonyítása, hogy találhatunk néhány olyan adatstruktúrát, amellyel kapott algoritmus mindig helyes, valamint hatékony is, ha a végrehajtás alatti megállások száma legfeljebb f, ahol $f \leq n - 1$ egy paraméter.

Az említett két struktúra, a kommunikációs gráf és a kommunikációs ütemezések leírásával kezdjük.

Kommunikációs gráf

Egy $G = (V, E)$ gráf csúcsok V halmazából és élek E halmazából áll. Ebben a fejezetben gráfon mindig ***egyszerű gráfot*** értünk, ami azt jelenti, hogy minden él egy csúcspár, és az élhez nincs irány rendelve. A gráfok kommunikációs minták leírására szolgálnak. Egy gráf csúcsainak V halmazát a szóban forgó osztott rendszer processzorai alkotják. Az E-ben lévő élek azon processzorpárokat határozzák meg, amelyek üzenetváltás útján egymással közvetlenül kommunikálnak, ez azonban nem feltétlenül jelenti azt, hogy közöttük létezik fizikai kapcsolat. A kommunikációs mechanizmust vonatkoztatjuk el: lehet, hogy az – az E-ben lévő éllel összekötött csúcsokon váltott – üzeneteket továbbítani kell és lehet, hogy át kell vinni a kommunikációs hálózat alapjául szolgáló – esetleg hosszú – útvonalon.

Egy adott n számú processzorhoz általunk használt gráftopológiák az egy végrehajtás közben tolerálni kívánt megállások f felső korlátjától függően különbözőek. A végrehajtás egy adott pontján ez egy olyan gráf, amelyet azon processzorok hoztak létre, amelyek a végrehajtás e lépéséig még nem álltak meg.

Ahhoz, hogy hatékony algoritmust kapjunk, a kommunikációs gráfnak ki rendelkeznie kell néhány szükséges tulajdonsággal, például a következő $\mathcal{R}(n, f)$ tulajdonsággal:

23.27. definíció. *Legyen $f < n$ pozitív egészek egy párja. A G gráfról azt mondjuk, hogy kielégíti az $\mathcal{R}(n, f)$ tulajdonságot, ha G-nek van n gráfpontja, és ha minden legalább $n - f$ méretű $R \subseteq G$ részgráfhoz létezik G-nek egy olyan $P(R)$ részgráfja, amelyre az alábbiak teljesülnek:*

1: $P(R) \subseteq R$	- öröklődés
2: $\lvert P(R)\rvert = \lvert R\rvert/7$	- nagy méret
3: A $P(R)$ *átmérője legfeljebb* $2 + 30 \ln n$	- logaritmikus kommunikáció
4: Ha $R_1 \subseteq R_2$*, akkor* $P(R_1) \subseteq P(R_2)$	- monotonitás.

Figyeljük meg, hogy $P(R)$ egy összefüggő gráf – akkor is, ha R nem az –, mivel átmérője véges. A következő eredmény igazolja, hogy létrehozhatók $\mathcal{R}(n, f)$ tulajdonságot kielégítő gráfok.

23.28. tétel. *Minden $f < n$-hez létezik $\mathcal{R}(n, f)$ tulajdonságot kielégítő $G(n, f)$ gráf. A $G(n, f)$ gráf Δ maximális fokszáma $O(n/(n - f))^{1.837}$.*

Kommunikációütemezés

A ***lokális permutáció*** a $[0\,..\,n-1]$ intervallumban lévő egész számok permutációja. Feltesszük, hogy a számítást megelőzően létezik az n lokális permutációk egy adott Π halmaza. Minden P_i processzornak létezik a Π-ből egy ilyen π_i permutációja. Az egyszerűség kedvéért tegyük fel, hogy $\pi_i(0) = P_i$. A lokális permutáció a szóbeszéd módszeres – a permutáció által megadott sorrendben való – összegyűjtésére alkalmazható, míg a kommunikációs gráf inkább a már összegyűjtött szóbeszédek cseréjére használható nagy és összefüggő, hibamentes gráfkomponensekben.

Általános algoritmus

Annak a célnak a meghatározásával kezdjük, amit a pletykáló algoritmusnak teljesítenie kell.

Akkor mondjuk, hogy a P_i ***processzor már hallott a*** P_j ***processzorról,*** ha P_i ismeri a P_j eredeti bemeneti szóbeszédét, vagy ha P_i tudja, hogy P_j már meghibásodott. Újraszövegezhetjük a pletykáló algoritmus helyességét abból a szempontból, hogy hallott-e már a többi processzorról: az algoritmus helyes, ha teljesíti az integritás és a kettősségmentesség tulajdonságokat, és ha az algoritmus befejeztével minden processzor hallott minden másik processzorról.

A pletykáló algoritmus kódja tartalmaz olyan objektumokat, amelyek függenek a rendszerben lévő processzorok n számától, valamint a „hatékonyan tolerált” meghibásodások $f < n$ felső korlátjától (ha a meghibásodások száma maximum f, akkor a tervezőalgoritmus üzenetbonyoultsága kicsi). A hozzávett paraméter egy τ megállási küszöb, ami befolyásolja a általános pletyka séma adott implementációjának futási idejét. Célunk egy olyan ÁLTALÁNOS-PLETYKA algoritmus konstruálása, amely helyes bármely hozzávett f, τ paraméter, tetszőleges kommunikációs gráf és ütemezési halmaz esetén, miközben hatékony f, τ bizonyos értékeire, valamint bizonyos $G(n, f)$ és Π struktúrákra.

Minden processzor ***gyűjtőként*** kezd pletykálni. Egy gyűjtő processzor a többi processzor szóbeszédeire vonatkozó információk után kutat – közvetlen kérdéseket küldve néhányukhoz. A gyűjtő, miután minden processzorról hallott már, ***terjesztővé*** válik. Az ezzel a státusszal rendelkező processzorok terjesztik ismereteiket – lokális véleményeket küldve kiválasztott másik processzoroknak.

Lokális vélemény
Minden P_i processzor kezdetben csak a saját azonosítóját, valamint saját bemenő ***szóbeszéd***$_i$ információját ismeri. A beérkező adatok tárolására a P_i processzor a következő tömböket építi fel: Szóbeszédek$_i$, Aktív$_i$ és Függő$_i$. Minegyik töm mérete n. Ezen tömbök mindegyike kezdeti értékként a NIL értéket tárolja. A P_i processzor egy X_i tömbjének j-edik bejegyzését $X_i[j]$-vel jelöljük – természetszerűen ez a bejegyzés a P_j processzorról tartalmaz valamilyen információt. A Szóbeszéd tömb a processzor által ismert összes szóbeszéd tárolására szolgál. Kezdetben a P_i processzor a Szóbeszédek$_i[i]$ értékét a saját bemenő *szóbeszéd*$_i$ értékére állítja. Minden alkalommal, amikor a P_i processzor megtud valamilyen *szóbeszéd*$_j$ információt, azonnal átállítja a Szóbeszédek$_i[j]$ értékét erre az értékre. Az Aktív tömb azon processzorok halmazát tárolja, amelyekről a tömb tulajdonosa úgy tudja, hogy összeomlott. Amikor a P_i processzor értesül arról, hogy a P_j processzor meghibásodott, Aktív$_i[j]$ értékét azonnal *meghibásodott* értékre állítja. Vegyük észre, hogy a P_i processzor akkor hallott a P_j processzorról, ha a Szóbeszédek$_i[j]$ és Aktív$_i[j]$ értékek valamelyike nem egyenlő a NIL értékkel.

A Függő tömb rendeltetése az, hogy elősegítse a terjesztést. Minden alkalommal, amikor a P_i processzor értesül arról, hogy valamely másik P_j processzor teljesen informált, azaz hogy az vagy egy terjesztő, vagy egy terjesztőként bejelentett processzor, ez az információ bekerül a Függő$_i[j]$ értékbe. A P_i processzor arra használja a Függő$_i$ tömböt, hogy szisztematikus módon küldhessen terjesztő üzeneteket, s teszi ezt úgy, hogy végignézi a Függő$_i$ tömböt, hogy megtalálja azokat a processzorokat, amelyek feltételezhetően még nem hallottak valamely másik processzorról.

A következő egy hasznos fogalommeghatározás az Aktív és a Függő tömb aktuális tartalmáról. A p_j processzor p_i ***szerint aktív,*** ha a p_i még nem kapott arra vonatkozó információt, hogy p_j összeomlott; ami azonos azzal, hogy az Aktív$_i[j]$ értéke NIL. A p_j processzort a p_i ***által értesítendőnek*** nevezzük, ha az a p_i szerint aktív, és ha a Függő$_i[j]$ értéke NIL.

Fázisok. A pletykáló algoritmus egy végrehajtása a processzorok összes lokális objektumot inicializáló műveletével kezdődik. A p_i processzor Szóbeszédek$_i$ listáját a NIL kezdeti értékkel tölti fel minden helyen, kivéve az i-ediket, ahol az érték *szóbeszéd*$_i$ lesz. A végrehajtás fennmaradó része egy ciklusként épül fel, amelyben fázisok ismétlődnek. Minden fázis három részből áll: üzenetek fogadása, helyi számítás, csoportos üzenetek. A fázisoknak két fajtája van: ***normál fázis*** és ***záró fázis***. Egy normál fázis közben a processzor üzeneteket fogad, frissíti helyi ismereteit, ellenőrzi a státuszukat, valamint elküldi ismereteit, a szóbeszédekkel kapcsolatos kérdéseit és a saját szóbeszédével kapcsolatos válaszokat a kommunikációs gráfban lévő szomszédaihoz. A záró fázis közben a processzor üzeneteket fogad, kérdéseket küld az összes olyan processzorhoz, amelyekről eddig még nem hallott, és válaszokat küld a saját szóbeszédével kapcsolatosan. A normál fázis τ-szor hajtódik végre; a τ szám egy ***megállási küszöb.*** Ezek után a záró fázis négyszer hajtódik végre. Ez egy általános pletykáló algoritmust határoz meg.

ÁLTALÁNOS-PLETYKA

```
    Kód a P_i, 0 ≤ i ≤ n − 1 processzorok esetén
1   kezdőértékek beállítása
2       a P_i processzor gyűjtővé válik
3       Szóbeszédek_i, Aktív_i és Függő_i tömbök kezdőértékének megadása
```

11 **repeat** τ-szor
12 normál fázis végrehajtása

20 **repeat** 4-szer
21 záró fázis végrehajtása

Most leírjuk a normál és a zárófázisban alkalmazott kommunikációt és üzenetfajtákat.

A normál fázis alatt használt gráf- és tartományüzenetek
A P_i processzor küldhet üzenetet a $G(n, f)$ gráfban lévő szomszédjának, amennyiben az P_i szerint aktív. Az ilyen üzenetet ***gráfüzenetnek*** hívjuk. Csak ilyen üzeneteket küldve a pletykálást nem szükségszerűen végzi el teljesen, mivel a megállások következtében a kommunikációs gráf válhat nem összefüggő gráffá. Így tehát másféle üzeneteket is kell küldeni annak érdekében, hogy szisztematikusan le tudjuk fedni az összes processzort. Egy ilyen típusú kommunikációban a P_i processzor a processzorokat a saját lokális π_i permutációja szerinti, azaz $\pi_i(0), \pi_i(1), \ldots, \pi_i(n-1)$ sorrendben rendezetteknek tekinti. Az ebben a folyamatban elküldött néhány további üzenetet ***tartományüzenetnek*** hívjuk.

A normál fázis alatt a processzorok a következő típusú tartományüzeneteket küldik: érdeklődő, válasz és értesítő üzenet. A P_i gyűjtő küld egy ***érdeklődő*** üzenetet az első olyan processzornak, amelyikről P_i eddig még nem hallott. Egy ilyen üzenet minden fogadója visszaküld egy tartományüzenetet, amelyet ***válaszüzenetnek*** nevezünk.

A terjesztők szintén küldenek a processzorok egy részhalmazának tartományüzeneteket. Az ilyen üzeneteket ***értesítő üzeneteknek*** nevezzük. A P_i terjesztő által kiválasztott célprocesszor az első olyan processzor, amit P_i-nek még értesítenie kell. Az értesítő üzeneteket nem szükséges megválaszolni: A küldő már ismeri az összes, szerinte aktív processzor szóbeszédét, az üzenet célja pedig az ismeretterjesztés.

A normál fázis pszeodokódja a következő oldalon van.

Utolsó remény üzenetek használata a záró fázis alatt
A záró fázis alatt küldött üzeneteket ***utolsó remény*** üzeneteknek nevezzük Ezen üzenetek az érdeklődő, válasz és értesítő kategóriákba sorolhatók csakúgy, mint a megfelelő tartományüzenetek, mivel ugyanazokat a célokat szolgálják. Azok a gyűjtők, amelyek nem hallottak még néhány processzorról, közvetlen érdeklődő üzenetet küldenek egyszerre az *összes* ilyen processzorhoz. Ezeket az üzeneteket ***érdeklődő*** üzeneteknek nevezzük. Ezeket az üzenetek a meg nem hibásodott fogadók a következő lépésben megválaszolják egy ***válasz*** üzenet küldésével. A következő fázisban minden egyes terjesztő küld egy üzenetet az *összes* általa értesítendő processzornak. Az ilyen üzeneteket ***értesítő*** üzeneteknek nevezzük.

A normál fázis egy lépésében egy processzor által küldött gráfüzenetek száma legfeljebb olyan nagy, mint a maximális csúcsok száma a kommunikációs gráfban. A normál fázis egy lépésében egy processzor által küldött tartományüzenetek száma legfeljebb olyan nagy, mint a fogadott érdeklődő üzenetek száma plusz egy konstans – így az összes processzor által a normál fázisokban elküldött üzenetek teljes száma számítható úgy, hogy az az elküldött érdeklődések számának (ami fázisonként és processzoronként egy) konstansszorosa. Ezzel ellentétben azonban nincs *eleve meghatározott* felső korlátja a záró fázis közben elküldött üzenetek számának. A τ megállási küszöb elég nagyra történő választásával ellenőrizhető, hogy a záró fázisban hány szóbeszédet lenne szükséges még begyűjteni.

NORMÁL-FÁZIS

```
    Kód a P_i, 0 ≤ i ≤ n − 1 processzorok esetén
 1  receive üzenetek

11  helyi számítások elvégzése
12      helyi tömbök frissítése
13      if P_i egy gyűjtő, ami az összes processzorról hallott már
14          then p_i terjesztővé válik
15      a célprocesszorok halmazának meghatározása: for minden P_j processzorra
16          if P_j a P_i szerint aktív és P_j a G(n, t) gráfban szomszédja P_i-nek
17              then adja hozzá P_j-t egy gráfüzenet célhalmazához
18          if P_i egy gyűjtő és P_j azon első processzor,
19              amelyről P_i még nem hallott
20              then küldjön egy érdeklődő üzenetet p_j-nek
21          if P_i terjesztő és P_j azon első processzor,
22              amely P_i által értesítendő
23              then küldjön egy értesítő üzenetet p_j-nek
24          if P_j egy gyűjtő, amelytől érdeklődő üzenet érkezett e fázis fogadó fázisában
25              then küldjön egy válasz üzenetet P_j-nek

30  send gráf/érdeklődő/értesítő/válasz üzenet küldése a célhalmazra vonatkozóan
```

Lokális vélemény frissítése

Egy processzor által elküldött üzenet magával hordozza a saját jelenlegi lokális ismereteit. Pontosabban a P_i processzor által küldött üzenet a következőket hordozza magával: a P_i azonosítót, a $\mathsf{Szóbeszédek}_i$, az $\mathsf{Aktív}_i$ és a $\mathsf{Függő}_i$ tömböt, valamint egy címkét, amely a fogadót értesíti az üzenet karakteréről. A címke a következők egyike lehet: *gráfüzenet, érdeklődés_gyűjtőtől, értesítés_terjesztőtől, ez_egy_válasz* – nevük magyarázza jelentésüket. Egy P_i processzor újonnan érkezett, valamely P_j processzor által küldött üzenet után kutat, hogy ismereteket szerezzen szóbeszédekről, meghibásodásokról, valamint más processzorok jelenlegi állapotáról. A kapott $\mathsf{Szóbeszédek}_j$ példányból átmásol minden szóbeszédet a $\mathsf{Szóbeszédek}_i$ tömbbe, abban az esetben, ha az még nincs ott. Beállítja az $\mathsf{Aktív}_i[k]$-t a *meghibásodott* értékre, amennyiben ez az értéke az $\mathsf{Aktív}_j[k]$-nak. Beállítja a $\mathsf{Függő}_i[k]$-t a *kész* értékre, amennyiben ez az értéke a $\mathsf{Függő}_j[k]$-nak. Beállítja a $\mathsf{Függő}_i[j]$-t a *kész* értékre, ha a P_j egy terjesztő és a kapott üzenet egy tartományüzenet. Ha a P_i maga egy terjesztő, akkor $\mathsf{Függő}_i[j]$-t *kész* értékre állítja rögtön azután, miután elküldött P_j-nek egy tartományüzenetet. Hogyha a P_i processzor üzenetet vár a P_j processzortól – például egy gráfüzenetet a kommunikációs gráfban lévő szomszédjától, vagy egy válaszüzenetet –, de nem érkezik üzenet P_j-től, akkor P_i tudja, hogy a P_j processzor meghibásodott, és ekkor azonnal beállítja az $\mathsf{Aktív}_i[j]$ értékét *meghibásodottra*.

ZÁRÓ-FÁZIS

```
    Kód a P_i, 0 ≤ i ≤ n − 1 processzorok esetén
 1  receive üzenetek
```

```
11  helyi számítások elvégzése
12      helyi tömbök frissítése
13      if P_i egy gyűjtő, ami az összes processzorról hallott már
14          then P_i terjesztővé válik
15      a célprocesszorok halmazának meghatározása: for minden P_j processzorra
16          if P_i egy gyűjtő és még nem hallott P_j-ről
17              then küldjön egy érdeklődő üzenetet P_j-nek
18          if P_i egy terjesztő és P_j egy processzor, amely P_i által értesítendő
19              then küldjön egy értesítő üzenetet P_j-nek
20          if egy érdeklődő üzenet érkezett P_j-től e fázis fogadó lépésében
21              then küldjön egy válasz üzenetet P_j-nek

30  send érdeklődő/értesítő/válasz üzenet küldése a célhalmazra vonatkozóan
```

Helyesség

A záró fázis garantálja a helyességet, amint azt a következő állítás mutatja.

23.29. lemma. *Az* Általános-pletyka *algoritmus helyes minden $G(n, f)$ kommunikációs gráfra és Π ütemezéshalmazra.*

Bizonyítás. Az integritás és a kettősségmentesség tulajdonság közvetlenül következik a kódból és a szinkron üzenetküldő rendszerben a többes üzenetküldés szolgáltatásból. Már csak azt kell bizonyítani, hogy minden egyes processzor hallott minden processzorról. Tekintsük az első záró fázisokat közvetlenül megelőző lépést. Ha a P_i processzor nem hallott még valamely más P_j processzorokról, akkor az első záró fázisban az küld P_j-nek egy utolsó remény üzenetet. Erre válasz érkezik a második záró fázisban, hacsak a P_j processzor már meg nem állt. Mindegyik esetben a P_i processzor a harmadik záró fázisban már vagy ismeri a P_j bemenő szóbeszédét, vagy megtudja, hogy a P_j meghibásodott. A negyedik záró fázik gondoskodik annak lehetőségéről, hogy ezeket a P_i által elküldött értesítő üzeneteket megkapják azok a processzorok, amelyeknek a P_i ezeket az üzeneteket elküldte. ■

A $G(n, f)$ kommunikációs gráf, a Π ütemezéshalmaz és a τ megállási küszöb megválasztása hatással van az Általános-pletyka algoritmus speciális alkalmazásának futási idejére és üzenetszámára. Először tekintsük azt az esetet, amikor $G(n, f)$ egy olyan kommunikációs gráf, amely kielégíti a 23.27. definíció $\mathcal{R}(n, f)$ tulajdonságát, Π tartalmaz n véletlen permutációt, és $\tau = c \lg n$ elegendően nagy pozitív c konstans esetén. A 23.28. tételt alkalmazva a következő eredményt kapjuk.

23.30. tétel. *Minden n és $f \leq c \cdot n$ esetén bizonyos $0 \leq c < 1$ konstanshoz létezik olyan $G(n, f)$ gráf, hogy az általános pletyka séma implementációja a $G(n, f)$-fel mint kommunikációs gráffal és véletlenszerű permutációk egy Π halmazával $O(\lg^2 n)$ várható futási idő alatt és $O(n \lg^2 n)$ várható üzenetszámmal valósítja meg a szóbeszéd gyűjtését, ha legfeljebb f a megállások száma.*

Tekintsük az Általános-pletyka algoritmus kis módosítását: a normál fázisban minden P_i processzor küld egy érdeklődő üzenetet az első Δ (egy helyett) processzornak, meg-

felelve a π_i permutációnak, ahol Δ az alkalmazott $G(n, f)$ kommunikációs gráf egy maximális foka. Figyeljük meg, hogy ez nincs hatással az üzenetszám nagyságrendjére, mivel az érdeklődő üzeneteken kívül minden P_i processzor minden egyes normál fázisban küld Δ gráfüzenetet.

23.31. tétel. *Minden n-hez létezik $f \leq n - 1$ és $\tau = O(\lg n)$ paraméter és létezik egy $G(n, f)$ gráf úgy, hogy a módosított* Általános-pletyka *algoritmus megvalósítása a $G(n, f)$-fel mint kommunikációs gráffal és véletlenszerű permutációk egy Π halmazával várt $O(\lg^2 n)$ időn belül és várt $O(n^{1.838})$ üzenetszámmal megvalósítja a pletykát, legyen a megállások száma bármennyi.*

Mivel a fenti tétel Π halmaza a számítás előtt lett kiválasztva, a következő determinisztikus létezési eredményhez jutunk.

23.32. tétel. *Minden n-hez létezik $f \leq n - 1$ és $\tau = O(\lg n)$ paraméter és létezik egy $G(n, f)$ gráf és ütemezések egy Π halmaza, hogy a módosított* Általános-pletyka *algoritmus megvalósítása a $G(n, f)$-fel mint kommunikációs gráffal és a Π ütemezésekkel $O(\lg^2 n)$ időn belül és $O(n^{1.838})$ üzenetszámmal valósítja meg a pletyka gyűjtését, legyen a megállások száma bármennyi.*

Gyakorlatok

23.7-1. Mutassuk meg, hogy a véletlenszerű $G(n, f)$ gráf – amelyben minden csúcs egymástól függetlenül véletlenszerűen választ ki $\frac{n}{n-f} \lg n$ élt önmagától más processzorokhoz – kielégíti a 23.27. definícióban szereplő $\mathcal{R}(n, f)$ tulajdonságot, és amelynek foka legalább $1 - O(1/n)$ valószínűséggel $O(\frac{n}{n-f} \lg n)$.

23.7-2. A vezetőválasztási probléma a következő: minden meghibásodásmentes processzornak választania kell ugyanabban a szinkron lépésben egy meghibásodásmentes processzort. Mutassuk meg, hogy a vezetőválasztás nem oldható meg gyorsabban, mint a pletyka probléma processzormegállással terhelt szinkron üzenetküldő rendszerekben.

23.8. Kölcsönös kizárás közös memóriában

Most bemutatjuk az osztott rendszerek leírására szolgáló második fő modellt, a ***közös memória*** modellt. Az ebben a modellben lévő algoritmusproblémák bemutatásához a kölcsönös kizárás problematikájának megoldásait tárgyaljuk.

23.8.1. Közös memóriájú rendszerek

Feltesszük, hogy a rendszer n processzort $(P_0, \ldots, P_{n-1})$, valamint m regisztert $(R_0, \ldots, R_{m-1})$ tartalmaz. Minden processzor állapotgépként van modellezve. Minden regiszternek van egy típusa, amely meghatározza

1. az általa tárolható értékeket,
2. a rajta végrehajtható műveleteket,
3. a műveletek által visszaküldött értéket (ha van), valamint

4. a regiszternek az egyes műveletek által eredményezett új értékét.

Minden regiszternek lehet kezdőértéke.

Egy egész értékű olvasás/írás regiszter például felvehet bármilyen egész értéket, és rendelkezik az *olvasás*(R, v) és az *írás*(R, v) műveletével. Az olvasás művelete az utolsó, az olvasást megelőző írás v értékét adja vissza, változatlanul hagyva R-et. Az *írás*(R, v) műveletnek van egy v egész paramétere, nem ad vissza értéket, és R értékét v-re cseréli. Konfigurációnak nevezünk egy $C = (q_0, \ldots, q_{n-1}, r_0, \ldots, r_{m-1})$ vektort, ahol a Q_i a P_i processzor állapota, r_j pedig az R_j regiszter egy értéke. Az *események* a számítás processzoroknál történő azon lépései, amelyeknél a következők történnek automatikusan (láthatatlanul):

1. P_i választ egy osztott változót a P_i jelenlegi állapotán alapulva egy adott művelet elvégzéséhez,
2. a megadott művelet az osztott változón kerül végrehajtásra,
3. P_i állapota változik a saját átmenetén alapulóan, a saját jelenlegi állapota és a végrehajtott közös memóriájú művelet visszaküldött értéke alapján.

A konfigurációk és a kezdeti értékkel kezdődő események egy véges sorozatát ***végrehajtási sorozatnak*** hívjuk. Aszinkron közös memóriájú rendszerben egy végtelen végrehajtási sorozat elfogadható, ha végtelen számú számítási lépéssel rendelkezik.

23.8.2. A kölcsönös kizárás problémája

Ebben a problémában a processzorok egy csoportjának olyan osztott erőforrást kell elérnie, amelyet egyidejűleg legfeljebb egy processzor használhat. A megoldásnak a következő tulajdonságokkal kell rendelkeznie.

(1) ***Kölcsönös kizárás:*** Minden processzornak végre kell hajtania egy ***kritikus szakasznak*** nevezett kódszegmenst úgy, hogy bármely adott időpillanatban legfeljebb egy processzor hajthatja végre azt (azaz van a kritikus szakaszban).

(2) ***Holtpontmentesség:*** Ha egy vagy több processzor megkísérel belépni a kritikus szakaszba, valamikor végül egy sikerrel jár, feltéve, hogy egy processzor sem tartózkodik a kritikus szakaszban örökké. Ez a két tulajdonság semmilyen különleges biztosítékot nem nyújt egyik processzor számára sem.

(3) ***Kiéheztetésmentesség:*** A kritikus szakaszba belépni kívánó processzor valamikor végül sikerrel jár, feltéve, hogy egy processzor sem tartózkodik a kritikus szakaszban örökké. Ennek a problémának az eredeti megoldásai olyan speciális szinkronizációs eszközökre támaszkodnak, mint a szemafor vagy a monitor. Néhány olyan *osztott megoldást* mutatunk be, amelyek csak közönséges osztott változókat használnak.

Feltesszük, hogy egy processzor programja az alábbi szakaszokra bomlik:

- **Belépő/Próbálkozó:** a kritikus szakaszba való belépés előkészítéséhez végrehajtott kód.
- **Kritikus:** az egyidejű végrehajtástól megvédendő kód.
- **Kilépő:** a kritikus szakasz elhagyásakor végrehajtott kód.
- **Fennmaradó:** a kód többi része.

Egy processzor a következő sorrenden megy végig ciklikusan ezeken a szakaszokon:

fennmaradó, belépő, kritikus és kilépő. A kritikus szakaszba belépni kívánó processzor először a belépő kódot hajtja végre. Ezt követően, amennyiben sikeres volt, belép a kritikus szakaszba. A processzor a kilépő szakasz futtatásával és a fennmaradó szakaszhoz való visszatéréssel feloldja a kritikus szakaszt. Feltesszük, hogy a processzor akárhányszor végrehajthatja az átmenetet a fennmaradó szakasztól a belépő szakaszig. Ezenkívül a belépő és a kilépő szakaszban elérhető osztott és lokális változók egyike sem érhető el sem a kritikus, sem a fennmaradó szakaszban. Végül is nincs olyan processzor, ami örökké a kritikus szakaszban tartózkodik. Egy közös memóriájú rendszerre vonatkozó algoritmus holtpont nélkül (vagy kizárás nélkül) megoldja a kölcsönös kizárás problémáját, ha teljesülnek a következők:

- **Kölcsönös kizárás:** Minden végrehajtási sorozat minden konfigurációja esetén legfeljebb egy processzor van a kritikus szakaszban.
- **Nincs holtpont:** Minden elfogadható végrehajtásnál, ha egy konfigurációban valamely processzor a belépő szakaszban van, akkor van egy későbbi konfiguráció, amely esetén *valamelyik* processzor a kritikus szakaszban van.
- **Nincs kiéheztetés:** Minden elfogadható végrehajtásnál, ha egy konfigurációban valamely processzor a belépő szakaszban van, akkor van egy későbbi konfiguráció, amely esetén *ugyanaz* a processzor van a kritikus szakaszban.

A kölcsönös kizárás összefüggésében egy végrehajtás ***elfogadható,*** ha minden P_i processzorra P_i vagy végtelen számú lépést tesz meg, vagy P_i a fennmaradó szakaszban fejeződik be. Sőt, nincs processzor a kilépő szakaszba örökre beragadva (akadálymentes kilépési feltétel).

23.8.3. Kölcsönös kizárás hatékony primitívek felhasználásával

Egyetlen bit elegendő a holtpont nélküli kölcsönös kizárás garantálásához egy hatékony tesztelés&beállítás regiszter használatával. A tesztelés&beállítás V változó egy bináris változó, amely két elemi műveletet, a tesztelés&beállítás és a visszaállítás műveletet támogatja, és amely a következő módon van definiálva:

```
tesztelés&beállítás(V : bináris változó) bináris érték visszaadása:
    temp ← V
    V ← 1
    visszatérés (temp)
visszaállítás(V : memóriacím):
    V ← 0
```

A tesztelés&beállítás művelet automatikusan olvassa és frissíti a változót. A visszaállítás művelete egész egyszerűen egy írás. Van egy olyan egyszerű holtpontmentes kölcsönös kizárás algoritmus, amely egy tesztelés&beállítás regisztert használ.

KÖLCSÖNÖS-KIZÁRÁS-TESZTELÉS&BEÁLLÍTÁS-REGISZTERREL

Kezdetben V értéke 0

⟨Belépés⟩:
1 várakozás, amíg tesztelés&beállítás(V) = 0
⟨Kritikus szakasz⟩
⟨Kilépés⟩:
2 visszaállítás(V)
⟨Fennmaradó⟩

Tegyük fel, hogy a V kezdőértéke 0. A belépő szakaszban a p_i processzor ismételten teszteli V-t mindaddig, amíg az 0-t nem ad vissza. Az utolsó ilyen teszt V-hez 1-et rendel, ami azt eredményezi, hogy minden más processzor által végzett teszt 1 értéket ad vissza – eltiltva ezáltal a többi processzort a kritikus szakaszba való lépéstől. A kilépő szakaszban P_i visszaállítja V értékét 0-ra; egy másik, a belépő szakaszban várakozó processzor most már beléphet a kritikus szakaszba.

23.33. tétel. *A tesztelő&beállító regisztert alkalmazó algoritmus holtpontmentes kölcsönös kizárást biztosít.*

23.8.4. Olvasás/írás regisztereket alkalmazó kölcsönös kizárás

Ha nem áll rendelkezésre egy olyan hatékony primitív, mint például a tesztelés&beállítás, akkor a kölcsönös kizárást az olvasás/írás műveletekkel kell megvalósítani.

A PÉKSÉG algoritmus
Lamport kölcsönös kizárásra vonatkozó PÉKSÉG (angolul BAKERY) algoritmusa csak osztott olvasás/írás regisztereket használ fel. Az algoritmus garantálja a kölcsönös kizárást, és nincs kizárás $O(n)$ regisztert használó n processzor esetén (de a regiszteknek szüksége lehet olyan egész értékek tárolására, amelyekre előre nem adható meg felső korlát). előtt).

A kritikus szakaszba belépni kívánó processzorok úgy viselkednek, mint a vásárlók a pékségnél. Mindegyikük kap egy számot, és a legkisebb számot kezében tartó lesz a következő „kiszolgált” vásárló. A sorban nem álló regiszterek a 0 számot kapják, amit nem veszünk legkisebb számként számításba.

Az algoritmus a következő osztott adatszerkezeteket használja: a *Szám*$[0..n-1]$ egy n egész alkotta tömb, melynek i-edik tárolt bejegyzése a P_i processzor jelenlegi száma. A *Választás*$[0..n-1]$ n logikai értéknek egy olyan tömbje, amelyben a *Választás*$[i]$ IGAZ, amikor a P_i számára a szám megkapása éppen folyamatban van. Minden, a kritikus szakaszba belépni akaró P_i processzor próbál olyan számot választani, ami nagyobb minden egyéb processzorénál, és beírja azt a *Szám*$[i]$-be. Ennek megtételéhez a processzorok olvassák a *Szám* tömböt és saját értékükként az olvasott legnagyobb számnál eggyel nagyobb számot választják. Mivel azonban egyidőben számos processzor olvashatja a tömböt, a szimmetria megtörik az i-edik jegyként a (*Szám*$[i]$, i) választásával. Párok lexikografikus rendezését felhasználva egy rendezés van a jegyek sorrendbe rakására meghatározva. A P_i jegyének kiválasztása után addig vár, amíg az a legkisebb nem lesz: Minden más P_j esetén a P_i addig

vár, amíg a P_j be nem fejezi a számválasztást, és ezután összehasonlítja a jegyeiket. Ha a P_j jegye kisebb, P_i addig vár, amíg a P_j végre nem hajtja a kritikus szakaszt, és ki nem lép abból.

PÉKSÉG

Kód a P_i, $0 \le i \le n-1$ processzorok esetén.
Kezdetben $Szám[i] = 0$ és
$Választás[i] =$ HAMIS, $0 \le i \le n-1$ esetén

⟨Belépés⟩:
1 $Választás[i] \leftarrow$ IGAZ
2 $Szám[i] \leftarrow \max(Szám[0],\ldots,Szám[n-1]) + 1$
3 $Választás[i] \leftarrow$ HAMIS
4 **for** $j \leftarrow 1$ **to** $n\ (\neq i)$ **do**
5 **wait until** $Választás[j] =$ HAMIS
6 **wait until** $Szám[j] = 0$ vagy $(Szám[j], j) > (Szám[i], i)$ nem lesz
⟨Kritikus szakasz⟩
⟨Kilépés⟩:
7 $Szám[i] \leftarrow 0$
⟨Fennmaradó⟩

A következő tételek bizonyítását meghagyjuk gyakorlatnak.

23.34. tétel. *A* PÉKSÉG *algoritmus garantálja a kölcsönös kizárást.*

23.35. tétel. *A* PÉKSÉG *algoritmus garantálja a kiéheztetésmentességet.*

Egy korlátos kölcsönös kizárás algoritmus n processzorra

A PÉKSÉG algoritmus tetszőlegesen nagy értékek használatát kívánja meg. A következőkben bemutatunk egy olyan algoritmust, amely megszünteti ezt a követelményt. Ebben a kétprocesszoros algoritmusban a processzorok párosával, irányított fa elrendezésben versenyeznek. Minden párosával folytatott verseny egy teljes bináris fára van rendezve. Minden processzor a fa egy adott leveléhez van hozzárendelve. Egy adott csúcs győztese minden szinten továbbhaladhat a következő magasabb szintre, ahol meg fog küzdeni az ezen csúcs egy másik gyerekéről feljövő győztessel (amennyiben létezik egy ilyen győztes). Annak a processzornak lesz lehetősége belépni a kritikus szakaszba, amelyik végül a gyökérben megnyeri a versenyt.

Legyen $k = \lceil \lg n \rceil - 1$. Tekintsünk egy 2^k levéllel és összesen $2^{k+1}-1$ csúccsal rendelkező teljes bináris fát. A fa csúcsai indukciósan a következő módon sorszámozottak. A gyökér sorszáma 1; az m sorszámú csúcs bal oldali gyerekének sorszáma $2m$, míg a jobboldali gyerekének sorszáma $2m+1$. A fa leveleinek számozása tehát: $2^k, 2^k+1,\ldots, 2^{k+1}-1$.

Minden egyes m csúcshoz három bináris osztott változó tartozik: az $Igény^m[0]$, $Igény^m[1]$, és $Elsőbbség^m$. Minden változónak van egy 0 kezdőértéke. Az algoritmus rekurzív. Az algoritmus kódja egy $Csúcspont(m, oldal)$ eljárásból áll, ami akkor hajtódik végre, amikor a processzor elérte az m csúcsot, miközben az *oldal* processzor szerepét tölti be. Minden csúcsnak van egy kritikus szakasza. Ebbe beleszámít a szülő csúcstól a gyökérig

vezető úton az összes csúcsnál lévő belépő szakasz, az eredeti kritikus szakasz, valamint a kilépő kód a gyökértől a szülő csúcsig vezető út minden csúcsán. Kezdésként a P_i processzor végrehajtja a $(2^k + \lfloor i/2 \rfloor, i \mod 2)$ csúcs kódját.

IRÁNYÍTOTT FA

```
    procedure Csúcs(m, oldal[0..1])
 1  Igény^m[oldal] ← 0
 2  wait until (Igény^m[1 − oldal] = 0 vagy Elsőbbség^m = oldal)
 3  Igény^m[oldal] ← 1
 4  if Elsőbbség^m = 1−oldal then
 5      if Igény^m[1 − oldal] = 1) then menjen az 1 sorra
 6  else várjon addig, amíg Igény^m[1 − oldal] = 0 nem lesz
 7  if v = 1 then
 8      ⟨Kritikus szakasz⟩
 9  else Csúcs(⌊m/2⌋, m mod 2)
10  Elsőbbség^m = 1 − oldal
11  Igény^m[oldal] ← 0
    eljárás vége
```

Az algoritmus korlátos értékeket használ, és amint azt a következő tételek megmutatják, kielégíti a kölcsönös kizárás, kizárásmentességi tulajdonságokat:

23.36. tétel. *Az* IRÁNYÍTOTT-FA *algoritmus garantálja a kölcsönös kizárást.*

Bizonyítás. Tekintsünk egy végrehajtást. A fa leveléhez legközelebb eső csúcstól indulunk ki. Egy processzor akkor lép be ennek a csúcsnak a kritikus szakaszába, ha eléri a 9. sort (elmegy a következő csúcsig). Tegyük fel, hogy annál az m csúcsnál vagyunk, amely kapcsolódik azokhoz a levelekhez, ahonnan P_i és P_j indul. Tegyük fel, hogy ez a két processzor valamely pontnál kritikus szakaszban van. A kódból következik, hogy ennél a pontnál ekkor $Igény^m[0] = Igény^m[1] = 1$. Az általánosság megszorítása nélkül feltehetjük, hogy P_i kritikus szakaszba történő belépése előtti utolsó írása $Igény^m[0]$-be követi a P_j kritikus szakaszba történő belépése előtti $Igény^m[1]$-be történő utolsó írását. Figyeljük meg, hogy P_i be tud lépni az (m-hez tartozó) kritikus szakaszba mind az 5., mind a 6. soron keresztül. P_i mindkét esetben $Igény^m[1] = 0$ értéket olvas. Annak ellenére, hogy P_i $Igény^m[1]$-et olvas, követi P_j $Igény^m[0]$-ba végzett írását. Tehát az, hogy P_i-nek $Igény^m[1]$ olvasásával 1-et kell visszaadnia, egy ellentmondás.

Az állítás a fa leveleire alkalmazott indukcióból következik. ■

23.37. tétel. *Az* IRÁNYÍTOTT-FA *algoritmus garantálja a kiéheztetésmentességet.*

Bizonyítás. Tekintsünk egy elfogadható végrehajtást. Tegyük fel, hogy valamely P_i processzor megállt. Egy afdott időponttól kezdve P_i örökre a belépő szakaszban marad. Most azonban megmutatjuk, hogy P_i nem ragad be örökre az m csúcs belépő szakaszába. Az állítás indukció alapján következik.

1. eset: Tételezzük fel, hogy P_j az $Elsőbbség^m$-nek a 0-ra állításával hajtja végre a 10. sort. Ezek után az $Elsőbbség^m$ már mindig 0 lesz. Tehát P_i áthalad a 2. soron, és az 5.

sorra lép. Így tehát P_i-nek a 6. sorban várakoznia kell, arra kell várni, hogy $Igény^m[1]$ 0 legyen, ami sohasem fog bekövetkezni. P_j így mindig a 3. és a 11. sor közötti sorokat hajtja végre. Mivel azonban P_j nem marad örökké a kritikus szakaszban, ez azt jelentheti, hogy P_j örökre a belépő szakaszba ragad, ami viszont lehetetlen, mivel P_j végrehajtja az 5. sort, és visszaállítja $Igény^m[1]$-et 0-ra.

2. eset: Tételezzük fel, hogy valamely későbbi pontnál P_j sohasem hajtja végre a 10. sort. Így P_j-nek várakoznia kell a 6. sorban vagy a fennmaradó szakaszban. Ha ez a belépő szakaszban van, akkor P_j áthalad a 2. sorban lévő teszten ($Elsőbbség^m$ értéke 1). Így tehát P_i nem éri el a 6. sort. Ekkor P_i $Igny^m[0] = 0$ mellett a 2. sorban várakozik. Így P_j áthalad a 6. sorban lévő teszten. Tehát P_j nem maradhat örökre a belépő szakaszban. Ha P_j örökre a fennmaradó szakaszban marad, a továbbiakban $Igény^m[1]$ egyenlő lesz 0-val. Tehát P_i nem ragadhat be az 2., 5. és 6. sorban, ami ellentmondás.

Az állítás a fa levelein vett indukció alapján következik. ■

Az írás/olvasás regiszterek számára adott alsó korlát

Eddig a bemutatott holtpontmentes kölcsönös kizárás algoritmusok megkövetelték legalább n osztott változó használatát, ahol n a processzorok száma. Mivel lehetséges volt olyan algoritmus kifejlesztése, amely csak korlátos értékeket használt, felmerül a kérdés, hogy van-e mód a használt osztott változók számának csökkentésére.

Burns és Lynch bebizonyították, hogy bármely csak osztott írás/olvasás regisztereket használó holtpontmentes kölcsönös kizárás algoritmusnak szüksége van legalább n osztott változóra, méretüktől függetlenül. Tételük a bizonyítása lehetővé teszi azt, hogy a változók többes-író változók legyenek. Ez azt jelenti, hogy minden processzor írhat minden változóba. Figyeljük meg, hogy ha a változók egyszeres írók, a tétel nyilvánvaló, mivel minden processzornak írnia kell valamit egy (különálló) változóba, mielőtt a kritikus szakaszba lép. Különben a processzor anélkül léphetne be a kritikus szakaszba, hogy bármely másik processzor tudna erről, ami megengedné azt, hogy egyidejűleg tetszőleges másik processzor is beléphessen a kritikus szakaszba, ami ellentmondana a kölcsönös kizárás tulajdonságnak.

A bizonyítás bevezet egy új bizonyítástechnikát, az ***argumentumlefedést.*** Adott egy tetszőleges A holtpontmentes kölcsönös kizárás algoritmus, ez azt mutatja meg, hogy van A-nak valamilyen olyan elérhető konfigurációja, amelyben az n processzor mindegyike azon van, hogy *egyedül* írjon egy osztott változóba. Ezt az osztott változók egy ***lefedésének*** nevezzük. Egy ilyen konfiguráció létezése megmutatható indukció alkalmazásával, és ez kihasználja azt a tényt, hogy a kritikus szakaszba történő belépés előtt minden processzornak írnia kell legalább egy osztott változóba. A bizonyítás megvalósítja az összes osztott változó egy lefedését. A processzor ezután belép a kritikus szakaszba. A lefedést követően az írások azonnal közzé vannak téve, ebből következően egyik processzor sem érzékeli a kritikus szakaszban lévő processzort. Így most egy másik processzor is beléphet egyidejűen a kritikus szakaszba, ami ellentmondás.

23.38. tétel. *Bármely írás/olvasás regisztert alkalmazó holtpontmentes kölcsönös kizárás algoritmusnak használnia kell legalább n osztott változót.*

23.8.5. Lamport gyors kölcsönös kizárás algoritmusa

Minden eddig bemutatott kölcsönös kizárás algoritmusban a processzorok által a kritikus szakaszba történő belépés előtt megtett lépések száma függött n-től, a processzorok számától a verseny (amikor több processzor egyszerre akar belépni a kritikus szakaszba) hiánya esetén is, amikor csak egyetlen processzor van a belépő szakaszban. A legtöbb valós rendszerben azonban a várható küzdelmek száma rendszerint n-nél jóval kisebb.

Egy kölcsönös kizárás algoritmust ***gyorsnak*** nevezünk, ha egy processzor egy konstans számon belüli lépésszámmal lép be a kritikus szakaszba, amikor ez az egyetlen, a kritikus szakaszba belépni próbáló processzor. Figyeljük meg, hogy egy gyors algoritmus többes-író és többes-olvasó változók használatát igényli. Amennyiben csak egyszeres író változókat használna, egy processzornak legalább n számú változót kellene olvasnia.

Az alábbi GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmust Lamport javasolta.

GYORS-KÖLCSÖNÖS-KIZÁRÁS

Kód a P_i processzorra, $0 \leq i \leq n-1$.
Kezdetben a *Gyors-lezárás* és a *Lassú-lezárás* mindegyike 0,
és *Igény*$[i]$ HAMIS minden i, $(0 \leq i \leq n-1)$ esetén

⟨ Belépés ⟩:
1 *Igény*$[i] \leftarrow$ IGAZ
2 *Gyors-lezárás* $\leftarrow i$
3 **if** *Lassú-lezárás* $\neq 0$ **then**
4 *Igény*$[i] \leftarrow$ HAMIS
5 várjon addig, amíg *Lassú-lezárás* $= 0$ nem lesz
6 menj 1-re
7 *Lassú-lezárás* $\leftarrow i$
8 **if** *Gyors-lezárás* $\neq i$ **then**
9 *Igény*$[i] \leftarrow$ HAMIS
10 minden j esetén, várjon addig, amíg *Igény*$[j]$ = HAMIS nem lesz
11 **if** *Lassú-lezárás* $\neq i$ **then**
12 **wait until** *Lassú-lezárás* $= 0$
13 menj 1

⟨Kritikus szakasz⟩
⟨Kilépés⟩:
14 *Lassú lezárás* $\leftarrow 0$
15 *Igény*$[i] \leftarrow$ HAMIS
⟨Fennmaradó⟩

Lamport algoritmusa két mechanizmus kifogástalan kombinációja, ezek közül az egyik a gyors belépés engedélyezése, amikor küzdelem nem érzékelhető, a másik pedig a holtpontmentesség biztosítása küzdelem esetén. A *Gyors-lezárás* és a *Lassú-lezárás* változókat használja a versenynélküli esetekben a hozzáférés vezérlésére. Ezen kívül minden P_i processzorhoz tartozik egy *Igény*$[i]$ logikai változó, amely értéke igaz, ha P_i érdekelt a kritikus szakaszba történő belépésben, egyébként hamis. A processzor akkor tud belépni a kritikus

szakaszban, ha vagy *Gyors-lezárás* = i-t talál – ebben az esetben *gyors útvonalon* lép be a kritikus szakaszba –, vagy *Lassú-lezárás* = i-t talál, amely esetben *lassú útvonal* mentén lép be a kritikus szakaszba.

Tekintsük azt az esetet, amikor nincs processzor sem a kritikus, sem a belépő szakaszban. Ebben az esetben a *Lassú-lezárás* értéke 0, és minden *Igény* bejegyzés 0. Ha ekkor P_i belép a belépő szakaszba, az *Igény*[i]-t 1-re, a *Gyors-lezárást* pedig i-re állítja. Ekkor ellenőrzi a *Lassú-lezárást,* ami 0. Ekkor újra ellenőrzi *Gyors-lezárást,* és mivel nincs másik processzor a belépő szakaszban, beolvassa i-t és belép a kritikus szakaszba három írással és két olvasással rendelkező gyors útvonal mentén.

Ha *Gyors-lezárás* $\neq i$, akkor P_i vár addig, amíg az összes *Igény* jelző vissza nem állítódik. Miután valamely processzor végrehajtja a 10. sor for ciklusát, a *Lassú-lezárás* változatlan marad mindaddig, amíg valamely kritikus szakaszt elhagyó processzor vissza nem állítja azt. Tehát legfeljebb egy P_j processzor találja azt, hogy *Lassú lezárás* = j, és ez a processzor fog lassú útvonal mentén belépni a kritikus szakaszba. Figyeljük meg, hogy a GYORS-KÖLCSÖNÖS KIZÁRÁS algoritmus nem garantálja a kiéheztetésmentességet.

23.39. tétel. *A* GYORS-KÖLCSÖNÖS-KIZÁRÁS *algoritmus holtpontmentes kölcsönös kizárást garantál.*

Gyakorlatok

23.8-1. Egy algoritmus megoldja a ***2-kölcsönös kizárás*** problémáját, ha bármely időpillanatban legfeljebb két processzor van a kritikus szakaszban. Mutassunk be egy algoritmust a 2-kölcsönös kizárás megoldására a tesztelés&beállítás regiszterek használatával.

23.8-2. Bizonyítsuk be, hogy a PÉKSÉG algoritmus kielégíti a kölcsönös kizárás tulajdonságát.

23.8-3. Bizonyítsuk be, hogy a PÉKSÉG algoritmus kiéheztetésmentességét nyújt.

23.8-4. Különítsük el a két processzorra vonatkozó, kizárásmentes korlátos kölcsönös kizárás algoritmust és az irányított fa algoritmust. Mutassuk meg, hogy az előbbi algoritmus rendelkezik a kölcsönös kizárás tulajdonságával.

23.8-5. Bizonyítsuk be, hogy a GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmus rendelkezik a kölcsönös kizárás tulajdonsággal.

23.8-6. Bizonyítsuk be, hogy a GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmus rendelkezik a holtpontmentesség tulajdonsággal.

23.8-7. Mutassuk meg, hogy a GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmus nem elégíti ki kiéheztetésmentesség tulajdonságát, azaz alkossunk meg egy olyan végrehajtást, amelyben egy processzor ki van zárva a kritikus szakaszból.

Feladatok

23-1. Az* ELÁRASZTÁS *algoritmus üzeneteinek a száma

Adott egy G gráf n csúccsal és e éllel. Bizonyítsuk be, hogy bármely végrehajtási sorozat esetén az ELÁRASZTÁS algoritmus $O(e)$ üzenetet küld. Mi az üzenetek pontos száma a csúcsok és az élek számának függvényében?

23-2. Vezetőválasztás gyűrűben
Tegyük fel, hogy az üzeneteket csak az óramutató járásával egyező irányban lehet küldeni. Tervezzünk egy aszinkron, $O(n \lg n)$ üzenetszámú algoritmust a gyűrűbeli vezető megválasztására. *Útmutatás.* A processzorok dolgozzanak fázisokban. Minden processzor *aktív módban* kezdjen egy, a processzor azonosítójának megfelelő *értékkel,* s bizonyos feltételek teljesülése mellett lépjen *továbbító* módba, ahol csak továbbítja az üzeneteket. Egy aktív processzor két másik aktív processzortól vár üzenetet, megvizsgálja az azok által küldött értékeket, és eldönti, hogy vezető lesz-e, aktív marad-e és elfogadja az egyik *értéket,* vagy továbbító módba vált. Határozzuk meg, hogy mi alapján kell a döntéseket meghozni úgy, hogy ha három vagy több aktív processzor van, akkor legalább az egyik aktív marad, illetve, hogy függetlenül attól, hogy milyen értékekkel rendelkeznek az aktív processzorok egy fázisban, legalább a processzorok fele maradjon aktív a következő fázisban.

23-3. A* GYORS-KÖLCSÖNÖS-KIZÁRÁS *algoritmus elemzése
Alkossuk meg a GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmus egy olyan végrehajtási sorozatát, amelyben két processzor van a belépő szakaszban, és mindkettő $\Omega(n)$ változót olvas a kritikus szakaszba történő belépése előtt.

23-4. Egyetértés érvényessége aszinkron rendszerekben
Mutassuk meg, hogy az érvényességi feltétel ekvivalens azzal a követelménnyel, hogy minden hibamentes processzor döntése valamely processzor bemenete legyen.

23-5. Egyetlen forrású egyetértés
Az egyetértési probléma egy változata megköveteli, hogy egy kijelölt processzor *(a tábornok)* bemenő értékét eljutassák az összes többi processzorhoz *(a hadnagyokhoz).* Ez a probléma ***egyetlen forrású egyetértés*** néven is ismert. A következő feltételeknek kell teljesülniük:

- **Megállás**: Minden hibamentes hadnagy valamikor végül dönt.
- **Megegyezés**: Az összes hibamentes hadnagy ugyanarra a döntésre jut.
- **Érvényesség**: Ha a tábornok hibamentes, akkor a közös eredmény a tábornok bemenete lesz.

Tehát, ha a tábornok hibamentes, akkor a hibamentes processzoroknak nem kell dönteniük a tábornok bemenetéről, de még mindig egyetértésre kell jutniuk egymással. Tekintsünk egy bizánci hibákkal rendelkező szinkron üzenetküldő rendszert. Mutassuk meg, hogyan lehet a 23.4.5. pontban leírt egyetértési probléma egy megoldását a tábornok probléma egy megoldásává át- és visszaalakítani. Mik a transzformáció üzenet- és menetszámbeli többletköltségei?

23-6. Banki tranzakciók
K épzeljük el, hogy n bank összeköttetésben áll egymással. Az i-edik bank m_i összegű pénzzel rendelkezik. A bankok nem emlékeznek a kezdőösszegre. A bankok pénzösszegeket utalnak egymás között <10> alakú üzenetekben, melyek az átutalt összeget reprezentálják. Adott időpontban az egyik bank úgy dönt, hogy megállapítja a rendszerben lévő pénz összegét. Tervezzünk egy algoritmust, ami kiszámolja az $m_1 + \cdots + m_n$ összeget úgy, hogy közben a pénzügyi tranzakciók nem állnak le.

Megjegyzések a fejezethez

Az osztott rendszerek definícióját Attiya és Welch könyvéből [26] vettük át. Az üzenetküldő rendszerek meghibásodás nélküli modelljét Attiya, Dwork, Lynch és Stockmeyer cikke [25] alapján tárgyaljuk.

A rendszer egyes processzorait – Lynch és Fischer cikkét követve – egy automatával (amely véges és végtelen is lehet) modellezzük [237].

A végrehajtási sorozat fogalma Fischer, Gries, Lamport és Owicki cikkeiből [237, 260, 261] származik.

Az *aszinkron rendszer* definícióját Awerbuch [27], valamint Peterson és Fischer [276] cikkeiből vettük át.

A Feszítőfa-üzenetszóró algoritmust Segall cikke [307] alapján ismertettük.

A szinkron és aszinkron rendszerekben is alkalmazható Bully algoritmust Hector-Garcia Molina javasolta 1982-ben [126]. Az ismertetett aszinkron vezetőválasztási algoritmus aszimptotikusan optimalitását Burns [58] bizonyította be.

A *két tábornok problémáját* Gray könyve [146] alapján tárgyaltuk.

Az *egyetértési problémát* először Lamport, Pease és Shostak [222, 267] vizsgálta.

Az Egyetértés-megállási-hibáknál algoritmus Dolev és Strong [93] cikkén alapul. A 23.14. tételhez hasonló állítás bizáci hibákra Fischer és Lynch [108] eredménye. Az idézett, megállásos hibákra vonatkozó tétel Dolev és Strong [93] cikkéből származik.

Az egyetértési problémát bizánci hibájú processzok esetén megoldó algoritmust Berman és Garay [40] javasolta.

Az osztott algoritmusok elméletének egyik alapvető eredménye, hogy aszinkron rendszerekben – megbízható kommunikáció mellett – az egyetértési probléma már akkor sem oldható meg, ha egyetlen processzor meghibásodhat. Ezt az eredményt először Fischer, Lynch és Paterson [109] cikke mutatta be.

Leslie Lamportnak a kölcsönös kizárásra vonatkozó Pékség algoritmusa [220] egy korai, klasszikus példája az olyan algoritmusnak, amely csak osztott olvasás/írás regisztereket használ fel. A Pékség algoritmus tetszőlegesen nagy értékek használatát kívánja meg. Ezt a követelményt Peterson és Fischer [276] küszöbölte ki.

Először Burns és Lynch [59] mutatta meg, hogy bármely – csak osztott írás/olvasás regisztereket használó, holtpontmentes kölcsönös kizárást biztosító – algoritmusnak használnia kell legalább n osztott változót, méretüktől függetlenül.

A fejezetben tárgyalt Gyors-kölcsönös-kizárás algoritmust Leslie Lamporttól [221] származik.

A 23-3., 23-4. és a 23-5. feladatok forrása Attiya és Welch könyve [26].

Az osztott algoritmusok témakörét összefoglaló mű Gerard Tel [334] és Nancy Ann Lynch [236] könyve – utóbbi magyarul is megjelent. Több osztott algoritmust (például a kölcsönös kizárás biztosítására) tárgyal Kozma László és Varga László könyve [212]. Sok hasznos ismeretet tartalmaz konkrét rendszerekről Tanenbaum és van Steen [332] könyve, amely szintén elérhető magyarul is.

24. Petri-hálók alkalmazása elosztott programok vizsgálatára

A Petri-hálók szemléletes gráf-modellek, amelyek párhuzamos folyamatok leírására, modellezésére, elemzésére alkalmasak. A modellezett folyamatok lehetnek kémiai, fizikai, gyártási, irányítási, társadalmi folyamatok vagy éppen párhuzamos, elosztott algoritmusok is. Petri-hálókkal különböző absztrakciós szinten fogalmazhatunk meg modelleket, részletekben gazdagabb leíráshoz általában nagyobb háló tartozik.

A fejezet első részében a Petri-hálókkal kapcsolatos alapfogalmakat vezetjük be. Ezt követően a Petri-hálókkal kifejezetten azzal a céllal foglalkozunk, hogy elosztott algoritmusok, programok tulajdonságait igazoljuk. Ezen cél eléréséhez a fejezet második részében a Petri-háló általánosan használt definícióját kiegészítjük, bevezetjük a Petri-dobozok fogalmát.

A modellezett rendszerek, például elosztott programok tulajdonságait oly módon is igazolhatjuk, hogy a nekik megfelelő Petri-hálót vizsgáljuk. Petri-hálók segítségével elsősorban a program egyes folyamatai vagy folyamatrészei közötti sorrendi, függőségi és szinkronizációs kapcsolatokat, az információ áramlását, a kommunikációt, a konfliktus-, illetve a konkurrenciahelyzeteket elemezhetjük. Egyidejűleg az egyes értékadások, függvényhívások jelentésétől elvonatkoztatunk. Ha két különböző rendszert modellező Petri-háló lényegében azonos, akkor a két rendszer kommunikációs, szinkronizációs struktúrája is megegyezik az adott absztrakciós szinten. Ebben az esetben hasonló dinamikus viselkedésre számíthatunk még akkor is, ha az egyik társadalmi folyamatokat modellez a másik pedig egy számítógépes hálózat protokolljának felépítését írja le.

Egy program szövegében könnyen azonosíthatók a szinkronizációs és kommunikációs elemek, így az algoritmus működését modellező háló akár programmal is előállítható és az elemzés is automatizálható. Állapot- vagy adatfolyamdiagramból is származtathatunk Petri-hálót. A háló vizsgálata alapján kapott eredmények felhasználásához azonban azt is tudnunk kell, hogy a háló egyes részei mely programrésznek vagy adatelemnek feleltek meg eredetileg. Ennek érdekében a hálók egyes elemeit a programszövegre utaló címkékkel láthatjuk el. Programok tervezésére, tulajdonságaik vizsgálatára készített modellek egyik legfontosabb jellemzője, hogy tudunk-e programrészek tulajdonságaiból az összetett rendszer tulajdonságaira következtetni. Ahhoz, hogy elemi programoknak megfelelő hálókból összetett algoritmusokat leíró hálókat állítsunk elő, a programkonstrukcióknak megfelelő kompozíciós műveleteket is definiálnunk kell a Petri-hálók felett.

24.1. Alapfogalmak

A Petri-háló fogalmát páros gráfok segítségével definiálhatjuk. Páros gráfnak nevezzük azt a gráfot, amelyben a csúcsok két diszjunkt halmazba sorolhatóak oly módon, hogy az azonos halmazba tartozó csúcsok között nem vezet él. A Petri-háló egy olyan rendezett pár, amelynek első eleme magát a hálót definiáló irányított gráfot adja meg, míg második eleme a kezdősúlyozást. A gráfban kétféle csúcspont van: hely, illetve átmenet. A helyek feltételeket jelölnek, az átmenetek eseményeket, például értékadásokat, összetett utasításokat, eljárásokat vagy függvényhívásokat. Az élek az átmenetekkel reprezentált események előfeltételeit ábrázoló helyeket kötik össze az átmenetekkel, illetve az átmeneteket az utófeltételeknek megfelelő helyekkel. Minden élhez tartozik egy súlyérték is, ez szükségtelenné teszi a többszörös élek használatát. Ha például egy él súlya kettő, akkor az él két egyszeres élnek felel meg. A helyekhez is rendelünk súlyokat, amelyekkel az elő-, illetve utófeltételek teljesülését reprezentáljuk. A helyekhez tartozó súlyértékek a Petri-háló működése során az egyes átmenetek hatására dinamikusan változnak. A Petri-háló állapotát azzal írjuk le, hogy megadjuk az egyes helyeken lévő súlyok számát.

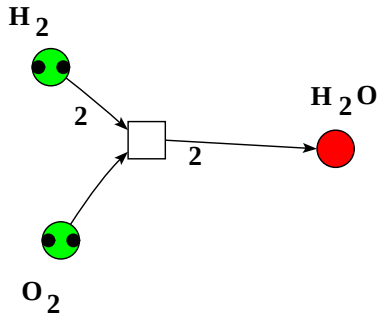
24.1. definíció (Petri-háló). *A **Petri-háló** egy (N, M_0) rendezett pár, ahol:*

- $N = (P, T, R, v)$ *a tartógráf, irányított, páros gráf, amelynek az élei súlyozottak,*
- P, T *: a csúcsok véges halmazai. P a helyek halmaza, T az átmenetek halmaza,*
- $P \cup T \neq \emptyset$*,*
- $P \cap T = \emptyset$*,*
- R *– az éleket megadó reláció:* $R \subseteq (P \times T) \cup (T \times P)$*,*
- $v : R \mapsto \mathbb{N}_0$ *- az élek súlyait megadó függvény,*
- *A helyek kezdeti súlyozását (kezdőállapot) M_0 adja meg:* $M_0 : P \to \mathbb{N}_0$*.*
  A helyek súlyozását (állapot) általában rendezett n-esként adjuk meg, például $M_0 = (1, 0, \ldots, 2)$*.*

Jelölések:
Petri-hálók tulajdonságainak tömör leírásához az alábbi jelöléseket vezetjük be:

- a p helyet megelőző átmenetek halmaza:
  ${}^\bullet p ::= R^{(-1)}(p)$, (a p hely R élreláció szerinti inverz képe),
- a t átmenetet megelőző, bemenő helyek halmaza:
  ${}^\bullet t ::= R^{(-1)}(t)$, (a t átmenet R élreláció szerinti inverz képe),
- adott csúcs utóda(i): $p^\bullet ::= R(p), t^\bullet ::= R(t)$.

A helyeket körrel, az átmeneteket négyzettel jelöljük. A 24.1. ábrán egy egyszerű Petri-hálót láthatunk, amely a víz oxigén- és hidrogénmolekulákból történő szintézisének folyamatát modellezi. A háló három helyet és egy átmenetet tartalmaz. Az átmenet két vízmolekula $(2 * H_2O)$ szintézisét reprezentálja, melynek előfeltétele, hogy legalább két hidrogénmolekula $(2 * H_2)$ és legalább egy oxigénmolekula (O_2) álljon rendelkezésre. Az ábrán látható élsúlyozás ezt fejezik ki, az átmenethez tartozó H_2 címkéjű helytől kettő súlyú él vezet az átmenethez. Az esemény bekövetkezésének egyik előfeltétele, hogy ezen a bemenő helyen legalább két súly (két hidrogénmolekula) jelenléte szükséges. Ha egy él súlyát nem



24.1. ábra. Víz szintézise.

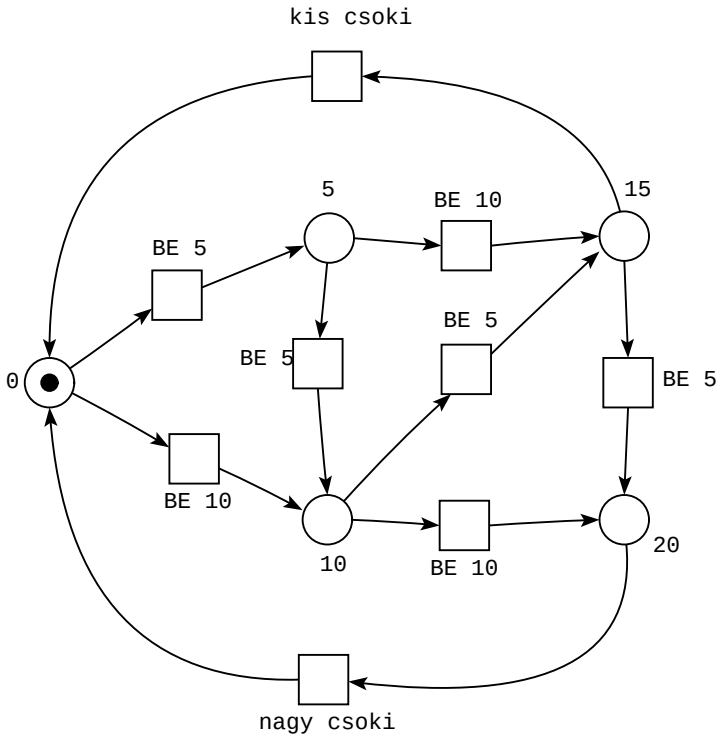
jelöljük, akkor az megállapodás szerint egy. Ilyen él vezet az O_2 címkéjű helytől az átmenethez. Ez azt jelenti, hogy az átmenet másik előfeltétele szerint legalább egy súly szükséges az alsó bemenő helyen is. Az ábrán látható esetben az átmenet előfeltételei teljesülnek, az átmenet tüzelhet. Az átmenet tüzelésének hatására a bemenő helyekről az onnan vezető élek súlyának megfelelő számú súly lekerül, a kimenő helyekre pedig az oda vezető élek súlyával egyenlő számú súly hozzáadódik.

Egy átmenethez tartozó esemény előfeltétele akkor teljesül, ha a gráfban az átmenetet megelőző helyeken kellő számú súly helyezkedik el. A Petri-háló működési szabálya adja meg, hogy egy átmenet mikor tüzelhet és hogyan változik a háló állapota. Többféle működési szabály is megadható. Jelöljük M-mel a Petri-háló aktuális állapotát leíró, illetve M'-vel a tüzelés után előálló állapotot megadó súlyvektort. $M(p)$ a p helyen lévő súlyok száma. $v(p,t)$ a p helyről t átmenethez, $v(t,p)$ pedig a t átmenettől a p helyhez vezető él súlya. Az él t tüzelése esetén éppen ennyi súlyt távolít el, illetve ad hozzá a p helyhez.

24.2. definíció (működési szabály (alapértelmezés)).

1. *t átmenet **tüzelhet** (aktivizálható, megengedett), ha* $\forall p \in {}^\bullet t : M(p) \geq v(p,t)$.
2. *t átmenet tüzelése után az új állapotot leíró M' súlyozás:*
   $\forall p \in P : M'(p) = M(p) + v(t,p) - v(p,t)$.

Általában nem teljesül, hogy a tüzelés során a háló különböző helyein lévő súlyok számának összege megmarad. Előfordulhat, hogy az átmenet több vagy kevesebb súlyt távolít el a bemenő helyekről, mint amennyit a kimenő helyeken elhelyez. Ha egy átmenetnek nincs bemenő helye, akkor az átmenet előfeltétele folyamatosan teljesül, tetszés szerinti időpontban tüzelhet, ú.n. ***forrás átmenet***: ${}^\bullet t = \emptyset$. Ha egy átmenetnek nincs kimenő helye, akkor a tüzelés során csak a bemenő helyeken csökken a súlyok száma, sehol nem jelenik meg azonban új súly. Az ilyen átmeneteket ***nyelő átmenetnek*** nevezzük: $t^\bullet = \emptyset$. Ugyanaz a hely lehet egyszerre bemenő és kimenő helye is ugyanazon átmenetnek, ilyenkor ***hurokról*** beszélünk: $(p,t) : p \in {}^\bullet t \wedge p \in t^\bullet$. Egy háló ***átlátszó***, ha hurokmentes. ***Normális*** hálónak nevezzük azokat a Petri-hálókat, amelyekben minden behúzott él súlya pontosan egy: $\forall r \in R : v(r) \leq 1$.

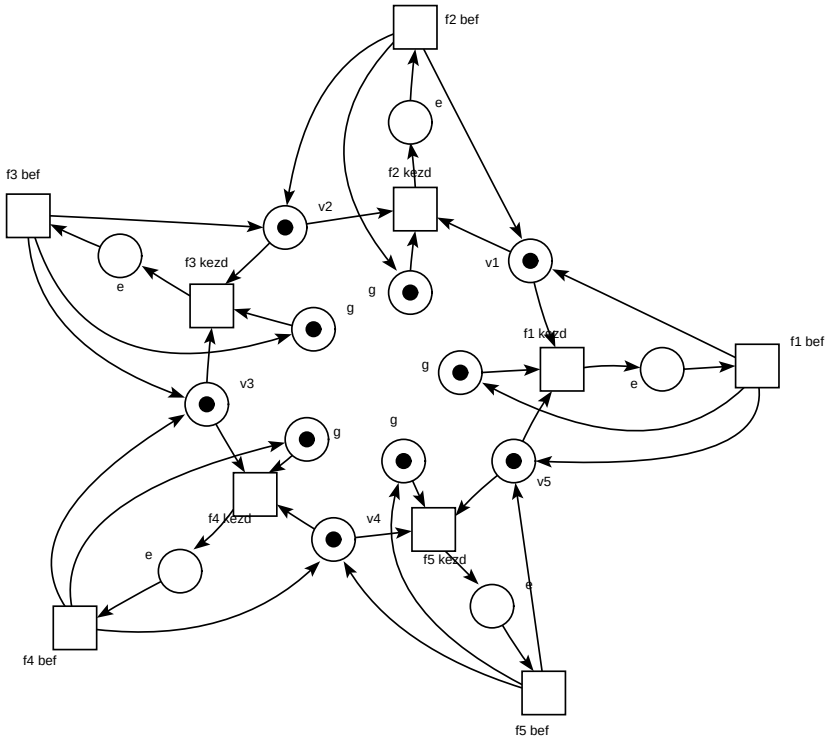


24.2. ábra. Csoki automata.

Akciósorozatnak nevezzük a háló által végrehajtott tüzelésekben szereplő átmenetek azonosítóinak sorozatát. Például: $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, \ldots$.

A 24.2. ábrán egy csoki automata működésének modelljét mutatjuk be. Kezdőállapotban az automata nyílásában 0 Ft értékű érme van, ezt a helyzetet mutatja az az állapot, amikor csak a 0-val jelölt helyen van súly. Két esemény előfeltétele is teljesül, ennek megfelelően a 0 helyet követő „BE 10”, illetve a „BE 5” eseménnyel címkézett átmenet is tüzelhet. Egy átmenet azonban nem feltétlenül tüzel akkor, ha az előfeltétele teljesül, külső feltételektől is függhet, hogy a tüzelésre sor kerül-e és mikor. Ha a fogyasztó egy 5 Ft-os érmét dob be, akkor a „BE 5” jelű átmenet tüzel, ha 10 Ft-os érme kerül a gépbe, akkor a „BE 10” jelű átmenet megy végbe. A Petri-háló működése nem-determinisztikus, a modell nem írja le, hogy a döntés mitől függ.[1] Az első esetben a súly a 0-val jelölt helyről az 5-tel, a másik esetben pedig a 10-zel jelölt helyre kerül át. Figyeljük meg, hogy ugyanazon esemény (pl. „BE 10”) egyes előfordulásait meg tudjuk különböztetni egymástól oly módon, hogy különböző átmenet tartozik hozzá. Ha a rendszer eljut a 15-tel jelölt állapotba, akkor a fogyasztó ismét választhat, hogy egy kis csokit kér, vagy egy újabb 5 Ft-os érme bedobása után egy nagy csokit ehet meg. Végül a gép visszatér alapállapotába.

[1] Egyes kiterjesztett modellek megengedik tiltó feltételek, valószínűségi értékek, illetve prioritások megadását is a nem-determinisztikus választás befolyásolására.



24.3. ábra. Étkező filozófusok.

Következő példánk E. W. Dijkstrától származik, aki az operációs rendszerek tárgy tanítása során a folyamatok közötti kölcsönös kizáráson alapuló erőforrásmegosztás elvét szemléltette vele. Ez a példa öt, egymással együttműködő párhuzamos folyamatból álló rendszert modellez. A történet szerint egy asztal körül öt filozófus ül, akik egy nagy közös tálból vehetnek maguknak makarónit. Ahhoz, hogy a makaróni a tányérba kerüljön, két villára van szükség. A tálat többen is használhatják egyszerre, de az étkezéshez szükséges villák mindegyikére külön-külön teljesülnie kell, hogy egyszerre csak egy filozófus használatában lehetnek. Minden filozófus számára két villa érhető el, ezek azonban olyan villák, amelyeket a szomszéd filozófusok is szeretnének használni. Ha egy filozófus felemeli az asztalról bal és jobb oldali villáját és eszik, akkor egyik szomszédja sem rendelkezhet az étkezéshez szükséges két villával.

A csillagalakú 24.3. ábrán az egyes filozófusokat egy-egy ág reprezentálja. Az egyes sorszámú filozófust például a jobb oldali vízszintes ággal modellezzük. Kezdőállapotban minden filozófus gondolkodik, a filozófus állapotát leíró súly a „g” jelű helyen van. Kezdetben az összes villa az asztalon fekszik, a „v1”, . . . , „v5” jelű helyek mindegyike tartalmaz egy-egy súlyt. Az „f1 kezd” jelű átmenet előfeltételei teljesülnek, az első filozófus elkezdhet enni oly módon, hogy mindkét szomszédos villát felveszi. Ha az „f1 kezd” átmenet tüzel, akkor mindhárom bemenő helyéről eltűnnek a súlyok és az egyes sorszámú filozó-

fus „e” jelű, étkezését jelző helyén jelenik meg egy súly. Ebben az állapotban az „f2”-vel és „f5”-tel jelölt átmenetek nem tüzelhetnek, az előfeltételeik közül egy-egy hamis, egy-egy villa hiányzik az átmenet végrehajtásához. Az első filozófus étkezésének befejeztekor az „f1 bef” jelű átmenet tüzel és visszateszi a súlyokat a kiinduló helyzetnek megfelelő helyekre. Figyeljük meg, hogy csak egymással nem szomszédos filozófusok étkezhetnek egyidejűleg. Ez az egyszerű modell kizárja holtpont kialakulását azzal, hogy a két villa felvételét egyetlen atomi eseménybe sűríti. Nem alakulhat ki holtpont oly módon, hogy minden filozófus felemeli a bal oldali villáját és a végtelenségig arra várakozik, hogy a jobb oldali villa is megszerezhető legyen. Egyes filozófusok kiéheztetése azonban előfordulhat, hiszen semmi sem akadályozza meg azt, hogy egy filozófust szomszédai konfliktus helyzetben mindig megelőzzenek a villák megszerzésében.

Figyeljük meg, hogy az étkező filozófusokat bemutató hálóban a súlyok elhelyezkedése egyrészt az egyes filozófusok lokális állapotait is és egyúttal a teljes rendszer globális állapotát is meghatározza!

Gyakorlatok

24.1-1. Egy juhász át szeretne kelni a folyón egy kecskével, egy farkassal és egy fej káposztával. Egyszerre csak egy dolgot vihet magával a csónakjában a túlsó partra. Egyik parton sem maradhat kettesben a kecske és a farkas, illetve a kecske és a káposzta. Készítsük el az átkelés modelljét Petri-háló segítségével.

24.1-2. Módosítsuk a csoki automata működését leíró hálót oly módon, hogy kis vagy nagy csoki választása mellett a pénz visszakérésének lehetőségét is választhassa a fogyasztó.

24.1-3. Készítsünk az étkező filozófusok feladatához egy finomabb modellt, amelyben a filozófus a két villát nem egyszerre, hanem egymás után veheti fel.

24.2. Kapacitáskorlát

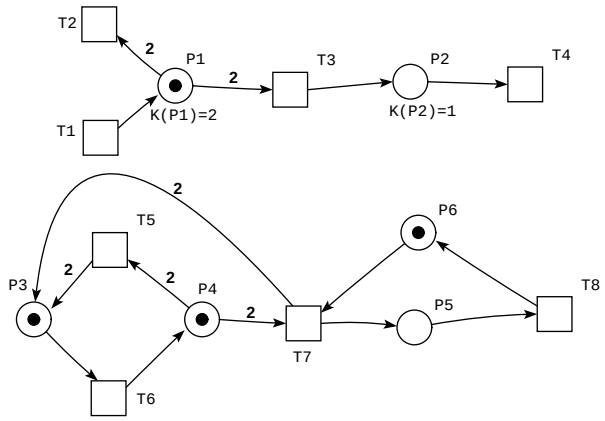
A csoki automata és az étkező filozófusok Petri-hálójára egyaránt igaz volt, hogy legfeljebb egy súly jelent meg egy-egy helyen. Elosztott programok és protokollok Petri-hálókkal történő modellezése során gyakran teljesül az, hogy egy-egy helyen egynél több súly egyszerre sohasem lehet. Ezen feltétel azonban nem minden hálóra teljesülne automatikusan. Ha biztosítani akarjuk, hogy a háló adott p helyén a súlyok száma sohasem haladhat meg egy előre megadott $k(p)$ kapacitásértéket, akkor ezt a működési szabály megváltoztatásával könnyen elérhetjük.

A szigorú működési szabály a kimenő helyeket is figyelembe veszi a tüzelés előfeltételének meghatározásakor. Ha a tüzelés hatására a kimenő helyek valamelyikén a súlyok száma meghaladja az arra a helyre érvényes kapacitáskorlátot, akkor a tüzelés nem engedélyezett.

24.3. definíció (szigorú működési szabály). *Legyen $k : P \mapsto \mathbb{N}_0 \cup \{\infty\}$ egy kapacitáskorlátot definiáló függvény.*

- *1/a. t átmenet tüzelhet (aktivizálható, megengedett), ha*
  $\forall p \in {}^\bullet t : M(p) \geq v(p,t)$ *és* $\forall p \in t^\bullet : M''(p) \leq k(p)$, *ahol*
  $M''(p) = M(p) + v(t,p) - v(p,t)$.

A tüzelés után az új állapot megegyezik a 24.2. definíció 2. pontjában meghatározottal.



24.4. ábra. Korlátos kapacitású helyek.

Egy ettől eltérő módszer szerint az átmenet mindenképp tüzelhet, de a kimenő hely kapacitáskorlátját meghaladó súlyok elvesznek.

24.4. definíció (gyenge működési szabály).

- *2/a. t átmenet tüzelése után az új M' súlyozás:* $\forall p \in P$: $M'(p) = \min(k(p), M(p) + v(t,p) - v(p,t))$.

Az átmenetek pontosan akkor tüzelhetnek, amikor azt a 24.2. definíció 1. pontja megengedte.

24.2.1. Korlátos kapacitású helyek kiküszöbölése

Számunkra a szigorú működési szabály érdekes elsősorban, mert a számítógépek erőforrásainak végessége miatt szükséges lehet algoritmusok modellezésekor egyes helyekre kapacitáskorlátot meghatározni. Megmutatható azonban, hogy minden kapacitáskorláttal rendelkező és a szigorú működési szabályt használó háló helyettesíthető egy, az eredeti működési szabályt alkalmazó hálóval. A két háló abban az értelemben ekvivalens, hogy pontosan ugyanazok az akciósorozatok lehetségesek mindkettőben.

24.5. definíció (komplemens hely transzformáció).

1. $\forall p \in P : k(p) < \infty$ *helyhez bevezetünk egy komplemens* p' *helyet úgy, hogy* $M'_0(p') = k(p) - M_0(p)$.
   Jelöljük a komplemens helyek halmazát P'*-vel.*

2. $\forall p \in P', t \in T$ *párhoz bevezetünk egy új élet úgy, hogy* $v(t,p') = v(p,t)$ *és* $v(p',t) = v(t,p)$.

A 24.4. ábrán egy példát láthatunk komplemens helyek bevezetésére.

A komplemens helyeket az átmenetekkel összekötő új éleket éppen úgy definiáltuk, hogy azok az eredeti tüzelési szabály mellett pontosan annyi súlyt visznek egy átmenet

tüzelésekor a komplemens helyre, amennyit az átmenet elvesz az eredeti helyről, illetve pontosan annyi súlyt vesznek el a komplemens helyről, amennyit az átmenet az eredetire tesz. A komplemens helyek bevezetése transzformáció tehát azt eredményezi, hogy a hely és a hozzá tartozó új komplemens hely súlyösszege a háló működése során állandó, a komplemens hely a szabad kapacitást tartja nyilván. A háló invariáns tulajdonsága, hogy a hely és a komplemens hely összege éppen a megadott kapacitáskorlát (helyinvariáns). Mivel helyek súlyértéke mindig nemnegatív, ezért sem az eredeti helyen, sem a komplemens helyen nem lehet sohasem több súly, mint a kapacitáskorlát. Az eredeti tüzelési szabály alkalmazásával működő háló a kimenő helyeknek megfelelő komplemens bemenő helyek használatával akadályozza meg a kapacitáskorlát túllépését. Nem aktivizálható egy átmenet a transzformációval kapott hálóban, ha nincs a bemenő komplemens helyen súly, azaz az eredeti kimenő helyen a súlyok száma elérte a kapacitáskorlátot. Belátható a következő tétel:

24.6. tétel (korlátos kapacitású helyek kiküszöbölése). *(N, M_0) átlátszó, korlátos kapacitású háló, szigorú működési szabállyal. (N', M_0') az (N, M_0)-ból komplemens hely transzformációval kapott háló. Ekkor (N, M_0) és (N', M_0') hálók ekvivalensek, a lehetséges akciósorozataik megegyeznek.*

Gyakorlatok

24.2-1. Mutassunk példát három olyan átmenetsorozatra, amely eltér egymástól abban az esetben, ha ugyanarra a hálóra az eredeti, a szigorú, illetve a gyenge működési szabályt alkalmazzuk.

24.2-2. Mutassuk meg, hogy a komplemens hely transzformációval készített háló mindig tüzelhet az alapértelmezés szerint érvényes működési szabálynak megfelelően, ha az eredeti háló tüzelhet a szigorú szabály szerint.

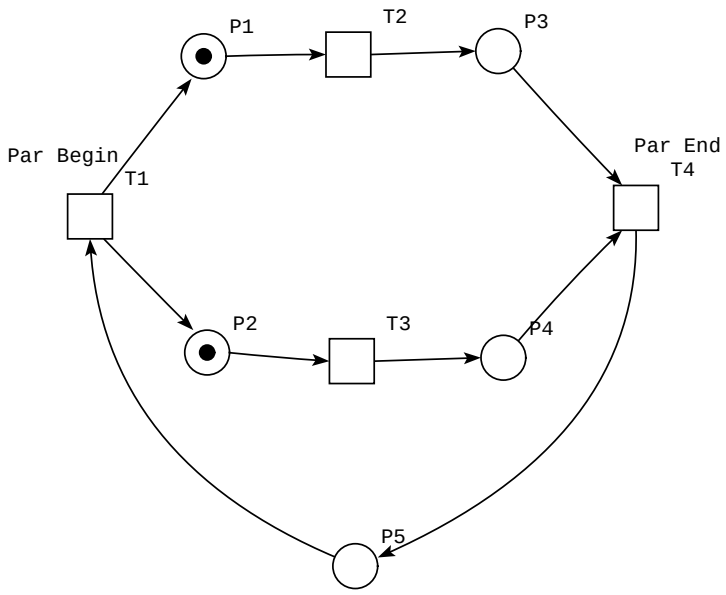
24.3. Párhuzamos folyamatok együttműködése

Párhuzamos és elosztott programok esetén a folyamatok közötti kapcsolatok leírásakor gyakran találkozunk néhány jellegzetes struktúrával. Az egyik ilyen példa a `Par Begin` `Par End` programszerkezet (24.5. ábra), amelyet Dijkstra vezetett be és széles körben használnak párhuzamos algoritmusok pszeudokóddal történő leírásakor.

Ebben az esetben két alapelem együttes megjelenését figyelhetjük meg. A T_2-vel és T_3-mal jelölt átmenetek előfeltételei egyaránt a „Par Begin” címkéjű átmenettől függenek, a feltételek egyidejűleg teljesülnek. A T_2 és T_3 átmenetek egymástól függetlenül, aszinkron módon futó folyamatokat modelleznek, melyek viszonya a konkurencia szóval jellemezhető, azaz egymás működését nem befolyásolhatják. Ezzel szemben a T_4 átmenet csak a két folyamat szinkronizációja után tüzelhet, a P_3 és a P_4 helyen is szüksége van egy-egy súlyra.

A 24.6. ábra egy egyszerű kommunikációs protokollt mutat be. A bal oldali irányított kör a szinkron kommunikációt kezdeményező folyamatot modellezi. A *Küld* átmenet hatására egy üzenet kerül a pufferbe, amelyet a jobb oldali irányított körrel reprezentált folyamat kiolvas, majd az üzenet fogadását nyugtázza. A küldő folyamat bevárja a nyugtát és ezután folytatja tevékenységét.

A 24.7. ábra bal oldali részén két folyamat, T_1 és T_2 konfliktushelyzetét figyelhetjük meg. Az előfeltételüket reprezentáló P_1 helyen csak egy súly található, amelyre azonban



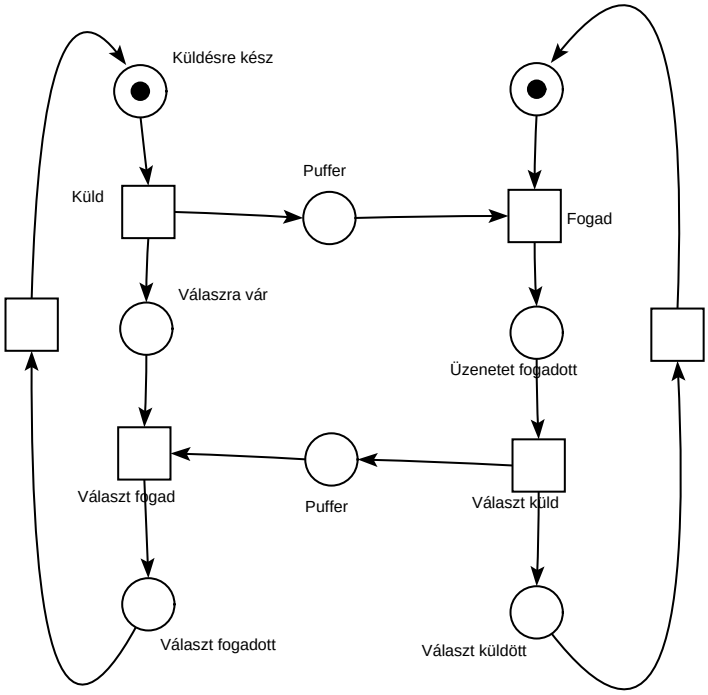
24.5. ábra. Konkurencia és szinkronizáció.

mindkettőnek szüksége lenne a tüzeléshez. Ezt a helyzetet már megfigyelhettük az étkező filozófusok esetében is, ahol a villa használatakor jelentkezett konfliktus.

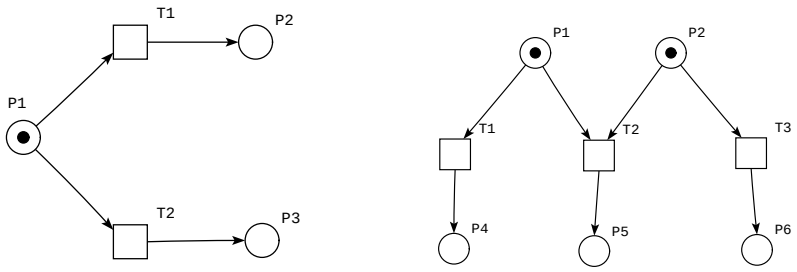
Bonyolultabb esetekben a konfliktus és a konkurencia egyszerre figyelhető meg. A 24.7. ábra bal oldali részén a T_1 és T_2 átmenetek konkurensek, de a T_2 és T_3 átmenetek konfliktusban vannak. Ezt a helyzetet ***zavarnak*** hívjuk. Megosztva használt adatok esetén egyidejűleg többen is olvashatják az adatokat, de egyidejű írás és olvasás, vagy egyidejűleg több folyamat által végzett írási műveletek nem megengedettek. Ezt a helyzetet modellezi a 24.8. ábrán látható Petri-háló. A modell egyidejűleg öt olvasó folyamatnak vagy egyetlen egy író folyamatnak ad lehetőséget a kritikus szakaszba történő belépésre, az adatokhoz való hozzáférésre. A P_1 hely a várakozó folyamatok számát, a P_5 pedig a szemafor értékét mutatja. Írási művelethez a T_1 átmenet tüzelésére van szükség, ennek előfeltétele, hogy a P_2 helyen mind az öt súly jelen legyen. Olvasási műveletet a T_2 átmenet tüzelése reprezentál, amelyhez a P_2 helyről egyetlen súly is elegendő. P_4 mutatja a kritikus szakaszban tartózkodó olvasó folyamatok számát.

A 24.9. ábrán az Ada programozási nyelvből ismert randevú időhöz kötött szelektív várakozásának modelljét figyelhetjük meg. A modell három folyamat együttműködését mutatja be, a T_1 és T_2 átmeneteket tartalmazó körrel modellezett folyamat randevúzik T_3, T_4, T_7 vagy a T_6, T_5, T_8 átmeneteket tartalmazó folyamatok egyikével. Randevú akkor jöhet létre, ha a T_4, illetve a T_5 átmenetek bemenő helyein egyidejűleg súly jelenik meg. Ha mindhárom partner kész a randevúra, akkor konfliktushelyzet áll elő, mert a P_2 helyen lévő egyetlen súlyra mindkét randevúhoz szükség lenne, de egyszerre csak az egyikhez használható fel.

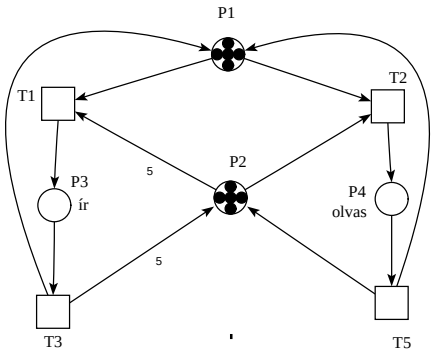
A 24.10. ábra egy adatfolyam segítségével megfogalmazott számítás Petri-hálós modelljét mutatja be. Adatfolyam modell esetén a számítási folyamat adatvezérelt, azaz ha az



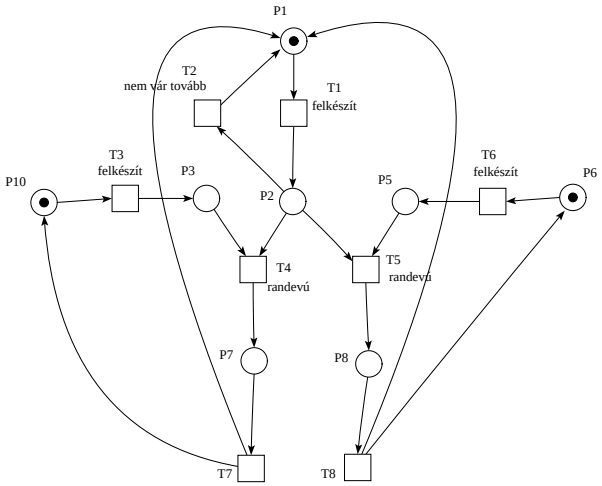
24.6. ábra. Egyszerű protokoll.



24.7. ábra. **a)** konfliktus **b)** zavar.



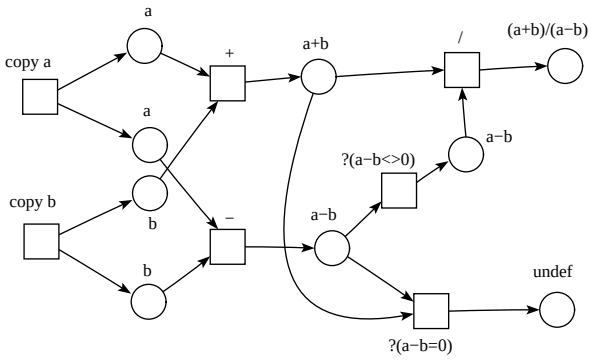
24.8. ábra. Író-olvasó szinkronizáció kölcsönös kizárással.



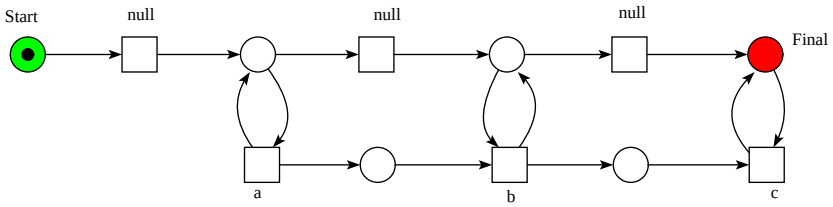
24.9. ábra. Szelektív, időhöz kötött várakozás.

átmenetekhez tartozó függvények argumentumai a bemenő helyeken megjelennek, akkor az átmenet tüzelésének eredményeként a függvény értéke a kimenő helyre kerül. Az ábrán az $(a + b)/(a - b)$ kifejezés értékének kiszámítását követhetjük nyomon.

Minden véges állapotú gép leírható Petri-hálóval, így Petri-hálókkal is modellezhetjük formális nyelvek felismerési és generálási szabályait oly módon, hogy egyes átmenetekhez szimbólumokat rendelünk hozzá és azok az akciósorozatok felelnek meg a nyelvhez tartozó szavaknak, amelyek végén a háló nem tartalmaz súlyt. Példaként bemutatjuk az $a^n b^n c^n$, $n \in N$ szavakat elfogadó Petri-hálót.



24.10. ábra. Adatfolyamszámítás.



24.11. ábra. Nyelvelfogadás.

Gyakorlatok

24.3-1. Terjesszük ki a Petri-hálókat a tiltó élek bevezetésével. A tiltó éleket tartalmazó Petri-hálók tüzelési szabálya a következőképpen módosul: ha egy helytől egy átmenethez tiltó él vezet, akkor az átmenet csak akkor tüzelhet, ha a hely súlyozatlan. (A tiltó élek növelik a Petri-hálók kifejezőerejét. A tiltó éleket is tartalmazó modell Turing-teljes.) Tiltó élek berajzolásával egészítsük ki az Ada randevút modellező Petri-hálót oly módon, hogy a T_1, T_2 körrel leírt folyamat csak akkor kerülhesse el a randevút, ha egyik partnere sem kész arra.

24.3-2. Készítsük el annak a rendszernek a modelljét, amelyben két termelő-fogyasztó folyamatpár működik és a második fogyasztó csak akkor fogadhat üzenetet, ha az első fogyasztó puffere üres (prioritásos termelő-fogyasztók).

24.4. Viselkedési tulajdonságok

Ebben az alfejezetben a Petri-hálók dinamikus viselkedésének olyan tulajdonságait vizsgáljuk, melyek szerkezetük mellett függnek kezdősúlyozásuktól is.

A tulajdonságok vizsgálatához az akciósorozat fogalmát használjuk. Párhuzamos programok tulajdonságainak vizsgálatához általában önmagában nem elegendő sem a program által érintett állapotok sorozatának ismerete, sem csak az egymás után tüzelő átmeneteket

tartalmazó akciósorozat ismerete. Ezért az akciósorozatot gyakran kiterjesztjük oly módon, hogy abban az átmeneteken kívül az állapotokat leíró súlyvektorokat is feltüntetjük.

24.4.1. Jelölések

Tegyük fel, hogy az M_0 súlyozásból a ς akciósorozattal elérhető az M_n súlyozás: $M_0\,[\varsigma> M_n$, ahol $\varsigma = t_1, t_2, \ldots, t_n$ akciósorozat.

Az összes érintett állapotot és átmenetet tartalmazó kiterjesztett akciósorozat jelölése: $M_0\,[t_1> M_1\,[t_2> M_2 \ldots M_{n-1}\,[t_n> M_n$.

$L(N, M)$ – azon akciósorozatok halmaza, amely az N-ben elérhető az M súlyozásból.

$R(N, M)$ az N hálóban az M súlyozásból elérhető súlyozások halmaza, az M ún. továbbsúlyozó osztálya. $R(N, M) = \{M' | \exists \varsigma \in L(N, M) : M\,[\varsigma> M'\}$. $R(N, M)$-et röviden $R(M)$-mel vagy $\overrightarrow{M}$-mel jelöljük, ha N egyértelműen adott.

Jelölje $\#(\varsigma, t)$ a t átmenet előfordulásainak számát ς-ban.

Az a kérdés eldönthető, hogy egy adott súlyozás elérhető-e, (azaz $M \in R(N, M_0)$ teljesül-e). Az azonban, hogy két adott háló ekvivalens-e az akciósorozatokra nézve, $(L(N_1, M_{1,0}) = L(N_2, M_{2,0}))$ általában **nem** eldönthető.

24.7. definíció (k-korlátosság, biztonságosság). *Az (N, M_0) Petri-háló **k-korlátos,** ha $\forall M \in R(N, M_0) : \forall p \in P : M(p) \leq k$. Egy Petri-háló **biztonságos,** ha 1-korlátos.*

Egy k-korlátos Petri-háló esetében a korlát univerzális, azaz minden helyre egységesen ugyanaz a felső korlát érvényes, eltérően a kapacitáskorláttal rendelkező hálóktól, ahol a kapacitáskorlát a hely függvényében változhat.

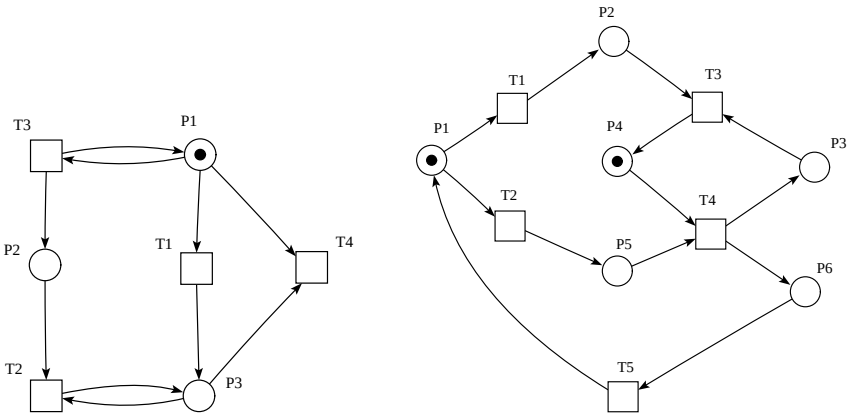
24.8. definíció (elevenség). *Legyen $t \in T$. M_0 kezdősúlyozástól függően az N Petri-hálóban a t átmenet*

- $\mathbf{L}_0$ ***szinten eleven*** *(holt), ha $\forall \varsigma \in L(N, M_0) : t \notin \varsigma$,*
- $\mathbf{L}_1$ ***szinten eleven*** *(aktivizálható), ha $\exists \varsigma \in L(N, M_0) : t \in \varsigma$,*
- $\mathbf{L}_2$ ***szinten eleven,*** *ha $\forall k \in \mathbb{N} : \exists \varsigma \in L(N, M_0) : \#(\varsigma, t) \geq k$,*
- $\mathbf{L}_3$ ***szinten eleven,*** *ha $\exists \varsigma \in L(N, M_0) : \#(\varsigma, t) = \infty$,*
- $\mathbf{L}_4$ ***szinten eleven,*** *ha $\forall M \in R(M_0)$-ra a t L_1 szinten eleven.*

24.9. tétel. *Ha egy t átmenet L_4 szinten eleven, akkor L_3 szinten is az.*

A négyes szintű elevenség szerint minden elérhető állapot után is létezik olyan akciósorozat, hogy abban újra előfordul az átmenet. Ez azt jelenti, hogy meg tudunk konstruálni egy olyan akciósorozatot, amelyben végtelen sokszor fordul elő. A tétel megfordítására ellenpélda a 24.12. ábra bal oldali részén látható Petri-háló T_3 átmenete.

24.10. tétel. *Ha egy t átmenet L_3 szinten eleven, akkor L_2 szinten is az.*



24.12. ábra. a) L_1, L_2 és L_3 eleven átmenetekkel rendelkező háló. (Lehetséges akciósorozatok: t_1; t_3! (végtelen sokszor t_3); $t_3(k)$,$t_2(k)$ (ahol k tetszőleges egész szám). **b)** biztonságos, szigorúan L_1 eleven háló. (Lehetséges akciósorozatok: t_1; t_2,t_4,t_5,t_2; t_2,t_4,t_5,t_1,t_3.)

Ha létezik egy olyan akciósorozat, amelyben az átmenet végtelen sokszor fordul elő, akkor tetszőleges k természetes számhoz tudunk mutatni olyan sorozatot, amelyben legalább k-szor fordul elő az átmenet. A tétel megfordítására ellenpélda a 24.12. ábra bal oldalán látható Petri-háló T_2 átmenete.

24.11. tétel. *Ha egy t átmenet L_2 szinten eleven, akkor L_1 szinten is az.*

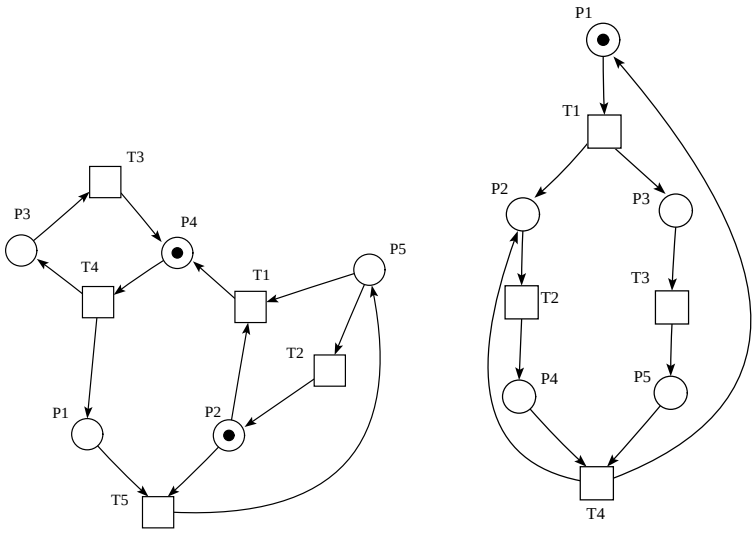
Ha tetszőleges k természetes számhoz tudunk mutatni olyan sorozatot, amelyben legalább k-szor fordul elő az átmenet, akkor ($k = 1$)-re is létezik ilyen. A tétel megfordítására ellenpélda a 24.12. ábra bal oldali részén látható Petri-háló T_1 átmenete.

24.12. tétel. *Egy átmenet L_1 szinten akkor és csak akkor eleven, ha nem holt.*

24.13. definíció. *Azt mondjuk, hogy egy átmenet szigorúan L_k eleven, ha L_k szinten eleven, de nem eleven L_{k+1} szinten. Egy (N, M_0) Petri-hálót L_k elevennek mondunk, ha $\forall t \in T : t$ L_k eleven.*

24.14. definíció (visszatérési képesség, otthonállapot). *Azt mondjuk, hogy egy Petri-háló rendelkezik a* ***visszatérési képességgel,*** *ha minden elérhető állapotból létezik olyan akciósorozat, amellyel a kezdőállapot elérhető. Formálisan: (N,M_0) Petri-háló esetén $\forall M \in R(M_0) : M_0 \in R(M)$.*
Otthonállapotról *beszélünk, ha van egy olyan, nem feltétlenül a kezdőállapottal azonos állapot, amelybe a háló tetszőleges elérhető állapotból el tud jutni. Formálisan: M' otthonállapot, ha $\forall M \in R(M_0) : M' \in R(M)$.*

Megvizsgálhatjuk, hogy egy tetszőlegesen megválasztott súlyozás lefedhető-e, azaz található-e olyan elérhető állapot, amelyben minden helyen legalább ekkora súly van.



24.13. ábra. a) nem korlátos (P_1 hely), nem eleven (T_1 átmenet), visszatérési képességgel rendelkező háló. **b)** nem korlátos (P_2 hely), eleven, perzisztens, visszatérési képességgel nem rendelkező háló.

24.15. definíció (lefedhetőség). (N, M_0) *Petri-háló esetén az* M *súlyozás* ***lefedhető,*** *ha* $\exists M' \in R(M_0) : \forall p \in P : M'(p) \geq M(p)$.

Egy tetszőleges $t \in T$ átmenetre legyen M az a minimális súlyozás, amely mellett t végrehajtható. Ekkor t L_1 szinten akkor és csak akkor eleven, ha M lefedhető, illetve akkor és csak akkor holt, ha M nem lefedhető.

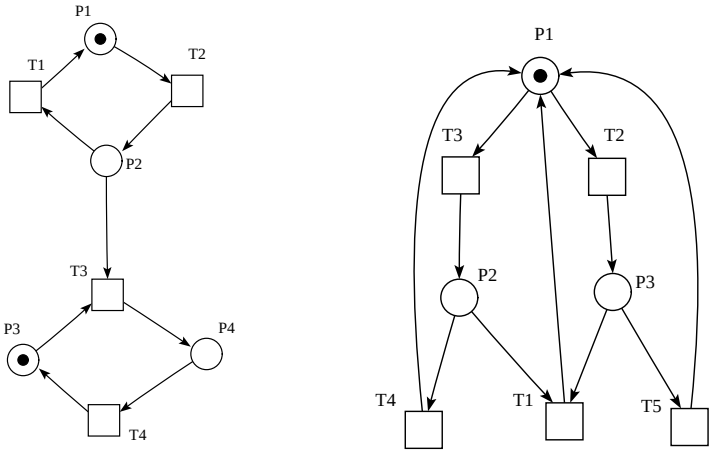
24.16. definíció (perzisztencia). *Az* (N,M_0) *Petri-hálót* ***perzisztensnek*** *nevezzük, ha tetszőleges* $t_1, t_2 \in T$ *állapotpárra teljesül, hogy ha* t_1 *és* t_2 *is engedélyezett (aktivizálható), akkor (közvetlenül)* t_1 *aktivizálása után* t_2 *továbbra is engedélyezett (aktivizálható) marad.*

Ha egy perzisztens hálóban egy átmenet aktivizálhatóvá válik, akkor mindaddig az marad, amíg végre nem hajtódik.

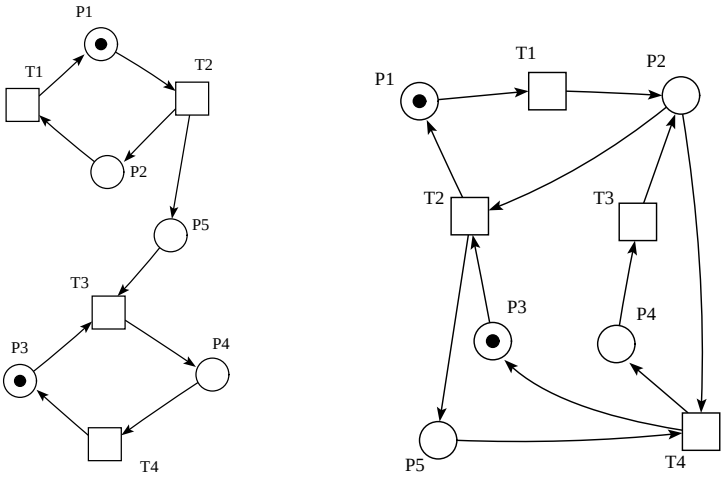
A 24.13.–24.16. példák segítségével megmutatjuk, hogy az elevenség, k-korlátosság, visszatérési képesség egymástól független tulajdonságok.

Petri-hálók átmeneteinek ok-okozati függetlenségének mértékéről ad tájékoztatást a szinkronizációs távolság. Meghatározásakor azt vizsgáljuk, hogy a két átmenethalmaz elemeinek előfordulásainak különbsége legfeljebb mekkora lehet a hálóhoz tartozó (véges) akciósorozatokban. Az előfordulások számának legnagyobb eltérése nem feltétlenül a kezdőállapotból induló akciósorozatok esetén figyelhető meg, így a szinkronizációs távolság kiszámításakor az összes elérhető állapotból induló akciósorozatot figyelembe kell venni.

Szinkronizációs távolságot speciális esetben két átmenetre is megadhatunk.



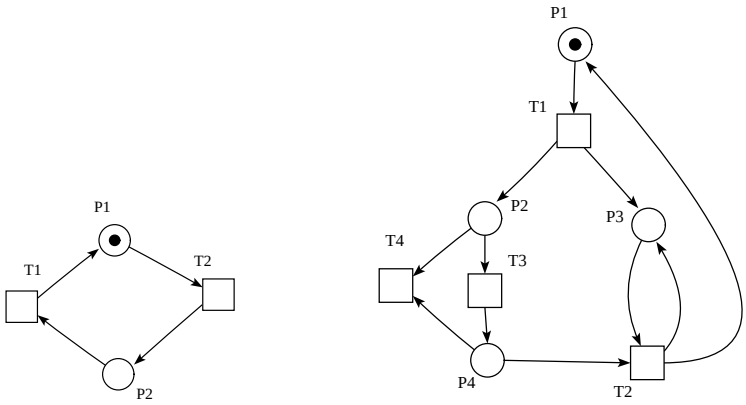
24.14. ábra. a) 1-korlátos (biztonságos), nem eleven, visszatérési képességgel nem rendelkező háló. (Lehetséges akciósorozatok: (t_1, t_2)(k),t_3,t_4.) **b)** 1-korlátos (biztonságos), nem eleven (T_1 átmenet), visszatérési képességgel rendelkező háló.



24.15. ábra. a) Példa nem korlátos (P_5), eleven, visszatérési képességgel rendelkező hálóra. **b)** Példa 1-korlátos (biztonságos), eleven, visszatérési képességgel nem rendelkező hálóra. (Lehetséges akciósorozatok: t_1, t_2,(t_1, t_4,t_3,t_2)!.)

24.17. definíció (szinkronizációs távolság). $(N, M_0) : t_1, t_2 \in T$ *esetén*
$d_{1,2} := \max_{\varsigma \in L(N,M), M \in R(M_0)} |\#(\varsigma, t_1) - \#(\varsigma, t_2)|$.

24.18. definíció (szinkronizációs távolság (2)). $(N, M_0) : E_1, E_2 \subseteq T$ *esetén*
$d_{E_1,E_2} := \max_{\varsigma \in L(N,M), M \in R(M_0)} |\#(\varsigma, E_1) - \#(\varsigma, E_2)|$.



24.16. ábra. a) Példa 1-korlátos (biztonságos), eleven, visszatérési képességgel rendelkező hálóra. **b)** Példa nem korlátos (P_3), nem eleven (T_4), visszatérési képességgel nem rendelkező hálóra. (Lehetséges akciósorozatok: $(t_1,t_3,t_2)!$.)

A szinkronizációs távolságot megjeleníthetjük grafikusan új helyek bevezetésével is. Ezek a helyek csak illusztrációként szolgálnak, szervesen nem kapcsolódnak a háló működéséhez. Az illusztrációs hely rákövetkező átmenetei aktivizálhatóak lehetnek abban az esetben is, ha a helyen nincs súly. Egy ilyen hely az egyik átmenethalmazhoz tartozó tüzelések számának a másik átmenethalmazhoz tartozó tüzelések számához viszonyított többletét jeleníti meg. Az egyik halmaz átmenetei tüzeléskor elhelyeznek egy súlyt az új helyen, a másik halmaz átmenetei pedig elvesznek egy súlyt erről a helyről (ha van rajta súly). Mivel egy-egy ilyen hely csak az egyik irányú többletet jeleníti meg, két adott átmenethalmaz szinkronizációs távolságának meghatározásához két új helyet kell bevezetnünk. A két helyen megjelenő legnagyobb súlyértékek maximuma adja a szinkronizációs távolságot.

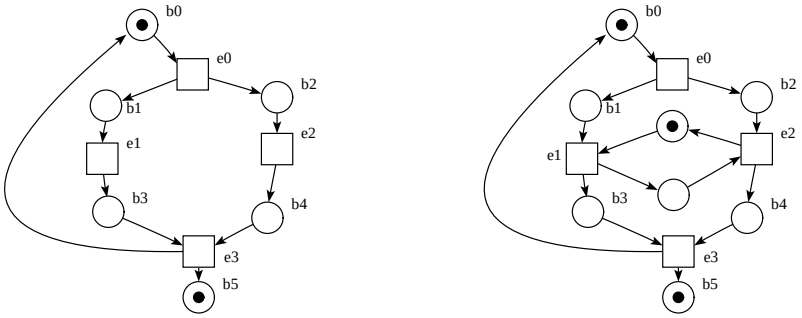
Például az E_1 és E_2 átmenethalmazok szinkronizációs távolságának mérésére bevezethetjük az $s_{1,2}$ és $s_{2,1}$ helyeket, ahol ${}^\bullet s_{1,2} = E_1, s_{1,2}^\bullet = E_2$, illetve ${}^\bullet s_{2,1} = E_2, s_{2,1}^\bullet = E_1$.

A pártatlanság fogalma általában az események ütemezéséhez kapcsolódik és csak végtelen működés esetén van értelme vizsgálni. A Petri-háló a működési szabállyal együtt egy absztrakt végrehajtási modellt határoz meg, így a szokásos értelemben nem beszélhetünk pártatlanságról.

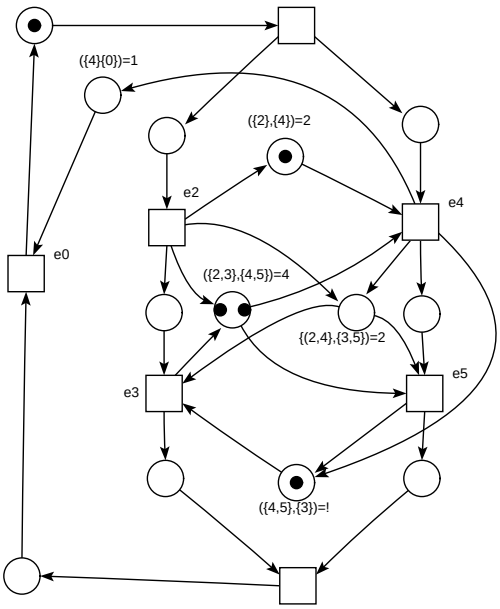
Megvizsgálhatjuk azonban, hogy a Petri-háló által megengedett végtelen akciósorozatokban két átmenet egymáshoz való viszonya milyen. Előfordulhat-e az valamely akciósorozatban, hogy az egyik átmenet akár végtelen sokszor is előfordul, míg a másik egyszer sem?

24.19. definíció (pártatlanság).

1. *Egy háló **korlátosan pártatlan** t_i, t_j-re nézve, ha bármely akciósorozatban t_i előfordulásainak száma felülről korlátos t_j következő előfordulásáig és viszont.*
2. *A háló **korlátosan pártatlan**, ha tetszőleges $t_i, t_j \in T$ átmenetpárra nézve a háló korlátosan pártatlan.*



24.17. ábra. a) Az e_1 és e_2 átmenetek szinkronizációs távolsága 2 ($d(1,2) = 2$). (Pl. e_1,e_2,e_3,e_0,e_2 akciósorozat.) **b)** Az e_1 és e_2 átmenetek szinkronizációs távolsága 1 ($d(1,2) = 1$).



24.18. ábra. Példa különböző szinkronizációs távolságokra.

Megfogalmazhatunk pártatlansági követelményt az egyes akciósorozatokra is. Egy akciósorozat szigorúan pártatlan, ha véges vagy minden átmenet végtelen sokszor fordul elő benne.

24.20. definíció (szigorú pártatlanság).

1. $\forall M \in R(M_0) : \forall \varsigma \in L(N, M)$ *esetén* ς ***feltétlenül szigorúan pártatlan***, *ha* $\forall t_j \in T : \#(\varsigma, t_j) = \infty$ *vagy* ς *véges.*
2. (N, M_0) *Petri-háló* ***feltétlenül szigorúan pártatlan***, *ha* $\forall M \in R(M_0) : \forall \varsigma \in L(N, M) :$ ς *feltétlenül szigorúan pártatlan.*

Petri-hálók pártatlanságát tehát kétféleképpen is definiáltuk. A 24.16. ábra bal oldali részén látható hálóra mindkét pártatlansági követelmény teljesül, a 24.14. ábra bal oldali hálójára egyik sem. A két követelmény azonban csak akkor ekvivalens, ha a háló korlátos (24.7. def.).

24.21. tétel. *Egy korlátosan pártatlan háló szigorúan pártatlan is.*

Ha bármely két átmenetre teljesül, hogy az egyik csak véges sokszor fordulhat elő úgy, hogy a másik nem fordul elő, akkor egy végtelen átmenetsorozatban minden átmenet végtelen sokszor fordul elő. A tétel megfordítása nem igaz. Előfordulhat két átmenet egy végtelen sorozatban végtelen sokszor úgy, hogy az egyik mindig eggyel többször fordul elő a másik következő előfordulásáig. Ezt az esetet mutatja be a 24.13. ábra jobb oldalán látható Petri-háló. Ebben a hálóban T_2 véges, de nem korlátos sokszor hajtható végre a T_3 következő előfordulásáig, attól függően, hogy hány súly halmozódott fel már a P_2 helyen.

24.22. tétel. *Ha egy háló korlátos és szigorúan pártatlan, akkor korlátosan pártatlan is.*

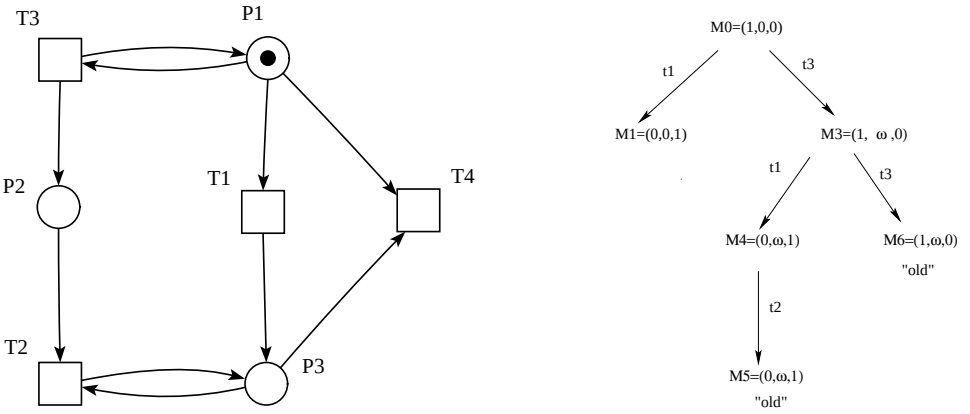
Gyakorlatok

24.4-1. Létezik-e olyan Petri-háló, amelyben t_1 és t_2 egy korlátosan pártatlan átmenetpár, szinkronizációs távolságuk mégsem véges szám?

24.5. Petri-hálók vizsgálata

Többféleképpen is vizsgálhatjuk Petri-hálók tulajdonságait. Az egyik lehetőség, hogy előállítjuk a háló elérhető állapotait leíró gráfot. Elkészíthetjük a háló fedési fáját, illetve gráfját, amelyben a csúcspontokat állapotokkal, az éleket átmenetekkel címkézzük. A gyökér címkéje a kezdőállapot. A csúcspontokat összekötő irányított élek a megengedett átmeneteknek felelnek meg.

Elindulunk a gyökérből és a megengedett átmenetek segítségével rendre új élekkel és csúcspontokkal bővítjük a fát. Az összes elérhető állapotot tartalmazó gráf azonban végtelen nagy is lehet, ha a háló nem korlátos (24.7. definíció). Elérhetőségi fa helyett ezért a fedési fát készítjük el, amelyben a súlyok ciklikus növekedését kizárjuk. Ha egy olyan csúcspontot találunk, amelyhez tartozó állapot már szerepel a gyökérből hozzá vezető úton, akkor ezt a csúcspontot már nem terjesztjük ki. Ha egy új csúcspont címkéjében szereplő súlyozás fedi a gyökérből hozzá vezető út egyik csúcspontjának súlyozását, de különbözik attól, akkor az új csúcs súlyvektorát módosítjuk. Azokra pozíciókra, ahol az új csúcshoz tartozó súlyérték nagyobb, a valódi súlyérték helyett a végtelen nagy súlyt jelző ω jelet tesszük. Az elérhető állapotok közül nem jelenik meg mindegyik a fában, de minden elérhető állapothoz található azt fedő súlyvektor.



24.19. ábra. I. példa fedési fára.

24.5.1. Elérhetőségi és fedési fa

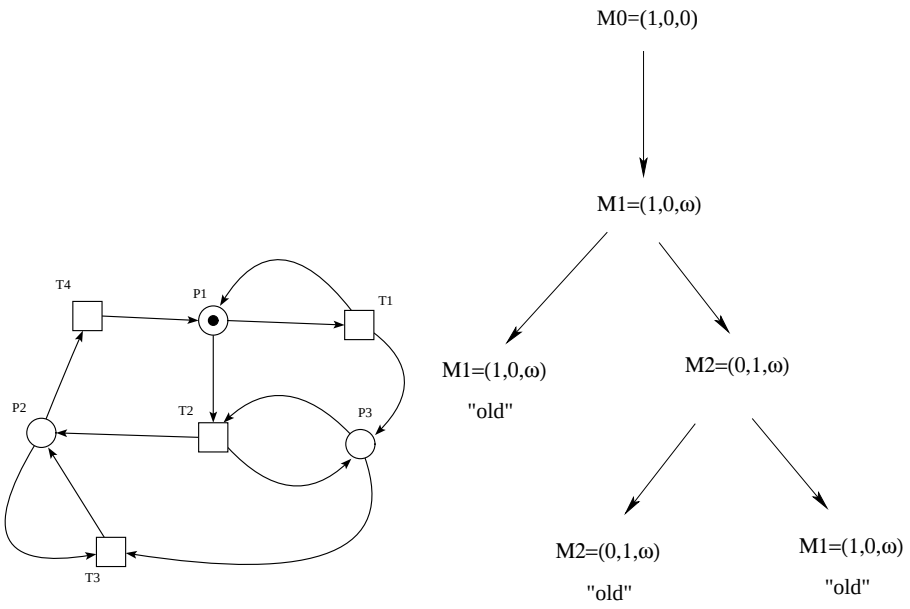
24.23. definíció (elérhetőségi fa). *Az (N, M_0) Petri-háló elérhetőségi (nem korlátos háló esetén fedési) fája olyan gráf, amelyben a csúcsok súlyozásokkal vannak címkézve, az élek pedig átmenetekkel és ezt a gráfot az alábbi módon állítjuk elő:*

1. *új* $:= \{M_0\}$
2. *ciklus, amíg új* $\neq \emptyset$
   (a) $M :\in$ *új*, *új* $:=$ *új* $\setminus \{M\}$
   (b) *ha M-ig a gyökértől létezik már M címkéjű csúcs* $\Rightarrow$ *M régi.*
   (c) *ha M-ben nincs megengedett átmenet* $\Rightarrow$ *M zsákutca.*
   (d) *M-ben* $\forall t$ *megengedett átmenetre kiszámoljuk* $M\,[t\rangle\, M'$*-t.*
      i. *Ha a gyökértől M-ig* $\exists M''$: M''*-t lefedi* M'*-t és* $M' \neq M''$*, akkor minden* $p \in M'$: $M'(p) > M''(p)$ *helyre:* $M'(p) := \omega$.
      ii. M' *új csúcs: új* $:=$ *új* $\cup \{M'\}$*, az él címkéje t lesz.*

A fedési fa alkalmas arra, hogy a háló korlátosságát vizsgáljuk és arra is, hogy eldöntsük, hogy egy átmenet holt-e.

24.24. tétel. *Legyen G a (N, M_0) Petri-háló elérhetőségi fája.*
Korlátos a Petri-háló akkor és csak akkor, ha nincs G-ben ω címkéjű csúcs.
1-korlátos (biztonságos) a Petri-háló akkor és csak akkor, ha kizárólag 0 és 1 súlyérték szerepel a címkékben.
A háló t átmenete holt (L_0 eleven) akkor és csak akkor, ha nincs t élcímke a G fedési fában.

24.25. tétel. *Ha $M \in R(M_0)$, akkor $\exists M'$-vel címkézett csúcs úgy, hogy M' lefedi M-et.*



24.20. ábra. II.a. példa fedési fára.

Az azonos címkéjű helyek összevonásával a fedési fából fedési gráf készíthető. A fedési fa, illetve a fedési gráf azonban nem tartalmaz minden elérhető állapotot, így eltérő tulajdonságú hálók fedési fája is lehet azonos. A 24.20. háló eleven, a 24.21. háló holt, fedési fájuk mégsem különbözik.

24.5.2. Elérhetőség szükséges feltételének meghatározása

Átlátszó hálók esetén lineáris algebrai eszközök is használhatók annak vizsgálatára, hogy egyes állapotok elérhetőek-e. Az alábbiakban egy szükséges feltételt adunk elérhetőségre.

24.26. definíció. *Legyen n az átmenetek, m a helyek száma. Legyen $A \in \mathbb{Z}^{n\times m}$ mátrix olyan, hogy*

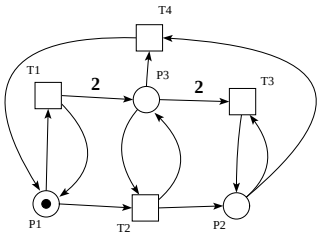
$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} ,$$

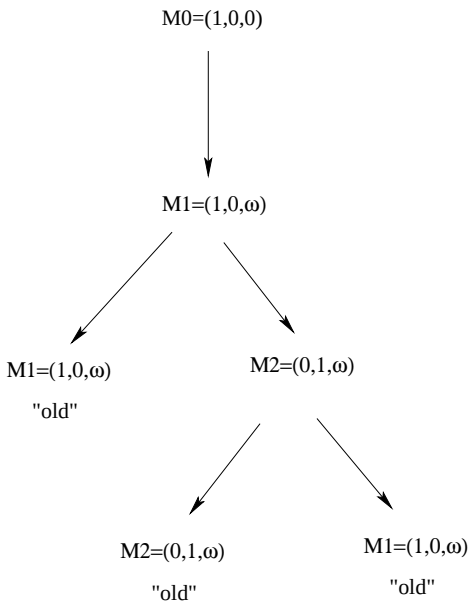
ahol

$$a_{ij}^{+} = v(t_i, p_j) ,$$

$$a_{ij}^{-} = v(p_j, t_i) .$$

A ς akciósorozathoz tartozó $M_k \in \mathbb{N}^{m\times 1}$ $(k = 0 \ldots |\varsigma|)$ vektorok tartalmazzák a k-adik átmenet aktivizálása utáni súlyozásokat. Legyenek az $u_k \in \mathbb{N}^{n\times 1}$ $(k = 1 \ldots |\varsigma|)$ kontroll-vektorok





24.21. ábra. II.b. példa fedési fára.

olyanok, hogy

$$(u_k)_i := \begin{cases} 1, & \textit{ha } \varsigma_k{=}t_i \ , \\ 0 & \textit{különben} \ . \end{cases}$$

Ekkor a következő összefüggéseket lehet felírni:

$$M_k = M_{k-1} + A^T u_k \ ,$$

$$M_d = M_0 + A^T \sum_{k=1}^{d} u_k \ .$$

Legyen $\Delta M = M_d - M_0 = A^T x$, ahol $x = \sum_{k=1}^{d} u_k$. Ekkor ha M_d elérhető M_0-ból, akkor $\exists x \in \mathbb{N}^{n\times 1} : \Delta M = A^T x$.

24.27. tétel. $\exists x \in \mathbb{R}^{n\times 1} : \Delta M = A^T x \Leftrightarrow \forall y \in \mathbb{R}^{m\times 1} : Ay = 0 :< \Delta M, y >= 0$ *(ahol* $< \Delta M, y >$ *a skaláris szorzatot jelöli).*

Bizonyítás. $\Rightarrow$ (Elégségesség)
Feltehető, hogy $\Delta M \neq 0$. Legyen $y \in \mathbb{R}^m : Ay = 0$. Mivel alkalmas x-szel $\Delta M = A^T x$, ezért $y^T \Delta M =< \Delta M, y >= y^T A^T x = (Ay)^T x = 0$.

$\Leftarrow$ (Szükségesség)
Legyenek az $a_1, \ldots, a_n \in \mathbb{Z}^{1\times m}$ vektorok az A mátrix sorvektorai. Azt kell belátni, hogy $\exists x : \Delta M = A^T x$, azaz $\Delta M^T \in \mathcal{L}(a_1, \ldots, a_n)$ (ahol $\mathcal{L}$ a lineáris kombinációt jelöli).

Definiáljuk a következő halmazokat:

$$Y_0 := \{y \in \mathbb{R}^{m\times 1} \,|\, a_i y =< y^T, a_i >= 0 \quad (i = 1 \ldots n)\} ,$$

$$Y_1 := \mathcal{L}(a_1, \ldots, a_n) ,$$

$$Y_2 := \{y \in \mathbb{R}^{m\times 1} \,|\, \forall z \in Y_1 :< z, y >= 0\} .$$

Ekkor $Y_0 = Y_2$, ui. $y \in Y_0 \Leftrightarrow Ay = 0 \Leftrightarrow \forall \alpha_1, \ldots, \alpha_n \in R :< y^T, \alpha_1 a_1 + \ldots + \alpha_n a_n >= 0 \Leftrightarrow \forall z \in Y_1 :< z, y >= 0 \Leftrightarrow y \in Y_2$.

A feltételek szerint $\forall y \in Y_2 :< \Delta M, y >= 0$. Már csak azt kell belátni, hogy $\forall x \in \mathbb{R}^m \forall y \in Y_2 :< x, y >= 0 \Rightarrow x \in Y_1$. Ehhez indirekt módon tegyük fel, hogy $\exists x \in \mathbb{R}^m \setminus Y_1 \forall y \in Y_2 :< x, y >= 0$. Mivel $\mathbb{R}^m = Y_1 \oplus Y_2$, ezért $\exists a \in Y_1, b \in Y_2 \setminus \{0\} : x = a + b$. Ezt felhasználva az indirekt feltevés szerint $< a + b, y >=< a, y > + < b, y >=< b, y >= 0$. Mivel y választása tetszőleges volt, legyen most y egyenlő b-vel. Ekkor viszont $\|b\|^2 = 0$, ami (a normált tér axiómái miatt) azzal ekvivalens, hogy $b = 0$. Ezzel pedig ellentmondásra jutottunk az indirekt feltevésünkkel.

A tétel feltételét felhasználva, valamint az előző állítást $x = \Delta M$-re alkalmazva a bizonyítandó állítást kapjuk. ■

Legyen M_d elérhető M_0-ból (ekkor $\exists x \in \mathbb{N}^n : \Delta M = A^T x$). Legyen $r = \rho(A)$ az A rangja. Sor és oszlopcserékkel az A mátrix a következő alakra transzformálható:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} ,$$

ahol $A_{11} \in \mathbb{Z}^{r\times(m-r)}, A_{12} \in \mathbb{Z}^{r\times r}, A_{21} \in \mathbb{Z}^{(n-r)\times(m-r)}, A_{22} \in \mathbb{Z}^{(n-r)\times r}$ és $|A_{12}| \neq 0$. Az A mátrix mellett hasonló módon ΔM-et átalakítva az eredeti egyenletrendszerrel ekvivalens egyenletrendszert kapunk. Legyen $\Delta M = \begin{bmatrix} \Delta M_1 \\ \Delta M_2 \end{bmatrix}$, ahol $\Delta M_1 \in \mathbb{N}^{m-r}$ és $\Delta M_2 \in \mathbb{N}^r$. Transzponáljuk az átalakított egyenletrendszert:

$$\left[\Delta M_1^T, \Delta M_2^T\right] = \left[x_1^T A_{11} + x_2^T A_{21}, x_1^T A_{12} + x_2^T A_{22}\right],$$

ahol $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ és $x_1 \in \mathbb{N}^r, x_2 \in \mathbb{N}^{n-r}$. Mivel $\rho([A_{11}, A_{12}]) = r$, így $[A_{11}, A_{12}]$ sorai lineáris kombinációként előállítják $[A_{21}, A_{22}]$ sorait. Ebből következik, hogy $\forall x_2 \exists \widehat{x_2} : x_2^T [A_{21}, A_{22}] = \widehat{x_2}^T [A_{11}, A_{12}]$. Ezt felhasználva:

$$\left[\Delta M_1^T, \Delta M_2^T\right] = x_1^T [A_{11}, A_{12}] + \widehat{x_2}^T [A_{11}, A_{12}] = \widehat{x}^T [A_{11}, A_{12}] ,$$

ahol $\widehat{x}^T = x_1^T + \widehat{x_2}^T$. Az egyenletrendszert szétbontva:

$$\Delta M_1^T = \widehat{x}^T A_{11},$$

$$\Delta M_2^T = \widehat{x}^T A_{12} \Leftrightarrow \Delta M_2^T A_{12}^{-1} = \widehat{x}^T .$$

Az első egyenletrendszerben $\widetilde{x}^T$ helyébe $\Delta M_2^T A_{12}^{-1}$-et helyettesítve kapjuk:

$$\Delta M_1^T = \Delta M_2^T A_{12}^{-1} A_{11} \Rightarrow \Delta M_1 = A_{11}^T (A_{12}^T)^{-1} \Delta M_2 \ .$$

Ekkor

$$\left[I_{m-r}, -A_{11}^T (A_{12}^T)^{-1} \right] \begin{bmatrix} \Delta M_1 \\ \Delta M_2 \end{bmatrix} = 0 \ .$$

24.28. tétel. *Legyen* $B_f = \left[I_{m-r}, -A_{11}^T (A_{12}^T)^{-1} \right]$. *Ha* M_d *elérhető* M_0*-ból, akkor* $B_f \Delta M = 0$.

Gyakorlatok

24.5-1. Rajzoljuk fel az evő filozófusok viselkedését modellező Petri-háló fedési fáját és fedési gráfját.

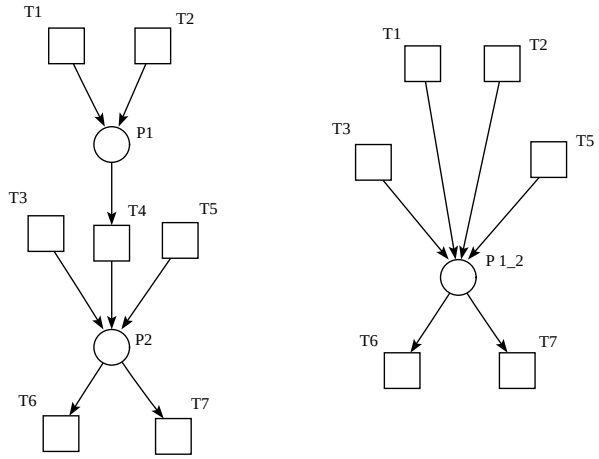
24.5-2. Határozzuk meg a 24.12. ábra két Petri-hálójának fedési fáját és fedési gráfját.

24.6. Elevenséget, biztonságosságot és korlátosságot megőrző transzformációk

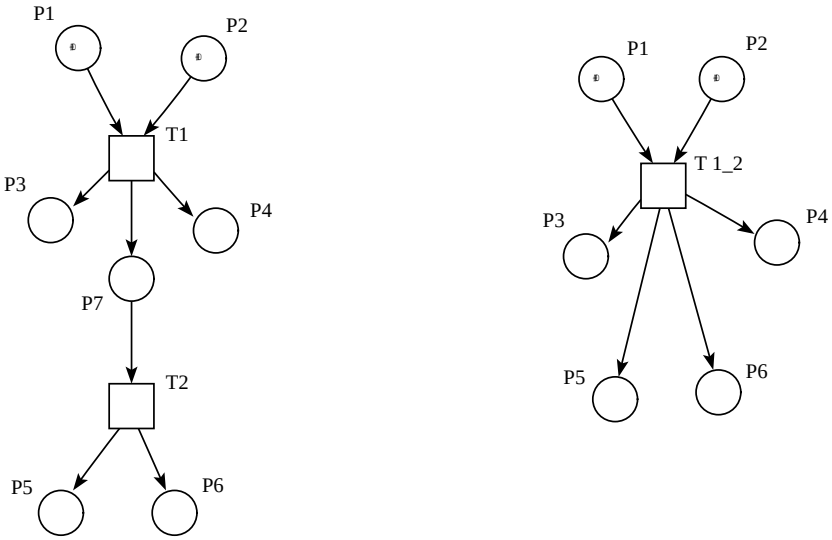
Minél kevesebb csúcspontot és élet tartalmaz egy Petri-háló gráfja, annál könnyebb a háló tulajdonságait megállapítani. Az alábbi, lokális transzformációk alkalmazásával lépésenként egyszerűsíthetjük a vizsgált Petri-hálót oly módon, hogy az eredményül kapott hálók és az eredeti háló elevenségét és korlátosságát leíró tulajdonságok nem változnak. Egy (N, M) Petri-hálóból kapott (N', M') Petri-háló akkor és csak akkor eleven, korlátos, illetve biztonságos, ha (N, M) rendelkezik a megfelelő tulajdonsággal. Ezen transzformációk alkalmazása emeli az absztrakció szintjét. A transzformációk külső kommunikációs és szinkronizációs kapcsolatokkal nem rendelkező szekvenciális részfolyamatok belső struktúrájának részleteit leíró részgráfokat törölnek a Petri-hálóból. Részgráfok törlése egyben azt eredményezi, hogy a folyamat működésének kevésbé részletgazdagabb modelljéhez jutunk. A transzformációk megfordításának alkalmazásával pedig úgy finomíthatjuk egy folyamat modelljét, hogy az elevenség, korlátosság, biztonságosság megmarad. A transzformációkat bemutató ábrákon látható gráfok nem feltétlenül teljes Petri-hálók, hanem azok részgráfjai is lehetnek.

A helyek sorozatának összevonása transzformáció (24.22. ábra) segítségével két helyet (P_1, P_2) és az őket összekötő átmenetet (T_4) helyettesíthetjük egyetlen hellyel (P_{1_2}). Indirekt módon belátható, hogy ha mindkét régi hely k-korlátos volt, akkor és csak akkor az új hely is k-korlátos. Ha ugyanis a P_{1_2} helyen több, mint k súly jelenne meg, akkor ezek a súlyok megoszthatóak lettek volna az eredeti hálóban a T_4 átmenet működésének alkalmas megválasztásával úgy, hogy mindegyik a P_1 helyen, illetve a P_2 helyen van. A törölt T_4 átmenet attól függően eleven vagy sem, hogy a P_1 helyet megelőző átmenetek között van-e eleven. Az átmenet eltűnésével tehát a háló elevensége sem változik.

A helyek sorozatának összevonása transzformáció duálisa az átmenetek sorozatának összevonása (24.23. ábra) transzformáció. A P_7 hely akkor és csak akkor k-korlátos, ha a rákövetkező átmenet kimenő helyei (P_5 és P_6) is k-korlátosak. A hely törlésével tehát a háló korlátossága nem változik. A T_2 átmenet, illetve a két átmenet összevonásával keletkező átmenet is pontosan akkor eleven, ha a T_1 átmenet eleven. Az átmenetek összevonásával tehát a háló elevensége sem változik.



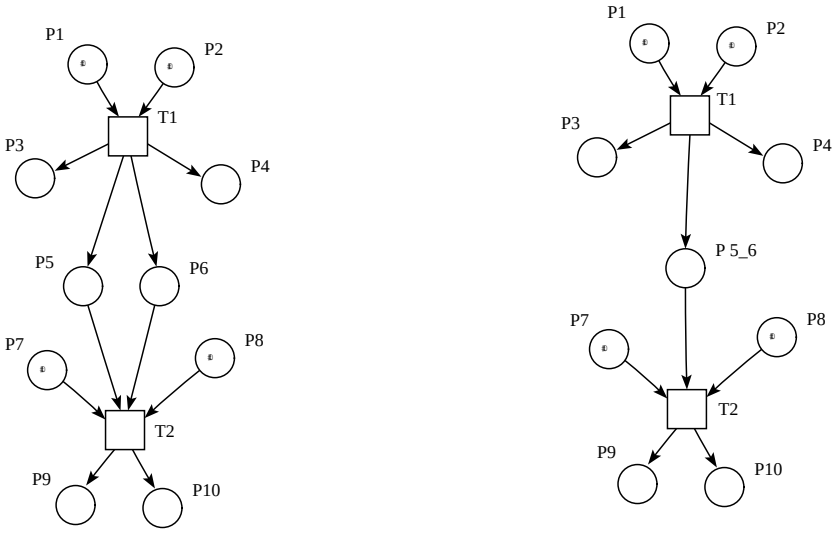
24.22. ábra. Helyek sorozatának összevonása.



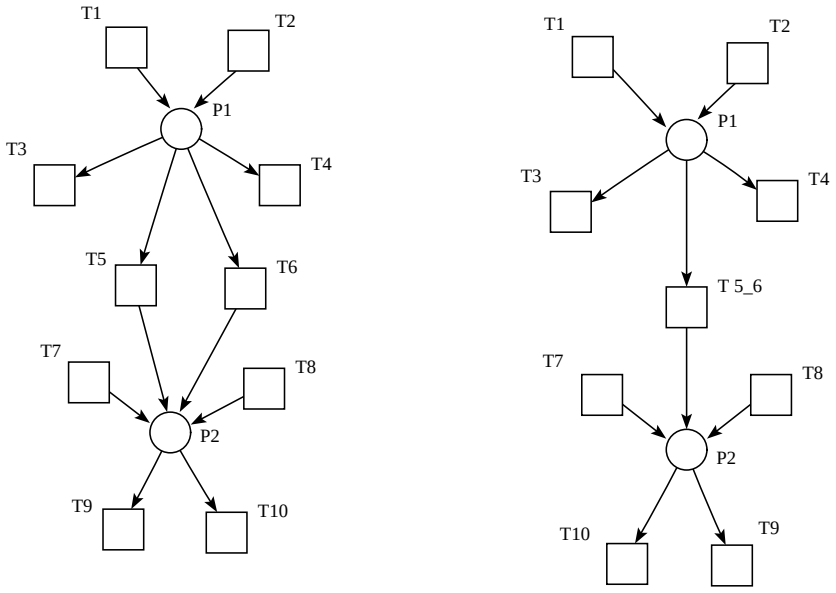
24.23. ábra. Átmenetek sorozatának összevonása.

A párhuzamos helyek összevonása transzformációt mutatja be a 24.24. ábra. A helyek (P_5,P_6) az átmenet különböző előfeltételeit modellezik. Ezek a feltételek azonban teljesen egyenértékűek a háló viselkedése szempontjából, így egyetlen átmenettel helyettesíthetőek.

A párhuzamos átmenetek összevonása transzformációt mutatja be a 24.25. ábra. Ha az átmenetek bármelyike tüzel, akkor a P_1 helyről a P_2-re helyez át súlyt. A modellezett folyamat szempontjából a két átmenet különböző eseményeknek felel meg, de a Petri-háló viselkedése szempontjából egyenértékűek és egyetlen átmenettel helyettesíthetőek.

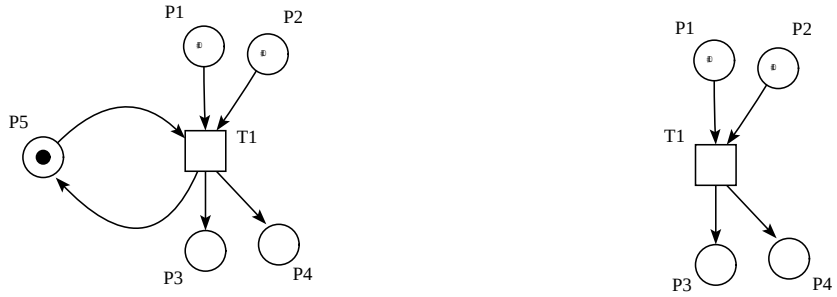


24.24. ábra. Párhuzamos helyek összevonása.

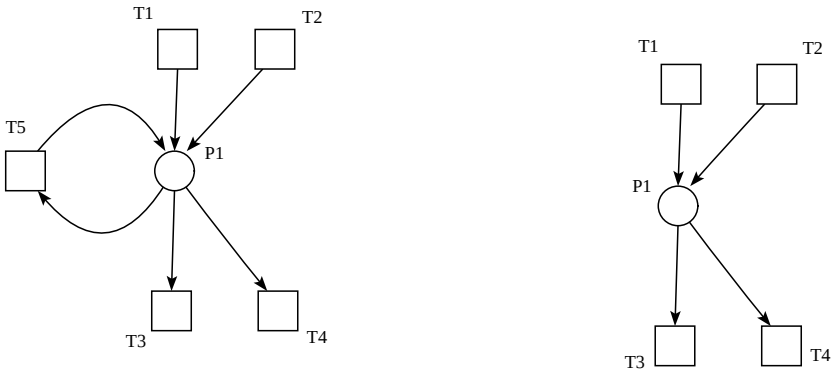


24.25. ábra. Párhuzamos átmenetek összevonása.

A 24.26. és a 24.27. ábrákon látható transzformációk hurkok elhagyására adnak lehetőséget. Ha egy hely súlyozott, egyetlen átmenethez kapcsolódik, annak egyszerre bemenő és kimenő helye, akkor az átmenet viselkedését nem befolyásolja. Ha egy átmenet egyetlen



24.26. ábra. Hurok helyek elhagyása.



24.27. ábra. Hurok átmenetek elhagyása.

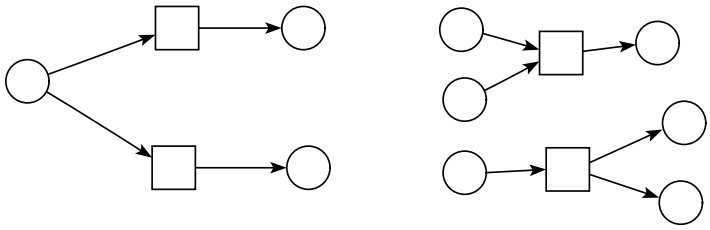
helyhez kapcsolódik és az onnan elvett súlyokat a tüzelése eredményeként visszahelyezi, akkor a háló többi részének viselkedésére nem hat ki a működése.

Gyakorlatok

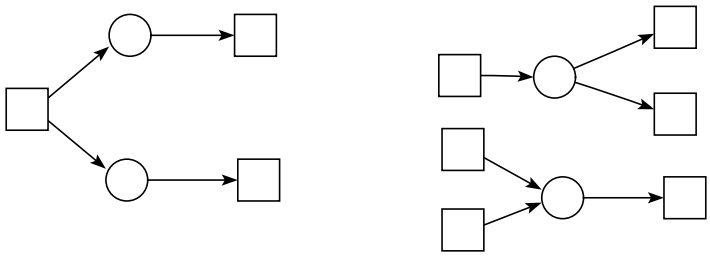
24.6-1. Állapítsuk meg a 24.14. ábra bal oldali hálójának elevenségi és biztonságossági tulajdonságait az alfejezetben ismertetett transzformációkkal.

24.7. Petri-hálók osztályozása

A Petri-hálók szerkezeti felépítése nagyon sokféle lehet, a háló viselkedése a struktúrája alapján általában nehezen határozható meg. Egyes részosztályokra azonban megadhatóak olyan könnyen ellenőrizhető feltételek, amelyek alapján könnyen eldönthető, hogy az adott osztályba tartozó hálók elevenek-e, illetve biztonságosak-e. Részosztályokat a normális hálók részhalmazán belül definiálunk, ahol minden él súlya egy. A részosztályok meghatáro-



24.28. ábra. Példa állapotgépre, illetve állapotgépben nem megengedett részgráfok.



24.29. ábra. Példa jelzett gráfra, illetve jelzett gráfban nem megengedett részgráfok.

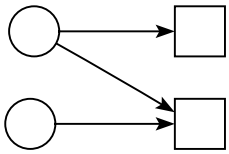
zása alapvető programozási fogalmakhoz kapcsolódik: a konfliktushoz, illetve a szinkronizációhoz.

24.29. definíció (állapotgép). *Az **állapotgép** olyan Petri-háló, melyben minden átmenetnek pontosan egy megelőző és egy rákövetkező helye van* ($\forall t \in T : |{}^\bullet t| = |t^\bullet| = 1$).

Egy állapotgépben nem fordulhat elő a 24.28. ábra jobb oldalán látható részgráfok egyike sem. Állapotgépben minden átmenetnek egyetlen hellyel reprezentált előfeltétele lehet csak, ily módon az állapotgép szinkronizációt nem enged meg. Ha csak egyetlen súly van a hálóban, akkor működése egy szekvenciális véges állapotú gépnek, a gép állapotai pedig egy-egy súlyozott helynek felelnek meg.

24.30. definíció (jelzett gráf). *A **jelzett gráf** olyan Petri-háló, melyben minden helynek pontosan egy megelőző és egy rákövetkező átmenete van* ($\forall p \in P : |{}^\bullet p| = |p^\bullet| = 1$).

Ha a Petri-háló egy-egy helyéből és a beléje mutató, illetve a belőle induló éléből egyetlen összetett élet készítünk, akkor a jelzett gráf reprezentálható olyan gráffal is, amelyben az egymást követő átmeneteket kötjük össze közvetlenül. Ebben az esetben a Petri-háló helyein megjelenő súlyokat a gráf megfelelő élein jelezzük, innen származik a részosztály elnevezése. Egy jelzett gráfban nem fordulhat elő a 24.29. ábra jobb oldalán látható részgráfok egyike sem, tehát nem alakulhat ki konfliktus. Minden jelzett gráf perzisztens kezdősúlyozásától függetlenül. Megjegyezzük, hogy nem kizárólag jelzett gráf lehet konfliktusmentes, de perzisztens és biztonságos háló jelzett gráffá konvertálható.



24.30. ábra. Szabadválasztású hálóban nem megengedett részgráfok.

Az állapotgépek és a jelzett gráfok osztályának metszete nem üres, van olyan Petri-háló, amelyik mindkettőnek eleme (24.16. ábra bal oldali hálója). A szabadválasztású hálók az állapotgépek és a jelzett gráfok általánosításai, konfliktust és szinkronizációt is megengednek, de ezek lokálisan együttes megjelenését, a zavart (24.7 ábra bal oldala) nem.

24.31. definíció (szabadválasztású háló (FC)). *Szabadválasztású (FC) a háló, ha* $\forall p \in P : |p^\bullet| \leq 1$ *vagy* ${}^\bullet(p^\bullet) = \{p\}$. *Másképpen:* $\forall p_1, p_2 : p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow |p_1^\bullet| = |p_2^\bullet| = 1$.

Ehhez a részosztályhoz tartozó hálók esetén az egy helyről kiinduló minden él vagy a hely egyetlen kimenő éle, vagy egy átmenet egyetlen bemenő éle. A szabadválasztású és a kiterjesztett szabadválasztású hálók elnevezése azt a tulajdonságukat tükrözi, hogy két, közös bemenő hellyel rendelkező átmenet esetén nincs olyan súlyozás, amelyiknél csak az egyik megengedett, tehát ezen átmenetek közül szabadon választhatunk, hogy melyik tüzeljen. A 24.30. ábrán bemutatott struktúra szabadválasztású hálóban nem fordulhat elő, így zavar sem jöhet létre.

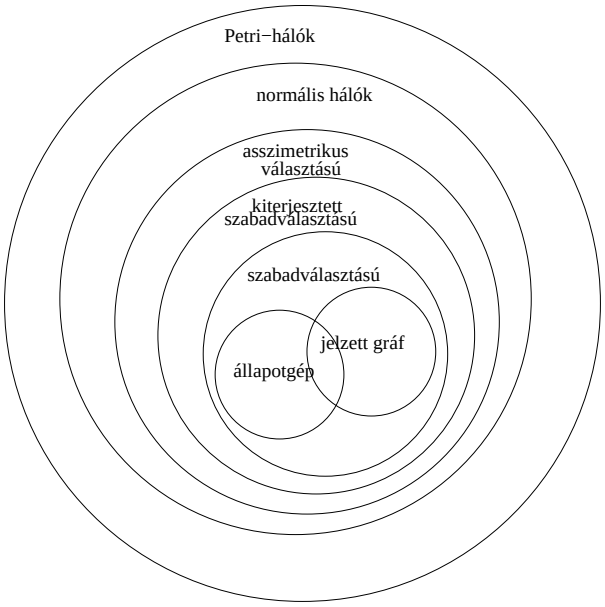
24.32. definíció (kiterjesztett szabadválasztású (EFC) háló). *Egy háló kiterjesztett szabadválasztású (EFC) háló, ha* $\forall p_1, p_2 : p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow p_1^\bullet = p_2^\bullet$.

Az aszimmetrikusan választó (vagy más néven egyszerű) hálók esetén még tovább gyengítjük a feltételeket és csak annyit követelünk, hogy két olyan átmenet esetén, ahol a bemenő helyek halmazainak metszete nem üres, az egyik halmaz tartalmazza a másikat. A 24.31. ábrán láthatjuk az egyes osztályok között fennálló kapcsolatokat.

24.33. definíció (aszimmetrikusan választható (AC) háló). *Egy háló aszimmetrikusan választható (AC) háló, ha* $\forall p_1, p_2 : p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow p_1^\bullet \subseteq p_2^\bullet \vee p_2^\bullet \subseteq p_1^\bullet$

Petri-hálók tulajdonságainak elemzéséhez hasznos alapfogalom a szifon, a csapda és a helyinvariáns. Mindhárom helyek részhalmazára vonatkozik. A szifon egy olyan helyhalmaz, amelyikre teljesül, hogy minden olyan átmenetnek, amelyiknek van a halmazhoz tartozó kimenő helye, annak van a halmazhoz tartozó bemenő helye is. Ez azt jelenti, hogy a szifonba súlyt helyező átmenetek a szifonból el is vesznek súlyt. Ha egy szifon sima, azaz nem tartalmaz súlyt, akkor sima is marad. Nem lehetnek elevenek azok az átmenetek, amelyeknek van olyan bemenő helye, amely sima vagy simává váló szifonhoz tartozik. A 24.47. lemma ad magyarázatot arra, hogy miért szokták a szifont holtpontnak is nevezni. Egy helyinvariáns egy olyan helyhalmaz, amelyen a súlyösszeg állandó. Helyinvariáns alkalmazására mutatott példát a 24.6. tétel bizonyítása.

24.34. definíció (szifon/holtpont). *S helyek halmaza* ***holtpont/szifon****, ha* ${}^\bullet S \subseteq S^\bullet$.



24.31. ábra. Petri-hálók osztályai.

A csapda egy olyan helyhalmaz, amelyikre teljesül, hogy minden olyan átmenetnek, aminek van a halmazhoz tartozó bemenő helye, annak van a halmazhoz tartozó kimenő helye is. Ez azt jelenti, hogy a csapdából súlyt eltávolító átmenetek a csapdába helyeznek is súlyt. Ha egy csapda súlyozott, akkor súlyozott is marad. Ez az csapdához csatlakozó átmenetek elevensége szempontjából kedvező helyzet.

24.35. definíció (csapda). *S helyek halmaza **csapda**, ha $S^\bullet \subseteq {}^\bullet S$.*

A definíciókból következik, hogy csapdák, szifonok uniója csapda, illetve szifon. Egy szifon vagy csapda alapszifon, illetve alapcsapda, ha nem más szifonok, illetve csapdák uniója. Minimális egy szifon vagy csapda, ha semmilyen valódi része nem szifon, illetve csapda.

A Petri-hálók részosztályai között tár fel kapcsolatokat a duális és a fordított háló fogalma. Egy háló fordítottja az a háló, amelyet úgy kapunk, hogy az összes irányított élet megfordítjuk.[2] Egy háló duálisának nevezzük azt a hálót, amelyiket úgy kapunk, hogy a gráfban az átmeneteket helyekkel, a helyeket átmenetekkel helyettesítjük. Jelzett gráf duálisa az állapotgép. Egy szabadválasztású Petri-háló fordított-duálisa is szabadválasztású Petri-háló. A szifon a csapda fordítottja.

[2] A háló fordítottjának (vagy duálisának) nevezzük azt a hálót is, amelyet a megfelelő transzformáció és átcímkézés kompozíciójával kapunk.

Szifonok meghatározása

Legyen $S \subseteq P$ az N Petri-háló helyeinek egy halmaza. Jelölje az s_i logikai változó a következő állítást: $p_i \in S$ $(i = 1 \ldots |P|)$. Legyen az A_i $(i = 1 \ldots |P|)$ predikátum a következő:

$$s_i \to ((s_{(j_1)_1} \vee \ldots \vee s_{(j_1)_{m_1}}) \wedge \ldots \wedge (s_{(j_n)_1} \vee \ldots \vee s_{(j_n)m_n})) \ ,$$

ahol $\left\{t_{j_1}, \ldots, t_{j_n}\right\} = {}^\bullet p_i$, $m_k = \left|{}^\bullet t_{j_k}\right|$ $(k = 1 \ldots n)$ és $\forall k = 1 \ldots n : \left\{p_{(j_k)_1}, \ldots, p_{(j_k)m_k}\right\} = {}^\bullet t_{j_k}$.

24.36. tétel. $S = \left\{p_{l_1}, \ldots, p_{l_{|S|}}\right\}$ *holtpont* $\Leftrightarrow A_1 \wedge \ldots \wedge A_{|P|} =$ IGAZ *az* $s_{l_1} = s_{l_2} = \ldots = s_{l_{|S|}} =$ IGAZ, $s_{l_{|S|+1}} = s_{l_{|S|+2}} = \ldots = s_{l_{|P|}} =$ HAMIS *igazságértékelés mellett.*

Bizonyítás. Tegyük fel, hogy $\exists i = 1 \ldots |P| : A_i =$ HAMIS. Feltehető, hogy $i \notin \{l_{|S|+1}, \ldots, l_{|P|}\}$ ui. ekkor $A_i = \neg s_i \vee (\ldots) =$ IGAZ. Ha $i \in \{l_1, \ldots, l_{|S|}\}$ (azaz $p_i \in S$) akkor $\neg s_i =$ HAMIS miatt a

$$(s_{(j_1)_1} \vee \ldots \vee s_{(j_1)_{m_1}}), \ldots, (s_{(j_n)_1} \vee \ldots \vee s_{(j_n)m_n})$$

klózok közül legalább az egyik hamis. Legyen ez pl. az

$$(s_{(j_1)_1} \vee \ldots \vee s_{(j_1)_{m_1}})$$

klóz. Ekkor $t_{j_1} \in {}^\bullet p_i \subseteq {}^\bullet S$ (azaz $(t_{j_1}, p_i) \in T \times S$) és $\forall q \in {}^\bullet t_{j_1} : q \notin S$ azaz ${}^\bullet t_{j_1} \bigcap S = \oslash$, amivel ellentmondásra jutottunk.

Tegyük fel, hogy S nem szifon. Ekkor $\exists (t, p_{l_1}) \in T \times S : {}^\bullet t \bigcap S = \oslash$. Legyen $\left\{p_{j_1}, \ldots, p_{j_m}\right\} = {}^\bullet t$. Ekkor $s_{j_1} = s_{j_2} = \ldots = s_{j_m} =$ HAMIS és $s_{l_1} =$ IGAZ, ezért

$$s_{l_1} \to ((s_{j_1} \vee s_{j_2} \vee \ldots \vee s_{j_m}) \wedge \ldots) = \text{HAMIS} \ ,$$

amivel ellentmondásra jutottunk. ■

Tehát az összes olyan igazságértékelés, amely kielégíti az $A_1 \wedge \ldots \wedge A_{|P|}$-t, meghatároz egy szifont. A feladat ezek után ezen igazságértékelések meghatározása. Ezzel ekvivalens feladat azon igazságértékelések megkeresése, melyek nem elégítik ki a $\neg(A_1 \wedge \ldots \wedge A_{|P|})$-t. Ezen igazságértékelések könnyen meghatározhatók úgy, hogy a $\neg(A_1 \wedge \ldots \wedge A_{|P|})$ konjunktív normálformájában (KNF) szereplő összes klózra megkeressük az őt hamissá tevő igazságértékeléseket.

24.1. példa.

$$s_1 \to s_2, s_2 \to s_3 \vee s_4, s_3 \to s_1 \wedge s_2, s_4 \to s_1 \ .$$

Az ebből kapott KNF:

$$(\neg s_1 \vee s_2) \wedge (\neg s_2 \vee s_3 \vee s_4) \wedge (\neg s_3 \vee s_1) \wedge (\neg s_3 \vee s_2) \wedge (\neg s_4 \vee s_1) \ .$$

A fenti KNF negáltjának KNF-je:

$$(\neg s_1 \vee \neg s_2 \vee \neg s_3) \wedge (\neg s_1 \vee \neg s_2 \vee \neg s_3 \vee s_4) \wedge (\neg s_1 \vee \neg s_2 \vee \neg s_4)$$

$$\wedge (\neg s_1 \vee \neg s_2 \vee s_3 \vee \neg s_4) \wedge (s_1 \vee s_2 \vee s_3 \vee s_4) \ .$$

Ebből a megoldások:

s_1	s_2	s_3	s_4
T	T	T	*
T	T	T	F
T	T	*	T
T	T	F	T
F	F	F	F

azaz az adott gráf holtpontjai: $\oslash, \{p_1, p_2, p_3, p_4\}, \{p_1, p_2, p_3\}, \{p_1, p_2, p_4\}$.

24.37. megjegyzés. *A tartalmazott klózok a negált KNF-ből elhagyhatók.*

Gyakorlatok

24.7-1. Állapítsuk meg, hogy a 24.13.–24.16 ábrák egyes hálói melyik Petri-háló osztályhoz tartoznak.

24.7-2. Keressünk csapdákat és szifonokat a 24.13. ábra hálóin.

24.8. Eleven és biztonságos Petri-hálók

Arra a kérdésre szeretnénk választ kapni, hogy adott struktúrájú Petri-hálóhoz létezik-e olyan kezdősúlyozás, hogy a háló eleven és biztonságos egyszerre. Először néhány egyszerűbb szükséges feltételt határozunk meg, majd a részletesebb választ a Petri-hálók egyes osztályaira külön-külön adjuk meg. Nem lehet egy Petri-háló biztonságos is és eleven is, ha forrás vagy nyelő átmenetet vagy olyan helyet tartalmaz, amelyiknek nincs legalább egy-egy bemenő és kimenő átmenete.

A 24.32. ábra bal oldali részén egy olyan helyet látunk, amelyiknek nincs bemenő éle: ${}^\bullet p = \emptyset$. A helyet követő t átmenet nem lehet eleven. Ha a kezdősúlyozás biztonságos és legfeljebb 1 súlyt helyez a P helyre, akkor a t tüzelése után a súly 0-ra csökken. További súly nem kerülhet a p helyre, így a t átmenet legfeljebb L_1 szinten eleven.

A 24.32. ábra jobb oldalán látható Petri-háló p helyének nincs kimenő átmenete: $p^\bullet = \emptyset$. Ha t eleven, akkor p nem lehet biztonságos. Az eleven átmenet újra és újra súlyokat helyez el a p helyen, így a hely semmilyen k természetes számra sem k-korlátos.

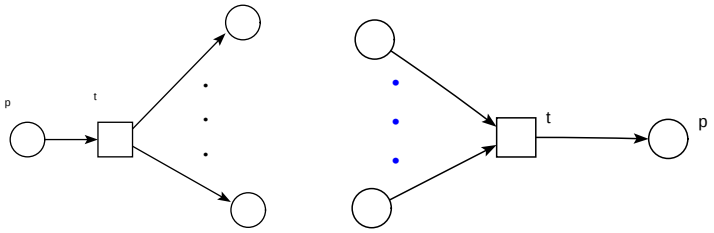
A 24.33. ábra bal oldali részén t forrás átmenet: ${}^\bullet t = \emptyset$. A forrás átmenet bármikor tüzelhet, így semmi sem korlátozza azt, hogy a p helyen felhalmozzon súlyokat. A p hely tehát nem biztonságos.

A 24.33. ábra jobb oldalán t nyelő átmenet: $t^\bullet = \emptyset$. Ha p biztonságos, akkor a p helyen t lustasága, inaktivitása esetén sem halmozódhatnak fel súlyok. Ez azt jelenti, hogy t nem lehet eleven, nem áll rendelkezésre elegendő súly ahhoz, hogy újra és újra tüzeljen.

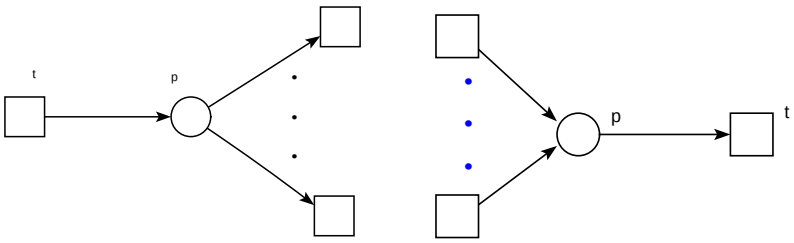
Beláttuk, ha (N, M_0) eleven és biztonságos, akkor $\forall x \in P \cup T : x^\bullet \neq \emptyset$ és ${}^\bullet x \neq \emptyset$. Erősen összefüggőnek nevezünk egy irányított gráfot, ha minden csúcsából minden csúcsába vezet irányított út. Teljesül a következő általánosabb tétel is:

24.38. tétel. *Összefüggő, eleven és biztonságos háló erősen összefüggő.*

A tétel általában nem megfordítható. Nem minden erősen összefüggő hálóhoz találunk



24.32. ábra. Példák.

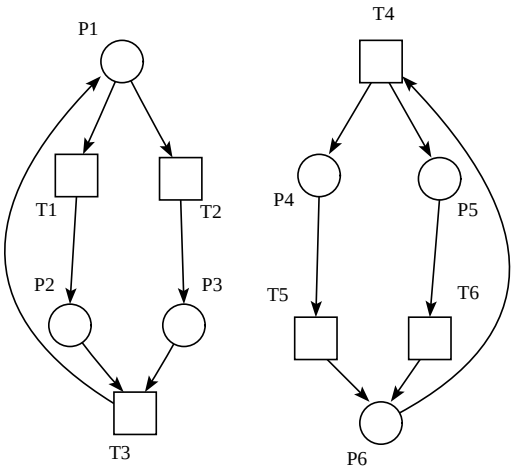


24.33. ábra. Példák.

eleven és biztonságos kezdősúlyozást. Két ellenpéldát mutat a 24.34. ábra. Az első hálónak nincs olyan kezdősúlyozása, amely mellett eleven, a második hálónak nincs olyan nem teljesen üres kezdősúlyozása, amely mellett biztonságos, pedig mindkettő erősen összefüggő. Állapotgépek és jelzett gráfok esetén azonban belátható, hogy a tétel megfordítható, ha a gráf erősen összefüggő, akkor ezekben az esetekben létezik eleven és biztonságos súlyozás.

24.8.1. Állapotgép eleven és biztonságos súlyozása

Az állapotgép egyszerű szerkezete könnyen megadhatóvá teszi az eleven súlyozás szükséges és elégséges feltételét. Állapotgépben a súlyok összege nem változhat a működés során, minden átmenet egy súlyt vesz el a bemenő helyéről és egy súlyt helyez el a kimenő helyére. Ha a gráf erősen összefüggő és legalább egy súlyt tartalmaz, akkor a súlyt tartalmazó bármelyik helytől tetszőleges átmenet előtti helyre odavihető a súly az út mentén elhelyezkedő átmenetek aktivizálásával. Így bármelyik átmenet aktivizálható újra és újra. Ha több,



24.34. ábra. Ellenpélda a 24.38 tétel megfordítására.

mint egy súly is van a gráfban, akkor ezek a súlyok utolérhetik egymást és a háló nem lesz biztonságos. Kimondhatjuk tehát az alábbi tételeket:

24.39. tétel. *Egy összefüggő állapotgép*

- *akkor és csak akkor eleven, ha erősen összefüggő és van benne legalább egy súly.*
- *akkor és csak akkor biztonságos, ha legfeljebb egy súly van benne.*
- *akkor és csak akkor eleven és biztonságos, ha pontosan 1 súly van benne és erősen összefüggő.*

24.8.2. Jelzett gráf eleven és biztonságos súlyozása

Jelzett gráfok esetén a helyeket „beleolvasztjuk” az átmeneteket összekötő élekbe. Az esetleges súlyt jelezni kell az él mentén. Tekintettel arra, hogy a normális hálók osztályán belül definiáljuk a jelzett gráfok osztályát, ezért minden átmenet pontosan egy-egy súlyt vesz le a bemenő élekről és egy-egy súlyt helyez el a kimenő élein, ha tüzel. Ha kiválasztunk egy tetszőleges irányított kört a gráfban, akkor a kör két szomszédos élének súlyösszege nem változik akkor, amikor a találkozásuknál elhelyezkedő átmenet tüzel. Ez azt jelenti, hogy tetszőleges irányított körhöz tartozó éleken elhelyezkedő súlyok összege állandó:

24.40. tétel. *Jelzett gráf irányított körén a súlyösszeg invariáns (helyinvariáns), azaz* $\forall M \in R(M_0) : \forall C : M_0(C) = M(C)$, *ahol a* C *az irányított kör élhalmaza.*

Súlymentes irányított körök tehát súlymentesek maradnak a háló működése során. Egy erősen összefüggő jelzett gráfban egy átmenet holt, ha rajta van egy súlymentes irányított körön. Indirekt úton belátható, ha egy átmenet egyetlen súlymentes irányított körnek sem

eleme, akkor a vele azonos irányított körökön lévő csúcspontokra is igaz ugyanez. A körök mentén tehát újra és újra eljutnak a bemenő éleire a súlyok.

24.41. tétel. *Erősen összefüggő jelzett gráf akkor és csak akkor eleven, ha minden irányított körén legalább egy súly található.*

Az eleven és biztonságos súlyozás megtalálásához el kell távolítanunk a hálóból a felesleges súlyokat. Ha egy irányított körben egynél több súly is van, akkor ezek a súlyok utolérhetik egymást, a háló nem biztonságos. Megmutatjuk, hogy egy eleven jelzett gráf súlyozása lépésenként csökkenthető oly módon, hogy az elevenség, mint invariáns tulajdonság mindig megmarad és a végén a háló biztonságos lesz. Az eljárás véges lépésben véget ér, eleven jelzett gráf súlyainak száma ugyanis mindig egy pozitív érték. A súlyok száma ily módon az eljárás variáns függvénye.

24.42. tétel (mini-max tétel). *Jelzett gráfban egy él maximális súlya megegyezik annak a rajta áthaladó irányított körnek a súlyösszegével, amelyiknek a legkisebb a súlyösszege.*

Bizonyítás. Jelöljük $C_1, \ldots, C_m$-mel az $e = (x, y)$ élen átmenő köröket, $e_1, \ldots, e_m$-mel az e előtti éleket. Mindegyik e előtti élre mozgassunk a lehető legtöbb súlyt, a rajta áthaladó irányított kör összes súlyát. Tüzeljünk x átmenettel minél többször úgy, hogy közben y átmenet ne tüzeljen. Ez legfeljebb annyiszor lehetséges, amíg valamelyik e előtti élről elfogynak a súlyok, tehát az irányított körök súlyösszegeinek minimuma a felső korlát. ■

A mini-max tétel szerint akkor biztonságos egy jelzett gráf, ha minden él legalább egy olyan irányított körhöz tartozik, amelynek súlyösszege egy. Eleven jelzett gráf tehát akkor és csak akkor biztonságos, ha minden élhez létezik egy olyan C irányított kör, amin rajta van és aminek kezdősúlyozása, $M_0(C) = 1$. Konstruáljuk meg egy erősen összefüggő jelzett gráf eleven és biztonságos súlyozását a következőképpen. Az elevenség érdekében helyezzünk el minden irányított körre legalább egy súlyt, majd vizsgáljuk meg, hogy van-e olyan elérhető súlyozás, amikor a súlyok száma egy adott élen több, mint egy. A felesleges súlyok az elevenség veszélyeztetése nélkül eltávolíthatók erről az élről, hiszen az élt tartalmazó irányított körön továbbra is marad súly. Folytassuk ezt az eljárást addig, amíg marad olyan állapot, amelyikben a súlyozás nem biztonságos. Beláttuk a következő tételt:

24.43. tétel. *Jelzett gráfnak akkor és csak akkor létezik eleven és biztonságos súlyozása, ha erősen összefüggő.*

Jelzett gráfok eleven és biztonságos súlyozásának feltételei megfogalmazhatóak a visszacsatoló élhalmaz fogalmának segítségével is. Visszacsatoló élhalmaznak nevezzük egy irányított gráf éleinek halmazát akkor, ha ezen élek eltávolítása után a gráf irányított kört nem tartalmaz. Egy adott gráf visszacsatoló élhalmaza többféleképpen is kiválasztható, nem feltétlenül egyértelmű.

24.44. definíció (visszacsatoló élhalmaz). *E' **visszacsatoló élhalmaz** (VéH) a $G(V, E)$ gráfban, ha $G' = (V, E - E')$ körmentes. Egy VéH minimális, ha egyetlen valódi részhalmaza sem VéH.*

A 24.41. tétel következményeként eleven jelzett gráf nem nulla súlyú éleinek halmaza VéH-t alkot, valamint ha egy jelzett gráf valamely VéH-ának minden eleme súlyozott, akkor a jelzett gráf eleven.

24.45. tétel. *Egy erősen összefüggő eleven jelzett gráf akkor és csak akkor biztonságos, ha* $\forall M \in R(M_0)$: *a súlyozott élek halmaza minimális VéH.*

A 24.43. tétel kimondásához vezető gondolatmenet szerint minimális visszacsatoló élhalmazok súlyozása esetén már nincs több eltávolítható súly az elevenség megőrzése mellett.

24.8.3. Elevenség és biztonságosság szabadválasztású és aszimmetrikus választású hálókban

Szabadválasztású Petri-hálók eleven és biztonságos súlyozásának feltételeit szifonok és csapdák vizsgálata alapján határozhatjuk meg.

Először megmutatjuk, hogy ha egy Petri-háló W átmenethalmazát megelőző sima helyek bemenő átmenetei is mind W-beliek, akkor vagy van megengedett t átmenet W elemei között, vagy egy sima szifon utódhalmaza tartalmazza W-t. Formálisan:

24.46. lemma. *Egy (N,M) Petri-hálóban jelölje* M^0 *a sima, illetve* M^+ *a súlyozott helyek halmazát,* $P = M^0 \cup M^+$. *Legyen* $W \subseteq T$ *átmenetek egy részhalmaza. Ha* ${}^\bullet({}^\bullet W \cap M^0) \subseteq W$, *akkor vagy* $\exists t \in W : {}^\bullet t \subseteq M^+$ *vagy létezik olyan sima szifon D, hogy* $W \subseteq D^\bullet$.

Bizonyítás. Tegyük fel, hogy nincs W-ben megengedett t. Ekkor $\forall t \in W : {}^\bullet t \cup M^0 \neq \emptyset$ és $t \in ({}^\bullet t \cap M^0)^\bullet$, így $W \subseteq ({}^\bullet W \cap M^0)^\bullet$, azaz $D = ({}^\bullet W \cap M^0)$ egy sima szifon (${}^\bullet D \subseteq D^\bullet$). ■

Az alábbi lemma szerint szabadválasztású Petri-hálókban a holt átmenetek halmaza jellemezhető egy szifonnal (más néven holtponttal). Megmutatható, hogy létezik olyan elérhető állapot, amelyben egy sima szifon kimenő átmeneteinek halmaza tartalmazza a holt átmenetek halmazát.

24.47. lemma. *Egy* (N, M) *szabadválasztású Petri-hálóban legyen* $W \subseteq T$ *a holt átmenetek részhalmaza (egyetlen M-ből induló akciósorozatban sem szerepel egyetlen W-beli átmenet sem). Ekkor létezik egy olyan M-ből elérhető* M' *súlyozás, amelyben van egy olyan D sima szifon, hogy* $W \subseteq D^\bullet$.

Bizonyítás. A bizonyítást a $T \setminus W$ halmaz számossága szerinti indukcióval végezzük el. Alapeset: tegyük fel, hogy $|T \setminus W| = 0$, azaz minden átmenet holt. Ekkor $W = T$, így ${}^\bullet({}^\bullet W \cap M^0) \subseteq W$. Ha nincs W-ben egyetlen megengedett átmenet sem, akkor a 24.46. lemma alkalmazásával kapjuk, hogy létezik olyan D szifon, amelyre: $W \subseteq D^\bullet$.

Indukciós lépés: legyen $|T \setminus W| > 0$. Legyen a kezdő súlyozás $M_0 = M$. Konstruálunk egy olyan akciósorozatot, amely a $M_1, \ldots, M_i, \ldots, M'$ súlyozásokhoz vezet és létezik egy $D \subseteq M'^\bullet$ sima holtpont és $W \subseteq D^\bullet$.

Először megmutatjuk, hogy egyetlen akciósorozat sem aktivizál $({}^\bullet W)^\bullet$-beli átmenetet. Tegyük fel indirekt módon, hogy létezik olyan $t_0 \in W$ és $p_0 \in {}^\bullet t_0$, hogy valamely $t_1 \in p_0{}^\bullet$ aktivizálható. t_0 holt, így $t_1 \in p_0{}^\bullet \setminus W$. Ez azonban ellentmond a szabadválasztású háló definíciójának, ha t_1 aktivizálható, akkor t_0 is az.

Legyen az aktuális súlyozás M_i. Ha ${}^\bullet({}^\bullet W \cap M_i^0) \subseteq W$, akkor a 24.46. lemma szerint $D = {}^\bullet W \cap M_i^0$ sima holtpont és $W \subseteq D^\bullet$. Ha $M_i = M'$, akkor az állítást kapjuk. Ha ${}^\bullet({}^\bullet W \cap M_i^0) \not\subseteq W$, akkor létezik olyan $t \in {}^\bullet({}^\bullet W \cap M_i^0) \setminus W$, amelyre

a) egyetlen akciósorozatban sem szerepel t, vagy

b) létezik olyan ς akciósorozat, amelyikben t előfordul.

Az a) esetben jelöljük a $W \cup \{t\}$ halmazt W'-vel. Egyetlen akciósorozat sem aktivizál W'-beli átmenetet. $|T \setminus W'| = |T \setminus W| - 1$, ezért az indukciós feltevés szerint van egy olyan ς akciósorozat, amely M'-höz vezet M-ből és van egy $D \subseteq M'^0$ sima holtpont, hogy $W' \subseteq D^\bullet$. Mivel $W \subseteq W'$, ezért a lemmát ebben az esetben bizonyítottuk.

A b) esetben jelöljük M_{i+1}-gyel azt a súlyozást, amelyiket M_i-ből kapunk a ς akciósorozattal. Megmutattuk, hogy egyetlen akciósorozat sem aktiválhat $({}^\bullet W)^\bullet$-beli átmenetet, így ς sem, azaz: $({}^\bullet W \cap M_i^+) \subseteq ({}^\bullet W \cap M_{i+1}^+)$. (Ami súlyozott volt ${}^\bullet W$-ben, az súlyozott marad.) Mivel t ebben az esetben ${}^\bullet W \cap M_i^0$-ba mutat és $({}^\bullet W)^\bullet$-ben ς nem aktivizál, $|{}^\bullet W \cap M_{i+1}^0| < {}^\bullet W \cap M_i^0$. Ismételjük meg az állítást M_{i+1}-re. Valahányszor b) esetet alkalmazzuk, a ${}^\bullet W \cap M_{i+1}^0$ számossága csökken, így végül az a) esethez jutunk. ■

A 24.47. lemmából következik, hogy ha a Petri-hálóban egyetlen szifon sem válik sima szifonná, akkor bármely adott átmenethez van olyan akciósorozat, amelyben az átmenet szerepel. (Legyen $W = \{t\}$.) Ha a szifon súlyozott csapdát tartalmaz, akkor nem válik simává. Egy szabadválasztású háló eleven, ha minden szifon tartalmaz súlyozott csapdát. A tétel megfordítása is teljesül, a bizonyítást a további tételek bizonyításával együtt az érdeklődő Olvasó megtalálhatja az idézett szakirodalomban.

24.48. tétel (szabadválasztású háló elevensége). *Egy szabadválasztású háló akkor és csak akkor eleven, ha minden szifon tartalmaz súlyozott csapdát.*

24.49. tétel (szabadválasztású háló elevensége és biztonságossága). *Egy szabadválasztású háló akkor és csak akkor eleven és biztonságos, ha lefedhető olyan erősen összefüggő állapotgépekkel, amelyeknek mindegyikében pontosan 1 súly van, és minden minimális szifon egy súlyozott erősen összefüggő állapotgép.*

24.50. tétel (aszimmetrikusan választó háló elevensége). *Aszimmetrikusan választó háló eleven akkor (de nem csak akkor), ha minden szifon tartalmaz súlyozott csapdát.*

Gyakorlatok

24.8-1. Mutassunk példát biztonságos és eleven jelzett gráfra, illetve állapotgépre.

24.8-2. Mutassunk példát biztonságos és eleven szabadválasztású hálóra.

24.9. Petri-dobozok

A fejezet eddigi részében a Petri-háló általános fogalmát vizsgáltuk. A továbbiakban kifejezetten párhuzamos programok, illetve elosztott algoritmusok vizsgálatához használható hálókkal, Petri-dobozokkal fogunk foglalkozni.

A Petri-doboz speciális tulajdonságokkal rendelkező címkézett Petri-háló. A címkézett Petri-háló egy speciális címkefüggvénnyel kiterjesztett Petri-háló. A címkefüggvény a háló helyeihez e, i és x címkéket rendel. Az e címke jelöli a háló belépési helyeit, az i a belső helyeket, míg x a kilépési helyeket.

Az átmenetekhez ennél jóval bonyolultabb címkéket, úgynevezett átcímkézéseket rendelünk. A Petri-dobozoknak két alapvető típusa van. A sima doboz, amelyet egy algoritmus vagy program leírására használhatunk, illetve az operátordoboz, amely programok közötti műveletek (programkonstrukciók, átnevezés, szinkronizáció) leírására szolgál. A sima doboz esetén egy átmenet végrehajtása a háló által szimulált program, algoritmus egy adott eseményének vagy több szinkronizált eseményének a végrehajtását jelenti. Így sima dobozok átmeneteihez a címkézés egy eseményzsákot rendel. Ezzel szemben operátordoboz esetén egy átmenet egy teljes programot (az ahhoz tartozó sima dobozt) reprezentál és az átmenethez tartozó címke szerves szerepet játszik az operátordoboz által meghatározott műveletben (lásd a 24.10 alfejezetet).

Ennek megfelelően az átcímkézésnek is két alapvetően különböző fajtája van, a konstans átcímkézés és a nem konstans átcímkézés. A konstans átcímkézés megfeleltethető egy eseményzsáknak, míg a nem konstans átcímkézés olyan zsákokhoz, amelyek eseményzsákokat tartalmazó halmazokból állnak, rendel hozzá eseményzsákokat tartalmazó halmazt.

Jelöljük $mult(A)$-val az A-beli eseményekből képzett zsákok halmazát. Egy $mult(A)$-beli eseményzsákot jellemezhetünk egy zsákfüggvénnyel, amely megadja, hogy melyik eseményből hány darabot tartalmaz.

24.51. definíció (esemény előfordulási száma – zsákfüggvény).
$\mu : A \longrightarrow \mathcal{N}_0$ *zsákfüggvény.* $mult(A) = \{\mu | \mu : A \to \mathbf{N}_0\}$.
Például μ *jelölje az* $\{aabccc\}$ *zsákot, ekkor* $\mu(a) = 2, \mu(b) = 1, \mu(c) = 3$.

Nézzük meg az átcímkézés definícióját formálisan. Legyen Act a primitív események egy előre megadott halmaza (ez a háló által szimulált program vagy algoritmus eseményeinek a halmaza).

24.52. definíció (átcímkézés).
$Lab = mult(Act)$
ρ *átcímkézés egy reláció:*
$\rho \subseteq (mult(Lab)) \times Lab$, *úgy, hogy* $(\emptyset, \alpha) \in \rho$ *akkor és csak akkor, ha* $\rho = \{(\emptyset, \alpha)\}$.

Speciális átcímkézések:
konstans átcímkézés: $\rho_\alpha = \{(\emptyset, \alpha)\}$,
megszorítás: $\rho_{Lab'} = \{(\{\alpha\}, \alpha) | \alpha \in Lab'\}$, csak a $Lab' \subset Lab$ beli eseményeket tartja meg,
identitás: $\rho_{id} = \{(\{\alpha\}, \alpha) | \alpha \in Lab\}$.

Mint látható, a konstans átcímkézést is $(mult(Lab)) \times Lab$ alakban adjuk meg, ám egy $\rho_\alpha = \{(\emptyset, \alpha)\}$ átcímkézés egyértelműen megfeleltethető α-nak. Ennek megfelelően a leírt példákban ρ_α helyett mindig csak α-t fogunk használni.

Ezek után nézzük a címkézett Petri-háló formális definícióját. Ez az előzőekben tárgyalt címkefüggvényt nem számítva csak jelölésekben különbözik az általános Petri-háló definíciójától.

24.53. definíció (címkézett Petri-háló).
A $\Sigma = (S, T, W, \lambda, M)$ *ötös egy* ***címkézett Petri-háló****, ahol* S *a helyek,* T *az átmenetek halmaza,* W *az éleket leíró reláció,* λ *a címkefüggvény,* M *pedig a súlyozás.*
$S \cap T = \emptyset$,
$W : ((S \times T) \cup (T \times S)) \to \mathbf{N}_0$,
$\forall s \in S : \lambda(s) \in \{e, i, x\}$,
$\forall t \in T : \lambda(t)$ *egy* $\rho \subseteq (mult(Lab)) \times Lab$ *átcímkézés,*
$M \subseteq S \times \mathbf{N}_0$.

Az események között párokat definiálhatunk, melyek egy külső környezet felől nézve kioltják egymás hatását. Például ilyen lehet egy fájl megnyitására irányuló kérelem és a fájl megnyitása, vagy elosztott esetben egy üzenet elküldése és fogadása.

24.54. definíció (párokat definiáló függvény). $\hat{} : A \to A$ *függvény párokat definiál* A *felett, bijekció,* $a \neq \hat{a}$ *és* $\hat{\hat{}} = \hat{}^{(-1)}$.

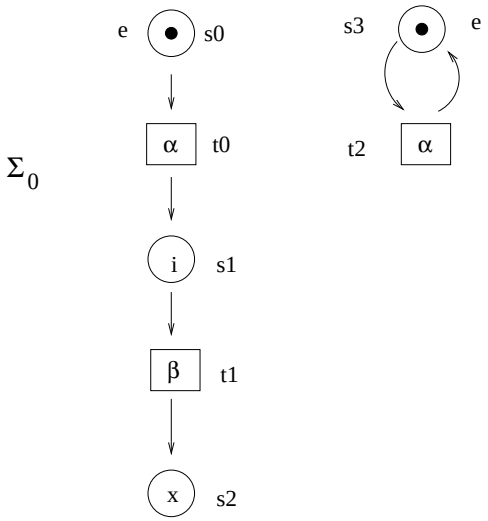
Zsákok esetén az alábbi jelölést használjuk:
Legyen adott egy A eseményhalmaz és egy $\hat{}$ párokat definiáló függvény A felett. $\hat{\mu}(a) ::= \mu(\hat{a})$.

Mint említettük, címkézett Petri-hálók esetén a címkézés alapján a helyeket három különböző halmazba soroljuk, a belépési helyek (${}^\bullet\Sigma$), kilépési helyek ($\Sigma^\bullet$) és a belső helyek ($\ddot{\Sigma}$) halmazába.

Azaz formálisan : $\Sigma = (S, T, W, \lambda, M)$. Legyen $s \in S$. Ha $\lambda(s) = e$ (entry), akkor s belépési hely, $\lambda(s) = x$ (exit), akkor s kilépési hely, $\lambda(s) = i$ (internal), akkor s belső hely, ${}^\bullet\Sigma = \{s \in S | \lambda(s) = e\}$ a belépési helyek halmaza, $\Sigma^\bullet = \{s \in S | \lambda(s) = x\}$ a kilépési helyek halmaza, $\ddot{\Sigma} = \{s \in S | \lambda(s) = i\}$ a belső helyek halmaza.

Valós alkalmazás szimulációja esetén általában az alkalmazás indításakor a belépési helyekre helyezünk súlyt, és az alkalmazás végrehajtásának megfelelő szimuláció során a súlyok a belső helyeken keresztül átkerülnek a kilépési helyekre, ha az összes súly kilépési helyen van, akkor a szimuláció befejeződött. Azonban a címkézett Petri-háló definíciója ennél jóval általánosabb, semmilyen megkötés nincs arra, hogy melyik helyhez milyen címkét rendeljünk. Megkötéseket majd csak a Petri-doboz definíciójánál vezetünk be.

Nézzünk meg most példaként egy címkézett Petri-hálót. A példa négy helyet ($\{s_0, s_1, s_2, s_3\}$) és három átmenetet ($\{t_0, t_1, t_2\}$) tartalmaz. A helyek közül s_0 és s_3 belépési, s_1 belső, s_2 pedig kilépési hely, kezdetben s_0 és s_3 tartalmaz súlyt. A t_0 átmenet α konstans címkével van címkézve, egy megelőző helye (s_0) és egy rákövetkező helye (s_1) van. A t_1 átmenet β konstans címkével van címkézve, egy megelőző helye (s_1) és egy rákövetkező helye (s_2) van. Végezetül pedig a t_2 átmenet egy hurok átmenet, amely α-val van címkézve és a megelőző és a rákövetkező helye is s_3.



24.35. ábra. A 24.2. példában leírt Σ_0 címkézett Petri-háló.

24.2. példa. [címkézett Petri-háló]
$\Sigma_0 = (S_0, T_0, W_0, \lambda_0, M_0)$
$S_0 = \{s_0, s_1, s_2, s_3\}$
$T_0 = \{t_0, t_1, t_2\}$
$W_0 = ((TS \cup ST) \times \{1\}) \cup (((S \times T) \setminus ST \cup (T \times S) \setminus TS) \times \{0\})$
$\lambda_0 = \{(s_0, e), (s_1, i), (s_2, x), (s_3, e), (t_0, \alpha), (t_1, \beta), (t_2, \alpha)\}$
$M_0 = \{(s_0, 1), (s_1, 0), (s_2, 0), (s_3, 1)\}$,
ahol
$TS = \{(t_0, s_1), (t_1, s_2), (t_2, s_3)\}$ és $ST = \{(s_0, t_0), (s_1, t_1), (s_3, t_2)\}$ (lásd 24.35. ábra).

24.9.1. Működési szabály címkézett Petri-hálón

Címkézett Petri-hálók esetén kiterjesztjük az eddigiekben megismert működési szabályt. Általános Petri-hálók esetén egyidejűleg csak egy átmenet hajtódhat végre. Ezzel szemben itt megengedjük, hogy egyidejűleg egyszerre több átmenet, sőt egy átment többször is végrehajtódjon. Ennek megfelelően címkézett Petri-hálók esetén egy lépés nem egy átmenet, hanem egy átmenetzsák végrehajtását írja le. Így itt egy lépés akkor engedélyezett, ha minden helyen van elég súly ahhoz, hogy szinkron végre tudjuk hajtani a zsákban szereplő összes lépést (amelyik többször szerepel, azt annyiszor, ahányszor szerepel a zsákban).

24.55. definíció (engedélyezett lépés).
$\Sigma = (S, T, W, \lambda, M)$
Legyen $U \in mult(T)$ *átmenetek egy véges zsákja, az* U *által meghatározott lépés engedélyezett* Σ*-ban, ha minden* $s \in S$ *helyre:*
$M(s) \geq \sum_{t \in U} U(t) * W(s, t)$.

A lépés végrehajtása során a zsákbeli összes átmenet levesz a megelőző helyeiről az átmenetbe vezető éleknek megfelelő számú súlyt és rárak a rákövetkező helyeire a kivezető éleknek megfelelő számú súlyt (természetesen, ha egy átmenet többször is szerepel a zsákban, akkor ez az előfordulási számnak megfelelően, többszörösen megtörténik).

24.56. definíció (lépés végrehajtása).
Jelölje M' az $U \in mult(T)$ által meghatározott lépés végrehajtása utáni súlyozást.
$\forall s \in S : M'(s) = M(s) + \sum_{t \in U} U(t) * (W(t,s) - W(s,t))$
Jelölés: $M\,[U> M'$, vagy $\Sigma\,[U> \Theta$, ahol $\Theta = (S, T, W, \lambda, M')$.

A háló működése során egy adott pillanatban bármelyik engedélyezett lépés végrehajtható. A különböző tulajdonságok vizsgálata szempontjából számunkra igazából nem is az egyes lépések, hanem az egymás után végrehajtható lépések sorozata, a lépéssorozat a fontos. Ha egy M_0 kezdősúlyozásból kiindulva van olyan végrehajtható lépéssorozat, amely egy adott M súlyozáshoz vezet, akkor azt mondjuk, hogy M elérhető M_0-ból. Hasonlóan, ha van olyan lépéssorozat, amely egy Σ_0 címkézett hálóból egy Σ címkézett hálóba vezet, akkor azt mondjuk, hogy Σ származtatható Σ_0-ból.

24.57. definíció (véges lépéssorozat).
$\sigma = U_1 \ldots U_k$ egy véges lépéssorozat Σ-ban, ha létezik $\Sigma_0, \ldots, \Sigma_k$ címkézett háló, úgy, hogy $\Sigma = \Sigma_0$ és $\forall i \in [1 \ldots k] : \Sigma_{i-1}\,[U_i> \Sigma_i$.
Jelölés: $\Sigma\,[\sigma> \Sigma_k$, vagy $M_\Sigma\,[\sigma> M_{\Sigma_k}$.
Elnevezés: M_{Σ_k} súlyozás elérhető M_Σ-ból ($M_{\Sigma_k} \in [M_\Sigma>$), és Σ_k származtatható Σ-ból ($\Sigma_k \in [\Sigma>$).

Egy hálóról akkor mondjuk, hogy önkonkurenciát tartalmaz, ha megenged olyan lépéssorozatot, amely nem átmenethalmazok sorozata (azaz a sorozat lépéseinek valamelyike valódi zsák).

Nézzünk egy példát végrehajtható lépéssorozatra.

24.3. példa. A 24.2. példabeli Σ_0-ban t_0 és t_2 konkurensen engedélyezett, így $\{t_0, t_2\}$ egy engedélyezett lépés. Miután ezt a lépést végrehajtjuk, t_1 és t_2 konkurensen végrehajthatóvá válik, így a $\{t_0, t_2\}\{t_1, t_2\}$ egy lépéssorozata Σ_0-nak.
Megjegyzés. Egy lépésnek nem kell maximálisnak lennie, például $\{t_0\}$ is egy lépése, illetve $\{t_0\}\{t_1\}\{t_2\}\{t_2\}$ és $\{t_0, t_2\}\{t_1\}\{t_2\}$ is egy lépéssorozata Σ_0-nak.

24.9.2. Címkézett Petri-hálók tulajdonságai

Ebben a részben definiálunk néhány, a címkézett Petri-hálókon értelmezett tulajdonságot. Az első, az úgynevezett *T-megszorítás* azt fogalmazza meg, hogy a háló minden átmenetének van megelőző és rákövetkező helye is, azaz el akarjuk kerülni a forrás és nyelő átmeneteket.

24.58. definíció (T-megszorítás). $\forall t \in T : {}^\bullet t \neq \emptyset \neq t^\bullet$
Ezt a tulajdonságot a továbbiakban minden hálóra feltesszük.

A következő tulajdonság az *ex-megszorítás*, amely akkor teljesül, ha a hálónak létezik legalább egy belépési és egy kilépési helye.

24.59. definíció (ex-megszorítás:). ${}^\bullet\Sigma \neq \emptyset$ *és* $\Sigma^\bullet \neq \emptyset$

Valós alkalmazások esetén a belépési helyekre csak az alkalmazás indításakor akarunk súlyokat helyezni, azaz nem akarjuk, hogy a belépési helyekre vezessen él (*e-irányított* háló). Illetve a kilépési helyekről nem akarunk súlyt elvenni, hiszen ha kerül súly egy kilépési helyre, akkor az az adott rész befejeződését jelenti. Tehát azt akarjuk, hogy egyik kilépési helyről se vezessen ki él (*x-irányított* háló).

24.60. definíció (e-irányított háló). Σ ***e-irányított,*** *ha*
$\forall s \in {}^\bullet\Sigma : \forall t : W(t,s) = 0$.

24.61. definíció (x-irányított háló). Σ ***x-irányított,*** *ha*
$\forall s \in \Sigma^\bullet : \forall t : W(s,t) = 0$.

24.62. definíció (ex-irányított háló). Σ ***ex-irányított,*** *ha e-irányított és x-irányított.*

A másik viszonylag természetes elvárás lehet egy hálóval szemben, hogy ne kerüljön egyszerre súly egy belépési és egy kilépési helyére. Ezt fogalmazza meg az *ex-kizárólagosság*, azzal a megszorítással, hogy mindezt ne csak a kezdősúlyozásból ne lehessen elérni, de azokból a súlyozásokból sem, amelyek csak a belépési ($M_{\bullet\Sigma}$ súlyozás), illetve kilépési helyekre ($M_{\Sigma\bullet}$ súlyozás) helyeznek súlyt.

24.63. definíció (ex-kizárólagos). *Legyen*

$$\forall s \in S : M_{\bullet\Sigma}(s) = \begin{cases} 1, & \text{ha } s \in {}^\bullet\Sigma, \\ 0, & \text{különben}. \end{cases}$$

$$\forall s \in S : M_{\Sigma\bullet}(s) = \begin{cases} 1, & \text{ha } s \in \Sigma^\bullet, \\ 0, & \text{különben}. \end{cases}$$

Σ ***ex-kizárólagos****, ha minden* M_Σ*-ból, vagy* $M_{\bullet\Sigma}$*-ból, vagy* $M_{\Sigma\bullet}$*-ból elérhető* M *súlyozásra* $M \cap M_{\bullet\Sigma} = \emptyset$ *vagy* $M \cap M_{\Sigma\bullet} = \emptyset$.

24.4. példa. A 24.2. példabeli Σ_0 háló ex-megszorított és x-irányított, de nem e-irányított és nem ex-kizárólagos.

A fenti tulajdonságok vizsgálata szempontjából érdekes lehet, hogy a háló tartalmaz-e olyan átmenetet, amelynek vagy a megelőző helyei között vagy a rákövetkező helyei között van belépési és kilépési pont is. Ha van ilyen átmenet egy címkézett Petri-hálóban, akkor biztosan tudjuk, hogy az a háló ex-megszorított, de nem ex-irányított. Abban az esetben pedig, ha az adott átmenet végrehajtható, akkor a háló nem ex-kizárólagos, mivel kell legyen olyan, a kezdősúlyozásból elérhető súlyozás, amely az átmenet minden megelőző helyére helyez súlyt és olyan is, amely az átmenet minden rákövetkező helyére rak súlyt, így a definíció szerint valamelyik súlyozás a kettő közül súlyt fog helyezni belépési és kilépési pontra is. A leírt tulajdonságú átmenetet *ex-aszimmetrikusnak* nevezzük.

24.64. definíció (ex-aszimmetrikus átmenet). *Egy $t \in T$ átmenet **ex-aszimmetrikus**, ha* ${}^\bullet t \cap {}^\bullet\Sigma \neq \emptyset \neq {}^\bullet t \cap \Sigma^\bullet$ *vagy* $t^\bullet \cap {}^\bullet\Sigma \neq \emptyset \neq t^\bullet \cap \Sigma^\bullet$.

Végül a lehetséges lépéssorozatok vizsgálata szempontjából fontos lehet, hogy melyek azok az átmenetek, melyek teljesen függetlenek egymástól. A függetlenséget fogalmazza meg formálisan az alábbi definíció.

24.65. definíció (függetlenség reláció).
$ind_\Sigma = \{(t,u) \in T \times T | ({}^\bullet t \cup t^\bullet) \cap ({}^\bullet u \cup u^\bullet) = \emptyset\}$.

Ha egy Σ címkézett Petri-háló biztonságos, akkor bármely két átmenet, amely megjelenik egy lépésben, *független* a fenti definíció szerint.

A címkézett Petri-hálókon definiálunk néhány műveletet, amelyek a hálók súlyozását módosítják (ezen kívül semmilyen más változtatást nem végeznek). Ezek közül az első elveszi az összes súlyt a hálóból, a második csak a háló belépési helyeire helyez súlyt, míg a harmadik a háló összes kilépési helyére rak súlyt (és sehova máshova nem).

24.66. definíció (súlyozást módosító műveletek).
Legyen $\Sigma = (S, T, W, \lambda, M)$, *ekkor*

$\lfloor\Sigma\rfloor = (S, T, W, \lambda, \emptyset)$

$\overline{\Sigma} = (S, T, W, \lambda, M_{\bullet\Sigma})$

$\underline{\Sigma} = (S, T, W, \lambda, M_{\Sigma\bullet})$.

24.9.3. Petri-doboz definíciója

A Petri-doboz egy címkézett Petri-háló, amely megadott tulajdonságokkal rendelkezik.

24.67. definíció (Petri-doboz). *Σ címkézett Petri-háló **Petri-doboz**, ha ex-megszorított, valamint ex-irányított (és T megszorított).*

Azaz egy Petri-doboztól megköveteljük, hogy legyen legalább egy belépési és egy kilépési pontja, valamint hogy ne vezessen él a belépési pontjaiba és ne vezessen ki él a kilépési pontjaiból. Ezen felül természetesen megköveteljük, hogy minden átmenetének legyen megelőzője és rákövetkezője (mint említettük ezt gyakorlatilag minden címkézett Petri-hálóra feltesszük).

A Petri-dobozok legegyszerűbb fajtája a *sima doboz*, amelynél minden átmenethez konstans átcímkézés van rendelve.

24.68. definíció (sima doboz). *Σ Petri-doboz sima doboz, ha minden $t \in T_\Sigma$ átmenetre a $\lambda_\Sigma(t)$ egy konstans átcímkézés.*

A sima dobozokat egyszerű párhuzamos folyamatok, algoritmusok szimulációjához használhatjuk. Az átmenetekhez rendelt konstans címkék az összetartozó eseményeket jelölik, amelyeket egyszerre hajtunk végre.

Egy valódi folyamat szimulációja esetén elvárhatjuk, hogy ha az összes belépési vagy kilépési helyén van súly, akkor máshol ne legyen. Ha egy súlyozás teljesíti ezt, akkor *tiszta súlyozásnak* nevezzük.

24.69. definíció (tiszta súlyozás). *Egy M súlyozás tiszta, ha nem valódi szuper-zsákja $M_{\bullet\Sigma}$-nak és $M_{\Sigma\bullet}$-nak, azaz, ha $M_{\bullet\Sigma} \subseteq M$, akkor $M_{\bullet\Sigma} = M$, valamint ha $M_{\Sigma\bullet} \subseteq M$, akkor $M_{\Sigma\bullet} = M$.*

A továbbiakban elsősorban olyan sima dobozokkal fogunk foglalkozni, melyeknél az összes elérhető súlyozás biztonságos és tiszta. Az ilyen dobozokon belül is két alapvető típust különböztetünk meg, az elsőnél gyakorlatilag csak a doboz szerkezete a fontos, azaz azok a dobozok tartoznak bele, amelyek nem tartalmaznak súlyt, míg a második típusba azon dobozok tartoznak, melyekhez nem üres súlyozás van rendelve.

24.70. definíció (statikus doboz). *Egy Σ súlyozatlan sima doboz statikus doboz, ha minden $M_{\bullet\Sigma}$-ból és $M_{\Sigma\bullet}$-ból elérhető súlyozás biztonságos és tiszta.*

24.71. definíció (dinamikus doboz). *Egy Σ súlyozott sima doboz dinamikus doboz, ha minden M_Σ-ból, $M_{\bullet\Sigma}$-ból és $M_{\Sigma\bullet}$-ból elérhető súlyozás biztonságos és tiszta.*

Ha Σ egy statikus doboz és Θ származtatható $\overline{\Sigma}$-ból (azaz abból a dobozból, amelyet Σ-ból úgy kapunk meg, hogy minden belépési helyre helyezünk súlyt), akkor Θ egy dinamikus doboz (azaz speciálisan $\overline{\Sigma}$ maga is egy dinamikus doboz). Ugyanakkor a 24.71. definíció nem követeli meg, hogy M_Σ elérhető legyen $M_{\bullet\Sigma}$-ból vagy $M_{\Sigma\bullet}$-ból, ha mégis az, akkor a definícióban az M_Σ-ra vonatkozó rész elhagyható.

A fentieken kívül a sima dobozoknak még két speciális osztályát különböztetjük meg, ezek a *belépési* és *kilépési* dobozok, olyan dinamikus dobozok, amelyeknél a súlyozás megegyezik $M_{\bullet\Sigma}$-val, illetve $M_{\Sigma\bullet}$-val. (A dinamikus dobozok halmaza tartalmazza a *belépési* és a *kilépési* dobozok halmazát.)

Az említett osztályokra a következő jelöléseket vezetjük be:
Statikus dobozok halmaza: Box^s,
Dinamikus dobozok halmaza: Box^d,
Belépési dobozok halmaza: Box^e,
Kilépési dobozok halmaza: Box^x,
Sima dobozok halmaza: Box

Dinamikus dobozokra megfogalmazhatjuk az alábbi egyszerű tételt:

24.72. tétel. *Legyen Σ egy dinamikus doboz és U egy engedélyezett lépés Σ-ban. Ekkor*

1. *minden Σ-ból elérhető címkézett háló egy dinamikus doboz,*
2. *U kölcsönösen független átmenetek egy halmaza:*
   $U \times U \subseteq ind_\Sigma \cup id_{T_\Sigma}$, ahol $id_X = \{(x,x) | x \in X\}$ minden X halmazra,
3. *és minden U-hoz kapcsolódó él egyszeres:*
   $W_\Sigma(U \times S_\Sigma) \cup W_\Sigma(S_\Sigma \times U) \subseteq \{0,1\}$.

A tétel bizonyítását az Olvasóra hagyjuk.

24.9.4. Operátordoboz

A Petri-dobozok másik alapvető fajtája az operátordoboz. Az ilyen dobozokat elsősorban programrészletek közötti műveletek (például programkonstrukciók, szinkronizáció, átnevezés) leírására használhatjuk. Ennek megfelelően az átmeneteihez nem konstans átcímkézéseket rendelünk. A nem konstans átcímkézéseket transzformációs átcímkézésnek nevezzük. Az átcímkézésen kívül az operátordoboz minden átmenetéhez hozzárendelhetünk egy sima dobozt is. Az operátordoboz által leírt műveletet az átmeneteihez rendelt sima dobozokon fogjuk elvégezni.

24.73. definíció (operátordoboz). *Egy Ω **operátordoboz** olyan doboz, melynek minden átmenetéhez transzformációs (azaz nem konstans) átcímkézés van rendelve. Az operátordobozhoz hozzárendelhetünk egy $\Sigma : T_\Omega \to Box$, az Ω átmeneteiről a sima dobozok halmazára képező függvényt. Σ-t rendezett $\Omega - n - es$-nek fogjuk nevezni. $\forall v \in T_\Omega$-ra $\Sigma(v)$-t Σ_v-vel jelöljük.*
Ha Ω átmenethalmaza véges, akkor feltesszük, hogy $T_\Omega = \{v_1, \ldots, v_n\}$ egy tetszőleges, de fix rendezés az átmenethalmazon és ekkor a $\Sigma = \{\Sigma_{v_1}, \ldots, \Sigma_{v_n}\}$ vagy a $\Sigma = \{\Sigma_1, \ldots, \Sigma_n\}$ jelölést használjuk.

Mint említettük, operátordobozok esetén egy-egy átmenet egy teljes sima doboznak, azaz egy teljes programrészletnek, programnak felel meg. Ennek megfelelően korántsem biztos, hogy egy átmenet egy időpillanat alatt végrehajtódik, sőt lehet olyan átmenet, amelynek végrehajtása akár végtelen időt is igénybe vehet (például holtpont, végtelen ciklus). Ezért szükség van annak nyilvántartására, hogy van-e olyan átmenetünk, amely aktivizálódott, de még nem fejeződött be (pillanatnyilag aktív). Ebből a célból vezetjük be a *komplex súlyozás* definícióját.

24.74. definíció (komplex súlyozás). *Egy Ω operátordoboz **komplex súlyozása** egy $\mathcal{M} = (M, Q)$ pár, ahol M egy szabványos súlyozás, Q pedig az Ω-beli aktív átmenetek véges zsákja. M-et az $\mathcal{M}$ súlyozás valós részének, míg Q-t a képzetes részének nevezzük. Egy szabványos M súlyozást az $(M, \emptyset)$ komplex súlyozással lehet reprezentálni.*

Az eddig megismert működési szabályt ki kell terjesztenünk komplex súlyozás esetére. Egy lépés teljes végrehajtása gyakorlatilag megegyezik az eddig megismert végrehajtással. A változás abban áll, hogy a kiterjesztett definíció mellett megengedett, hogy egy lépés csak aktívvá váljon (azaz elkezdődjön a végrehajtása), de ne fejeződjön be, illetve egy éppen aktív lépés bármikor befejeződjön.

24.75. definíció (kiterjesztett működési szabályok). *Egy U lépés engedélyezett az $\mathcal{M} = (M, Q)$ súlyozás mellett, ha M mellett engedélyezett. Ezt $\mathcal{M}[U>$-val jelöljük.*

*Egy engedélyezett U lépés (teljesen) **végrehajtottá** válhat:*
$\mathcal{M}[U > \mathcal{M}'$, ahol $\mathcal{M}' = (M', Q)$, úgy, hogy

$$\forall s \in S : \; M'(s) = M(s) + \sum_{t \in U} U(t) * (W(t, s) - W(s, t)).$$

*Egy engedélyezett U lépés **aktívvá** válhat:*
$\mathcal{M}[U^+ > \mathcal{M}'$, ahol $\mathcal{M}' = (M', Q + U)$, úgy, hogy

$$\forall s \in S : \; M'(s) = M(s) - \sum_{t \in U} U(t) * W(s,t).$$

Valamint egy $U \subseteq Q$ aktív lépés bármikor ***befejezetté*** *válhat:*
$\mathcal{M}[U^- > \mathcal{M}'$, *ahol* $\mathcal{M}' = (M', Q - U)$, *úgy, hogy*

$$\forall s \in S : \; M'(s) = M(s) + \sum_{t \in U} U(t) * W(t,s).$$

általánosan a fenti három esetet együtt a következőképpen jelöljük:
$\mathcal{M}[U : V^+ : Y^- > (M', Q')$, *ha* $Y \subseteq Q$, $Q' = Q + V - Y$

$$\text{és } \forall s \in S : M(s) \geq \sum_{t \in U+V} (U(t) + V(t)) * W(s,t)$$

$$M'(s) = M(s) + \sum_{t \in U+Y} (U(t) + Y(t)) * W(t,s) - \sum_{t \in U+V} (U(t) + V(t)) * W(s,t)$$

Az $\mathcal{M}$-ből elérhető komplex súlyozásokat $[\mathcal{M}>$*-el jelöljük.*

Komplex súlyozások esetére viszonylag egyszerűen kiterjeszthetjük a *biztonságosság*, *korlátosság* és *tisztaság* definícióit.

24.76. definíció (kiterjesztett definíciók). *Egy $\mathcal{M} = (M, Q)$ komplex súlyozás biztonságos, k-korlátos és tiszta, ha megfelelően M biztonságos, k-korlátos és tiszta.*

Ha Ω egy $\mathcal{M} = (M, Q)$ komplex súlyozású operátordoboz, akkor a doboz által definiált művelet sima dobozok bármely Σ rendezett $\Omega - n$-esére alkalmazható feltéve, hogy Σ_v akkor és csak akkor súlyozott, ha $v \in Q$.

Gyakorlatok

24.9-1. Rajzoljunk fel olyan Petri-dobozt, amely ex-kizárólagos.

24.9-2. Bizonyítsuk be a 24.72. tételt.

24.10. Az operátordoboz által definiált művelet, hálófinomítás

Mint említettük, egy operátordoboz mindig valamilyen programrészek közötti műveletet ír le. Egy Ω operátordoboz esetén egy adott Σ rendezett $\Omega - n$-es határozza meg, hogy a doboz által definiált műveletet milyen sima dobozokra kell alkalmazni. A művelet maga két részből áll. Először a doboz átmeneteihez rendelt átcímkézések által meghatározott *interfész váltást* kell elvégezni az adott átmenethez a Σ által hozzárendelt sima dobozon. Ez a műveletrész a sima dobozok átmenetein végez átalakítást, ezáltal megváltoztatható azok szerkezete.

24.77. definíció (interfész váltás). *Legyen Ω egy operátordoboz és Σ egy rendezett $\Omega - n$-es. A Σ-ra vonatkozó Ω szerinti* ***interfész váltás*** *azt jelenti, hogy minden Σ-beli Σ_v-re végrehajtjuk a megfelelő v átmenet $\lambda_\Omega(v)$ átcímkézése által meghatározott interfész váltást.*

A második műveletrészben az így átalakított sima dobozokat kapcsoljuk össze az operátordoboz szerkezete alapján. Ezt a műveletrészt *átmenet finomításnak* nevezzük, mivel az operátordoboz átmeneteit finomítjuk azáltal, hogy az átmenetek helyére a hozzájuk rendelt sima dobozokat illesztjük. A két műveletrészt együtt *hálófinomításnak* nevezzük

24.78. definíció (hálófinomítás). *Legyen Ω egy operátordoboz, Σ pedig egy rendezett $\Omega-n$-es. Σ Ω szerinti hálófinomítását, $\Omega(\Sigma)$-t úgy kapjuk meg, hogy minden Σ-beli Σ_v-re vesszük a megfelelő $\lambda_\Omega(v)$ szerinti interfész váltást, majd az így kapott új hálókat felhasználva elvégezzük az Ω szerinti átmenetfinomítást.*

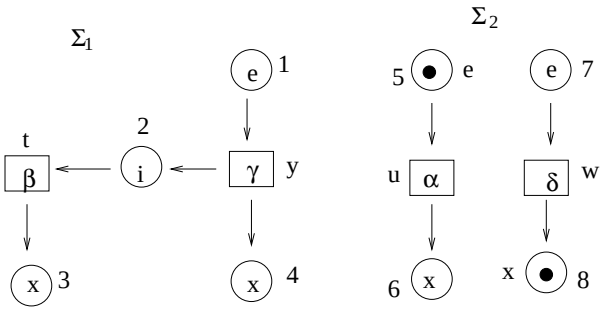
Nézzük meg kicsit részletesebben a két műveletrészt. Először vizsgáljuk meg egy adott Σ_v-re vonatkozó $\rho_v = \lambda_\Omega(v)$ szerinti interfész váltás algoritmusát. Az eljárás csak Σ_v átmeneteit változtatja meg, a helyeket változatlanul hagyja (a súlyozásukkal együtt). Az algoritmus veszi a Σ_v-beli átmenetek által képzett összes lehetséges nem üres halmazt, mivel egy címkét több átmenethez is rendelhetünk, így ezek a halmazok címkezsákokat fognak meghatározni. A kapott címkezsákok közül ki kell választani azokat, amelyek szerepelnek a ρ_v átcímkézés értelmezési tartományában. Ha van ilyen címkezsák, akkor venni kell a hozzá tartozó átmenethalmazt, az ebbe tartozó átmeneteket össze kell vonni egyetlen átmenetté, és a ρ_v által a vizsgált címkezsákhoz rendelt címkét kell rendelni ehhez az összevont átmenethez.

INTERFÉSZVÁLTÁS(Σ_v, ρ_v)

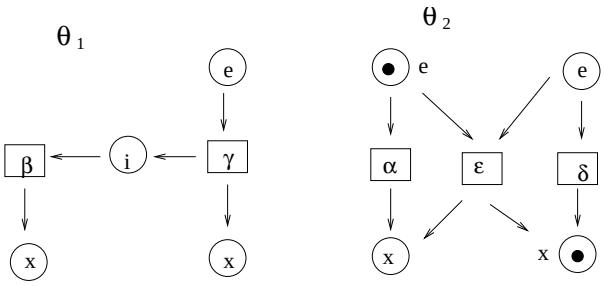
```
 1  legyen T a Σ_v átmeneteinek halmaza
 2  HT ← 2^T \ ∅
 3  Θ_v ← üres doboz
 4  Θ_v ← BEILLESZT-HELYEK(Θ_v, Σ_v)
 5  minden α ∈ HT halmazra
 6        c ← CÍMKEZSÁK(α)
 7        if c ∈ D_ρv
 8           then újátmenet ← ÖSSZEVONT-ÁTMENETET-KÉSZÍT(α)
 9                Θ_v ← BEILLESZT-ÁTMENET(Θ_v, újátmenet)
10  return Θ_v
```

A fenti algoritmusban szereplő BEILLESZT-HELYEK(Θ_v, Σ_v) függvény a Θ_v Petri-dobozba beilleszti a Σ_v doboz minden helyét, annak súlyozásával együtt. A CÍMKEZSÁK(α) függvény meghatározza a paraméterül kapott α halmazbeli átmenetekhez rendelt címkék zsákját. Az ÖSSZEVONT-ÁTMENETET-KÉSZÍT(α) függvény összevonja az α átmenethalmaz elemeit egyetlen átmenetté, mely átmenetnek minden olyan él bemenő éle lesz, amely bemenő éle volt valamelyik α halmazbeli átmenetnek, és hasonlóan minden olyan él kimenő éle lesz, amely él valamely α halmazbeli átmenetnek kimenő éle volt. Végül a BEILLESZT-ÁTMENET(Θ_v, *újátmenet*) függvény beilleszti a paraméterként kapott átmenetet (az összes bemenő és kimenő élével együtt) a Θ_v Petri-dobozba.

24.5. példa. Példaként tekintsük a 24.36. ábrabeli Σ_1 és Σ_2 sima dobozokat, valamint egy olyan operátordobozt (például a 24.38. ábrán látható dobozt), amely két átmenetéhez rendelt átcímkézés: $\rho_1 = \rho_{id}$ (ezt használjuk majd Σ_1-hez), és $\rho_2 = \rho_{id} \cup \{(\{\alpha, \delta\}, \epsilon)\}$ (ezt használjuk majd Σ_2-höz). Itt mivel mindkét címkézés tartalmazza ρ_{id}-et, ezért az eredményül kapott Petri-dobozok tartalmazni fogják az eredeti hálók minden átmenetét. Ezen felül, mivel $(\{\alpha, \delta\}, \epsilon) \in \rho_2$ és a Σ_2 dobozbeli u és w átmenetekhez



24.36. ábra. Σ_1 és Σ_2.



24.37. ábra. Az interfész váltás után Σ_1-ből kapott Θ_1, illetve Σ_2-ből kapott Θ_2.

rendelt címke α és δ, ezért ezen átmenetek összevonásával képezni kell egy új átmenetet (az $\{u, w\}$ halmazhoz tartozó $\{\alpha, \delta\}$ címkezsák eleme lesz D_{ρ_2}-nek), melynek címkéje ϵ (azaz $\rho_2(\{\alpha, \delta\})$) lesz. Az interfész váltás után kapott két háló a 24.37. ábrán látható.

Vizsgáljuk meg most a második műveletrészt, melyben az interfész váltás eredményeként kapott dobozokat kapcsoljuk össze a belépési és kilépési pontjaikon keresztül. Tegyük fel, hogy Ω egy operátordoboz, Σ pedig egy rendezett $\Omega - n$-es (amely a már átalakított dobozokat rendeli Ω megfelelő átmeneteihez). Az átmenetfinomítás során Ω minden helyére meg kell vizsgálni a megelőző és rákövetkező átmeneteket, pontosabban az átmenetekhez rendelt sima dobozokat. Majd az összes lehetséges módon venni kell a megelőző átmenetekhez rendelt dobozok egy-egy kilépési helyét, illetve a rákövetkező átmenetekhez rendelt dobozok egy-egy belépési helyét és egyesíteni kell ezeket a helyeket. Az összekapcsolt dobozok átmenetei, belső helyei, illetve az átmenetek és a belső helyek közötti élek változtatás nélkül átkerülnek az új Petri-dobozba.

ÁTMENETFINOMÍTÁS(Σ, Ω)

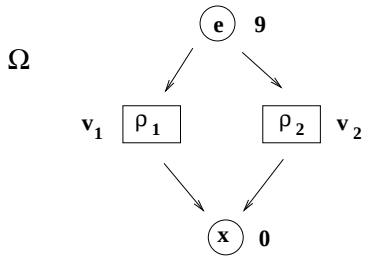
```
1  legyen S az Ω helyeinek halmaza
2  legyen T az Ω átmeneteinek halmaza
3  Θ ← üres doboz
4  minden t ∈ T átmenetre
5          Θ ← BEILLESZT-BELSŐ-HELYEK-ÁTMENETEK(Θ, Σ(t))
6  minden s ∈ S helyre
7          Helyek-direktszorzata ← ∅
8          minden t ∈ •s átmenetre
9                  if Helyek-direktszorzata = ∅
10                   then Helyek-direktszorzata ← •Σ(t)
11                   else Helyek-direktszorzata ← Helyek-direktszorzata × (•Σ(t))
12         minden t ∈ s• átmenetre
13                 if Helyek-direktszorzata = ∅
14                   then Helyek-direktszorzata ← Σ(t)•
15                   else Helyek-direktszorzata ← Helyek-direktszorzata × (Σ(t)•)
16         minden α ∈ Helyek-direktszorzata rendezett n-esre
17                 újhely ← ÖSSZEVONT-HELYET-KÉSZÍT(α)
18                 Θ ← BEILLESZT(Θ, újhely)
19 return Θ
```

Az algoritmusban szereplő BEILLESZT-BELSŐ-HELYEK-ÁTMENETEK(Θ, $\Sigma(t)$) függvény változtatás nélkül beilleszti a $\Sigma(t)$ Petri-doboz minden átmenetét, belső helyét, illetve az átmenetek és a belső helyek közötti éleket a Θ dobozba. Az ÖSSZEVONT-HELYET-KÉSZÍT(α) függvény paramétere egy rendezett n-es, melynek minden komponense egy hely, a függvény összevonja ezeket a helyeket oly módon, hogy minden él, amely bemenő éle valamely komponens helynek, az bemenő éle lesz az új helynek, illetve hasonlóan minden él, amely kimenő éle volt valamely komponens helynek, az kimenő éle lesz az új helynek. A BEILLESZT(Θ, *újhely*) függvény pedig beilleszti a paraméterül adott helyet az összes bemenő és kimenő élével együtt a Θ Petri-dobozba.

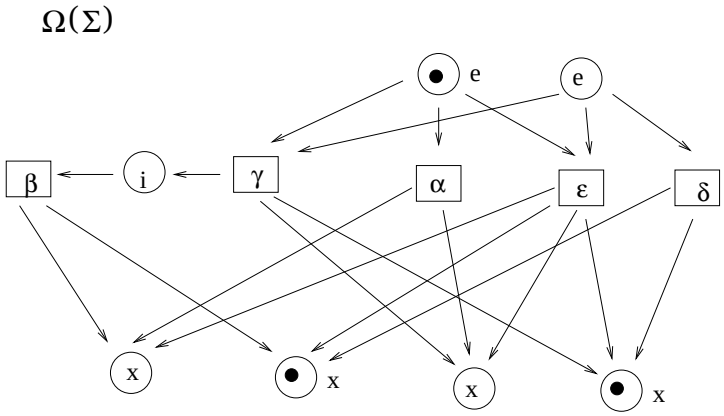
24.6. példa. Példaként tekintsük a 24.37. ábrán szereplő, a 24.5. példabeli interfész váltás után kapott Θ_1 és Θ_2 Petri-dobozokat és a 24.38. ábrán szereplő Ω operátordobozt. Az átmenet finomítás során Θ_1 és Θ_2 bemenő helyeit kell az összes lehetséges módon összekapcsolni (mivel az operátordobozban v_1 és v_2 is rákövetkezője a 9-es címkéjű helynek), illetve hasonlóan Θ_1 és Θ_2 kimenő helyeit is össze kell kapcsolni az összes lehetséges módon (mivel v_1 és v_2 is megelőzője a 0-ás címkéjű helynek). Az eredményül kapott háló a 24.39. ábrán látható.

24.10.1. Speciális operátordobozok

Az operátordobozok elsődleges célja, hogy segítségükkel leírható legyen, hogy egy adott programkonstrukciós műveletnek milyen konstrukciós művelet felel meg a Petri-dobozok szintjén. Ezáltal könnyebbé (sőt bizonyos mértékig automatikussá) válik egy bonyolult programhoz rendelt Petri-háló előállítása. Elegendő megadni a program alapvető alkotó részeihez tartozó sima Petri-dobozokat továbbá azt, hogy ezen programrészekből milyen konst-



24.38. ábra. Ω operátordoboz az előbbi ρ_1 és ρ_2 átcímkézéssel.



24.39. ábra. Ω(Σ).

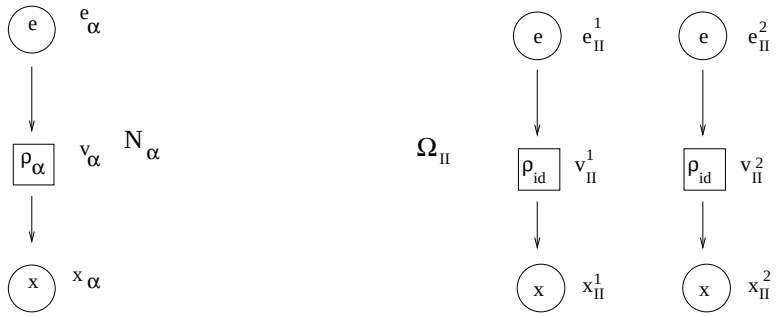
rukciós műveletekkel áll elő a teljes program. Ha minden konstrukciós művelethez adott a megfelelő operátordoboz, akkor a teljes programhoz tartozó Petri-doboz megkapható az alkotórészekhez tartozó dobozokból, elvégezve rajtuk az operátordobozok által definiált hálófinomításokat.

Egy konkrét esetben az *Act* halmaz adja meg a program, algoritmus legalapvetőbb alkotórészeit, eseményeit. Mivel párhuzamos folyamatokról van szó, ezért ezek közül az események közül egyszerre, egy időben több is végrehajtódhat (bizonyos esetekben akár némelyik esemény többször is), ezért ezen programok elemi lépéseit *Act*-beli elemek zsákjával modellezhetjük.

Ebből adódik, hogy az alapvető programrészekhez rendelt Petri-doboz a legtöbbször a 24.40. ábrán látható N_α szerkezetű, úgynevezett *alap doboz.*

24.79. definíció. *Legyen $\alpha \in Lab$ egy esemény, ekkor N_α (lásd 24.40. ábra) egy alap doboz. Az átmenethez rendelt címkézés a $\rho_\alpha = \{(\emptyset, \alpha)\}$ konstans címkézés.*

Érdemes megjegyezni, hogy N_α természetesen egy statikus doboz (azaz nem operátordoboz).



24.40. ábra. a) Az N_α sima doboz. **b)** A párhuzamos kompozíciót leíró Ω_{II} operátordoboz.

Az alapvető egységeknek megfelelő Petri-dobozok definiálása után a továbbiakban az általánosan használt konstrukciókhoz tartozó operátordobozokat definiáljuk. Ezek közül az első a párhuzamos kompozíciót leíró Ω_{II} doboz (lásd 24.40. ábra). Ez az operátordoboz két teljesen különálló másolatot készít a paraméterként adott két sima dobozról (abban az esetben is, ha a két doboz megegyezik), és beilleszti őket egyetlen hálóba.

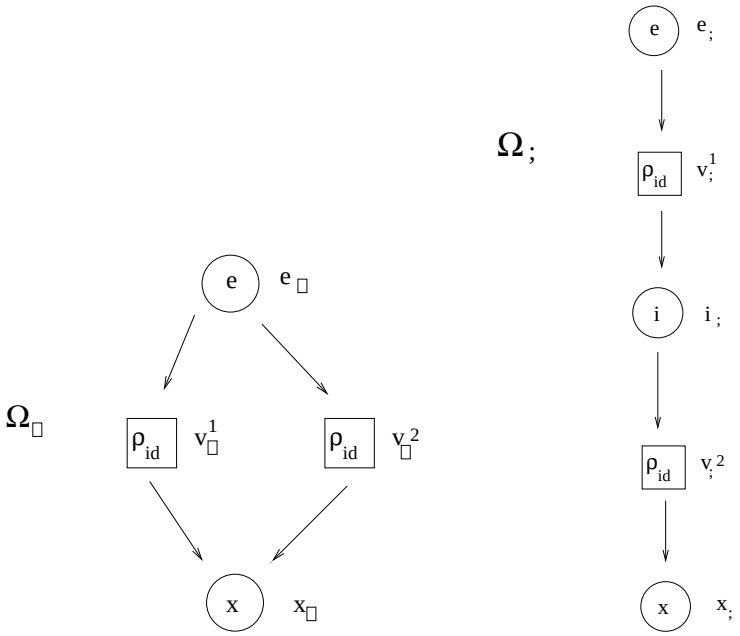
A második operátordoboz, amit definiálunk, az elágazást leíró $\Omega_\square$ doboz (lásd 24.41. ábra). Ennek az operátordoboznak az előbbihez hasonlóan két paramétere van, melyek belépési, illetve kilépési helyeit kapcsolja össze. Ez a doboz hasonlít a 24.6. példabeli operátordobozhoz, azzal a különbséggel, hogy itt minden átmenethez identikus átcímkézést rendelünk, azaz a paraméterként megadott dobozok átmenetein nem változtatunk.

A következő vizsgált operátor az $\Omega_;$ szekvenciális kompozíció (lásd 24.41. ábra), melynek szintén két paramétere van. A művelet az első paraméter kilépési helyeit kapcsolja össze a második paraméter belépési helyeivel, így modellezve a két alkotórész végrehajtásának szekvenciáját.

A vezérlési szerkezet operátorok közül utolsóként nézzük meg a ciklus $\Omega_{[*]}$ operátordobozát (lásd 24.42. ábra). Ez egy olyan ciklust modellez, melyben három különböző utasításunk van. Az első a ciklus inicializáló utasítása, a második a ciklusmag, a harmadik pedig egy lezáró utasítás. Megjegyezzük, hogy ez az operátor nem minden esetben ad biztonságos hálót eredményül, csak az biztosítható, hogy a kapott háló mindig 2-korlátos legyen. Megadható olyan (a bemutatottnál jóval bonyolultabb szerkezetű) ciklus operátor is, amely biztonságos hálót ad, de ennek tárgyalásától most eltekintünk.

Ezek után vizsgáljunk meg néhány olyan operátort, amelynek csak egyetlen paramétere van. Az ilyen dobozok az eddigiekkel ellentétben nem különböző programrészek valamely programkonstrukció szerinti összekapcsolását modellezik, hanem csak egy adott programrészt leíró Petri-dobozon végeznek változtatásokat. A legegyszerűbb ilyen változtatás az $\Omega_{[f]}$ átnevezés (lásd 24.42. ábra), amely csak az átmenetek címkéit változtatja meg. Az operátordobozban szereplő átcímkézés a $\rho_{[f]} = \{(\{\alpha\}, f(\alpha)) | \alpha \in Lab\}$ átcímkézés, ahol $f : Act \to Act$ egy függvény, melyet kiterjesztünk a $Lab = mult(Act)$ halmazra $(\forall \alpha \in Lab : f(\alpha)(f(a)) = \alpha(a))$.

Némileg bonyolultabb művelet a szinkronizáció, amelyet mindig egy adott $a \in Act$ esemény szerint végzünk el. Ez a művelet új, összevont átmenetet készít bármely két olyan átmenethez, amelyre teljesül, hogy közülük az egyikhez olyan címke van rendelve, amely

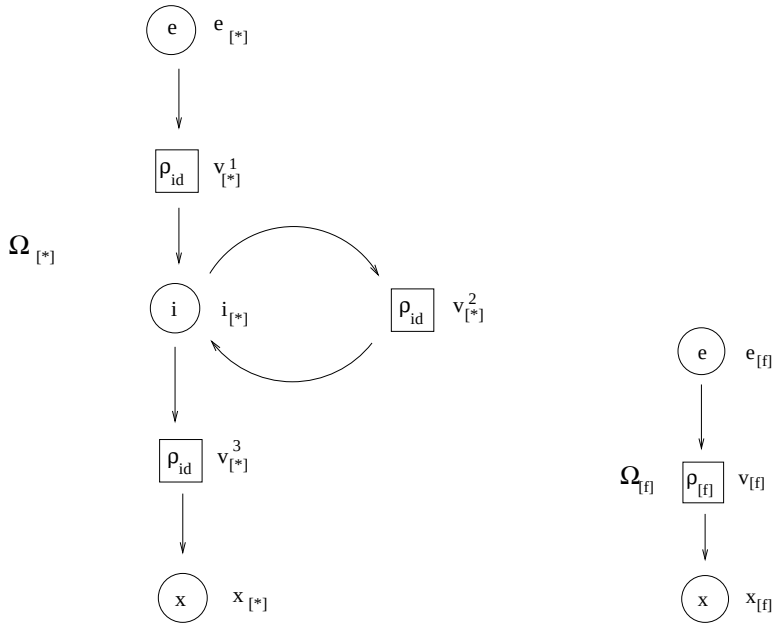


24.41. ábra. a) az elágazást leíró $\Omega_{\square}$ operátordoboz. **b)** a szekvenciális kompozíciót leíró $\Omega_{;}$ operátordoboz.

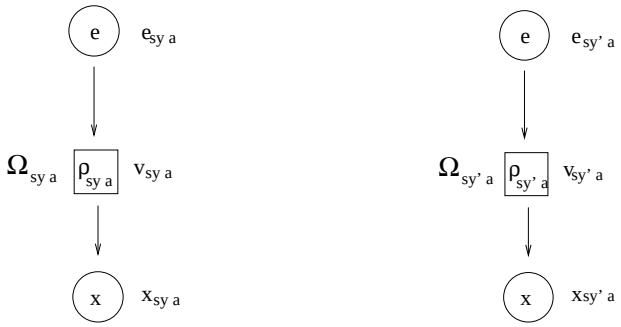
tartalmazza a-t, míg a másikhoz olyan, amelyik tartalmazza $\hat{a}$-t (ahol $\hat{}$ a 24.54. definícióban szereplő, párokat definiáló függvény). Ha az egyik átmenethez rendelt címke $\alpha + \{a\}$, a másikhoz rendelt pedig $\beta + \{\hat{a}\}$, akkor az új átmenethez az $\alpha + \beta$ címkét rendeljük (itt $+$ a zsákokra vonatkozó összeadást jelöli). A művelet az eredeti átmeneteket változatlanul hagyja, addig működik, amíg van két megfelelő átmenet, melyekből még nem készítettünk újat. Rekurzívan működik, egy már újonnan létrehozott átmenethez is készít új összevont átmenetet, ha létezik hozzá megfelelő pár. Egy adott a esemény szerinti szinkronizációt ír le a 24.43. ábrán látható $\Omega_{sy\,a}$ operátordoboz. Itt $\rho_{sy\,a}$ a legszűkebb olyan átcímkézés, amely tartalmazza ρ_{id}-et, és ha $(\Gamma, \alpha + \{a\}) \in \rho_{sy\,a}$ és $(\Delta, \beta + \{\hat{a}\}) \in \rho_{sy\,a}$, akkor $(\Gamma + \Delta, \alpha + \beta) \in \rho_{sy\,a}$.

A következő művelet nagyon hasonló, az egyetlen különbség az, hogy itt egy átmenetet önmagával is szinkronizálhatunk, azaz ha létezik olyan átmenet, melyhez rendelt címke $\alpha + \{a, \hat{a}\}$ alakú, akkor az átmenet alapján létrehozunk egy új átmenetet α címkével. A műveletet önszinkronizációnak nevezzük (lásd 24.43. ábra, $\Omega_{sy'\,a}$ háló). A műveletben szereplő $\rho_{sy'\,a}$ átcímkézés a legszűkebb olyan átcímkézés, amely tartalmazza $\rho_{sy\,a}$-t, és ha $(\Gamma, \alpha + \{a, \hat{a}\}) \in \rho_{sy'\,a}$, akkor $(\Gamma, \alpha) \in \rho_{sy'\,a}$.

Vizsgáljuk meg ezek után a megszorítás műveletét (lásd 24.44. ábra), melyet szintén egy adott $a \in Act$ esemény szerint végzünk el. A művelet eltávolítja a paraméteréül adott sima dobozból az összes olyan átmenetet, melynek címkéjében szerepel a vagy $\hat{a}$. Itt $\rho_{rs\,a} = \rho_{mult(A \setminus \{a,\hat{a}\})}$. Megjegyezzük, hogy a megszorítás során a hálóból akár az összes átmenet törlődhet (például lásd 24.44. ábrán látható Stop doboz).



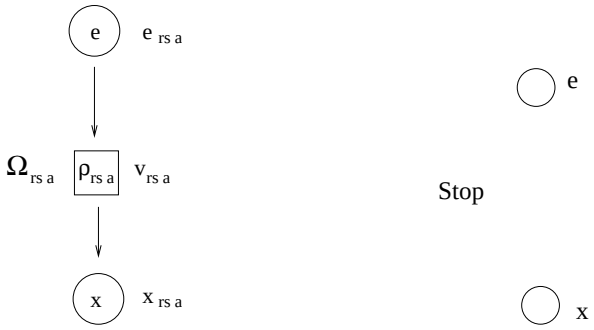
24.42. ábra. a) a ciklust leíró $\Omega_{[*]}$ operátordoboz. **b)** az átnevezést leíró $\Omega_{[f]}$ operátordoboz.



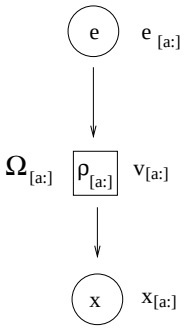
24.43. ábra. a) az a esemény szerinti szinkronizációt leíró $\Omega_{sy\,a}$ operátordoboz. **b)** az a esemény szerinti önszinkronizációt leíró $\Omega_{sy'\,a}$ operátordoboz.

Végezetül vizsgáljuk meg a hatókör operátort (lásd 24.45. ábra), amelyet ugyancsak egy adott $a \in Act$ esemény szerint végzünk el, és hatását tekintve megegyezik a szinkronizáció és a megszorítás műveletének kompozíciójával. Az operátordobozban szereplő átcímkézés a $\rho_{[a:]} = \{(\Gamma, \alpha) \mid \alpha(a) = \alpha(\hat{a}) = 0 \wedge (\Gamma, \alpha) \in \rho_{sy\,a}\}$ átcímkézés.

A definiált operátordobozok alapján létrehozhatunk egy olyan absztrakt nyelvet, melyben a dobozok által modellezett operátorok a megengedettek. Így, ha egy programot vagy



24.44. ábra. a) Az a esemény szerinti megszorítás műveletet leíró $\Omega_{rs\ a}$ operátordoboz. **b)** Átmenet nélküli Stop doboz.



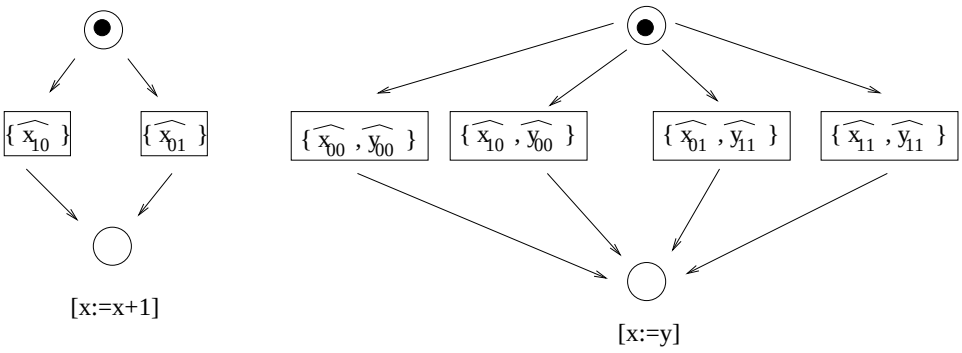
24.45. ábra. Az a esemény szerinti hatókör műveletet leíró $\Omega_{[a:]}$ operátordoboz.

algoritmust le tudunk írni ezen az absztrakt nyelven, akkor a hozzá tartozó Petri-háló automatikusan generálható az operátordobozok által leírt műveletek segítségével. Nézzük meg ennek az absztrakt nyelvnek a pontos definícióját.

24.80. definíció (Petri-doboz kifejezések szintaxisa).

$E ::= \alpha \mid E \| E \mid E \Box E \mid E; E \mid [E * E * E] \mid$
$X \mid E$ `sy` $a \mid E[f] \mid E$ `rs` $a \mid [a : E]$.

A definícióban $\alpha \in Lab$ egy alap esemény, amelyhez egy alap doboz rendelhető. Az operátorokat két részre bonthatjuk a három bináris operátor (a $\|$ párhuzamos operátor, az $\Box$ elágazás, a ; szekvencia) és a 3-áris $[**]$ ciklus operátor *vezérlő szerkezet operátor.* Míg az unáris operátorok, azaz az $[f]$ alap átnevezés, a *sy* a szinkronizáció, a *rs* a megszorítás és a $[a :]$ hatókör operátor *kommunikációs interfész operátorok*. Ezekhez értelemszerűen az előzőleg leírt operátordobozokat rendelhetjük hozzá. Végezetül az X egy változó az előre definiált változóhalmazból, melyhez egyértelműen hozzárendelődik egy E Petri-doboz kifejezés. A változók a különböző szintű absztrakciót segítik elő.



24.46. ábra. [x:=x+1] || [x:=y] ábrázolása.

24.10.2. Programok modellezése

Ebben a részben megvizsgáljuk, hogyan lehet Petri-hálók segítségével modellezni konkrét programokat. Tekintsük először az alábbi egyszerű párhuzamos programot.

24.7. példa.

```
...
 begin var x:{0,1};
  [x:=x+1] || [x:=y]
 end
...
```

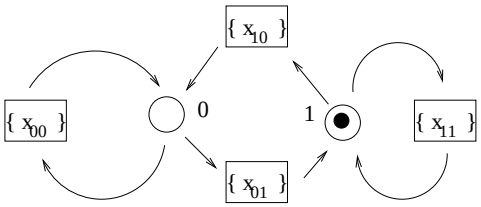
A programban az egyszerűség kedvéért olyan változókat használunk, melyek a {0,1} értékeket vehetik fel és feltételezzük, hogy y egy korábban deklarált (szintén $\{0,1\}$ halmazbeli) változó. A változók típusából adódóan az összeadás modulo 2 értendő. A [] jel az atomi, felbonthatatlan akciókat jelöli.

A programot legegyszerűbben úgy modellezhetjük, hogy bevezetjük az $x_{vw}, \hat{x_{vw}}$ akciókat, ahol $v,w \in \{0,1\}$. Egy ilyen akció átállítja az x értékét v-ről w-re, illetve, ha $v = w$, akkor csak lekérdezi, hogy valóban v a változó értéke. A ˆ-vel megjelölt akciókat elképzelhetjük úgy, mint egy kérelmet (például az operációs rendszer felé) a leírt akció valós, fizikai végrehajtására, a jelöletleneket pedig, mint egy ilyen valós végrehajtást.

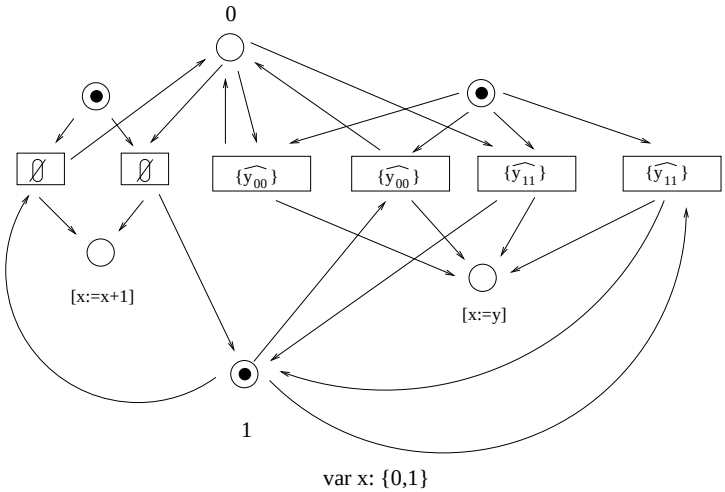
Ezek után a vizsgált program atomi akcióit a következőképpen írhatjuk át általunk már könnyen kezelhető akciókká:
$[x := x + 1] \rightsquigarrow \{\hat{x_{01}}\}\square\{\hat{x_{10}}\}$,

$[x := y] \rightsquigarrow \{\hat{x_{00}}, \hat{y_{00}}\}\square\{\hat{x_{10}}, \hat{y_{00}}\}\square\{\hat{x_{01}}, \hat{y_{11}}\}\square\{\hat{x_{11}}, \hat{y_{11}}\}$.

Az x_{vw} és $\hat{x_{vw}}$ akciókhoz alap dobozokat rendelünk. A leírt operátordobozok műveleteinek végrehajtása után megkapjuk az $[x := x + 1]|||[x := y]$ programhoz tartozó Petri-hálót (lásd 24.46. ábra).



24.47. ábra. A bináris x változó ábrázolása.



24.48. ábra. A program működését leíró háló.

Ahhoz azonban, hogy egy program valós, fizikai működését modellezni tudjuk, vizsgálni kell az egyes változók viselkedését is. Azaz azt is modelleznünk kell, hogy egy változó milyen értékeket vehet fel, és milyen változtatásokat végezhetünk rajta (hogyan változtathatjuk meg az értékét). Az $x \in \{0, 1\}$ esetén ezt írja le a 24.47. ábra, amelynél az egyszerűség kedvéért azt feltételezzük, hogy a változó kezdeti értéke 1.

A valódi működést modellező hálót úgy kaphatjuk meg a 24.46. ábrán és a 24.47. ábrán látható hálóból, hogy vesszük a két háló párhuzamos kompozícióját és az így kapott hálóra elvégezzük mindegyik x_{vw} atomi akció szerint a hatókör operátort. A végeredményként kapott hálót mutatja be a 24.48. ábra.

A bemutatott programhoz nem valós programozási nyelvet használtunk, de a leírt kód könnyen implementálható. Most definiálunk egy olyan absztrakt programozási nyelvet, melyben leírt utasításokra teljesül egyrészt, hogy könnyen implementálhatóak valós konkurens programozási nyelvekkel, másrészt hogy szemantikájuk könnyen megadható Petri-doboz kifejezésekkel. Ezt az absztrakt nyelvet *Razor* nyelvnek fogjuk nevezni.

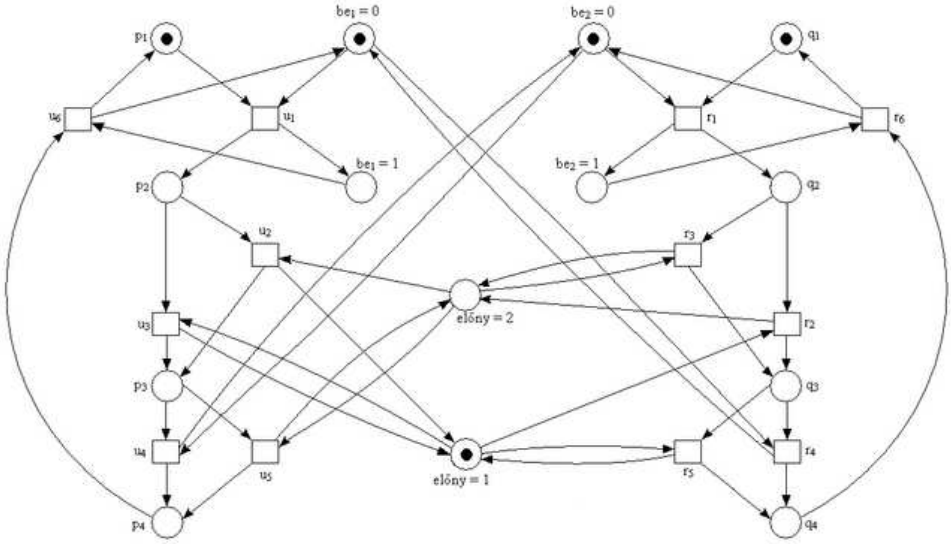
24.81. definíció (*Razor* konkurens programozási nyelv szintaxisa).

1. $Program ::= Block$
2. $Block ::= \texttt{begin}\ Body\ \texttt{end}$
3. $Body ::= Decl; Body \mid Com$
4. $Decl ::= \texttt{var}\ id : Set \mid \texttt{var}\ chanid : \texttt{chan}\ Capacity\ \texttt{of}\ Set$
5. $Com ::= Block \mid [Act] \mid Com_1; Com_2 \mid Com_1 \| Com_2 \mid$
   $\texttt{if}\ GC_1 \square \ldots \square GC_m\ \texttt{fi} \mid \texttt{do}\ GC_1 \square \ldots \square GC_m\ \texttt{od}$
6. $Act ::= id := Expr \mid chanid?x \mid chanid!Expr$
7. $GC ::= GC; Com \mid [Bexpr] \mid [Bexpr; Act]$
8. $Expr ::= id \mid Const \mid Expr\ Binop\ Expr \mid Unop\ Expr \mid Bexpr$
9. $Bexpr ::= boolid \mid \texttt{false} \mid \texttt{true} \mid$
   $Bexpr\ Boolop\ Bexpr \mid \neg Bexpr \mid Expr\ Relop\ Expr$
10. $Binop ::= + \mid - \mid * \mid \texttt{mod} \mid \texttt{div}$
11. $Unop ::= + \mid -$
12. $Boolop ::= \wedge \mid \vee \mid \rightarrow$
13. $Relop ::= = \mid \neq \mid < \mid \leq$

Mint a definícióban látható, a *Razor* nyelv blokkokból áll. Egy blokk tartalmazhat deklarációkat, beágyazott blokkokat, atomi utasításokat, szekvenciát, párhuzamos utasítást, elágazást és ciklust. Az elágazás és a ciklus őrfeltételes utasításokat tartalmaz (GC), amelyek mindenképpen egy (nulladrendű) logikai kifejezéssel kezdődnek (vagy egyszerűen csak egyetlen logikai kifejezésből állnak) és csak akkor hajthatóak végre, ha a megadott logikai feltétel teljesül. A nyelvben egy speciális típust vezetünk be a párhuzamos folyamatok közötti kommunikáció leírására. Ez a típus a csatorna. A csatornán kétféle műveletet végezhetünk, elküldhetünk rá egy kifejezés által definiált értéket, vagy levehetünk róla egy értéket egy változóba. Ezek a műveletek atomiak. Ezen kívül csak egyetlen atomi műveletet vezetünk be, az értékadást, amelynek során egy kifejezés által definiált értéket teszünk egy változóba. Egy kifejezésben csak az összeadás, kivonás, szorzás, osztás, modulo képzés megengedett.

Egy *Razor* nyelvbeli kifejezést könnyedén át lehet írni Petri-doboz kifejezéssé. A pontos szabályokra jelen fejezetben nem térünk ki, de megjegyezzük, hogy valójában csak az őrfeltételes utasítás és a csatornaműveletek esetén kell átírási szabályokat megfogalmaznunk, hiszen a többi elem (részben az előzőekben tekintett, 24.46. ábrán bemutatott példa alapján) automatikusan megfeleltethető egy-egy Petri-doboz kifejezésbeli elemnek.

Látható, hogy a definiált nyelv még mindig igen leegyszerűsített. Mégis ezen a nyelven már nagyon sok algoritmus formalizálható. Vizsgáljuk meg példaként a kölcsönös kizárást,



24.49. ábra. A Peterson-algoritmust reprezentáló Petri-háló.

ahol két folyamat ugyanazt a közös erőforrást próbálja meg használni a *KritikusSzakasz* műveletrészben (például ugyanarra a nyomtatóra próbálnak írni, vagy ugyanazt a fájlt megnyitni). Ezért garantálnunk kell, hogy egyszerre csak az egyikük léphessen be ebbe a végrehajtási szakaszába. Erre a problémára ad megoldást Peterson jól ismert algoritmusa. Az algoritmust a következőképpen lehet leírni *Razor* nyelven.

begin
var be_1, be_2 : {0, 1} (**init** 0); *előny* : {1, 2} (**init** 1);

do		**do**	
$[be_1 := 1];$	a_1	$[be_2 := 1];$	b_1
$[előny := 1];$	a_2	$[előny := 2];$	b_2
if $[be_2 = 0]$		**if** $[be_1 = 0]$	
$\square\ [(előny \neq 1)]$ **fi**;	a_3	$\square\ [(előny \neq 2)]$ **fi**;	b_3
$KritikusSzakasz_1$;		$KritikusSzakasz_2$;	
$[be_1 := 0];$	a_4	$[be_2 := 0];$	b_4
od		**od**	

end

Az algoritmusnak megfelelő Petri-háló automatikusan generálható. Az eredményül kapott hálót mutatja be egyszerűsített formában a 24.49. ábra. Az ábrán az u_1 átmenet felel meg az algoritmusbeli a_1 műveletnek, az u_2 és u_3 átmenet az a_2 műveletnek, az u_4 és u_5 átmenet az a_3 műveletnek, és az u_6 átmenet az a_4 műveletnek. A $KritikusSzakasz_1$ programrészbe való belépést pedig a p_4 hely modellezi. Szimmetrikusan az r_j átmenetek felelnek meg a megfelelő b_i műveletnek és a $KritikusSzakasz_2$ programrészbe való belépést a q_4 hely modellezi.

Az így kapott háló segítségével megvizsgálhatjuk az algoritmus különböző tulajdonságait. Példaként bebizonyítjuk, hogy az algoritmus valóban teljesíti azon feltételt, hogy a két párhuzamos folyamat nem léphet be egyszerre a kritikus szakaszába. Ez a Petri-háló szintjén azt jelenti, hogy nem lehet egyszerre súly a p_4 és q_4 helyek mindegyikén. Ennek bizonyításához először tekintsük a következő helyhalmazokat.
$\theta_1 = \{p_1, p_2, p_3, p_4\}$,
$\theta_2 = \{q_1, q_2, q_3, q_4\}$,
$\theta_3 = \{előny = 1, előny = 2\}$,
$\theta_4 = \{be_1 = 0, be_1 = 1\}$,
$\theta_5 = \{be_2 = 0, be_2 = 1\}$,
$\theta_6 = \{be_1 = 0, p_2, p_3, p_4\}$,
$\theta_7 = \{be_2 = 0, q_2, q_3, q_4\}$.

Könnyen belátható, hogy ezek a halmazok helyinvariánsokat alkotnak, méghozzá a kezdeti súlyozás miatt oly módon, hogy az egyes halmazokban lévő helyek közül pontosan az egyiken van súly. Emellett az is könnyen belátható, hogy a következő két helyhalmaz egy-egy csapdát alkot, melyekben kezdetben egy-egy súly van.

$F_1 = \{be_1 = 0, p_2, előny = 1, q_3\}$,
$F_1 = \{be_2 = 0, q_2, előny = 2, p_3\}$.

Ezek után tegyük fel, hogy valamely, a kezdeti súlyozásból elérhető M súlyozás a p_4 és q_4 helyek mindegyikére helyez súlyt. Felhasználva, hogy θ_6 és θ_7 helyinvariáns, ebből az következik, hogy a többi hely, amely a θ_6 vagy θ_7 halmazban van, nem tartalmaz súlyt, azaz

$$M(be_1 = 0) = M(p_2) = M(p_3) = M(be_2 = 0) = M(q_2) = M(q_3) = 0\ .$$

Mivel θ_3 is helyinvariáns, ezért az $előny = 1$, illetve $előny = 2$ helyek közül csak az egyiken lehet súly. Ebből viszont az következik, hogy az F_1, F_2 csapdák valamelyike üres, ami ellentmond annak, hogy egy olyan helyhalmaz, ami csapdát alkot és kezdetben legalább egy súlyt tartalmaz, nem válhat súlyozatlanná egy Petri-háló működése során. Tehát végeredményképpen azt kaptuk, hogy nem létezik olyan elérhető súlyozás, amely a p_4 és q_4 helyek mindegyikére helyez súlyt, azaz Peterson-algoritmusa valóban teljesíti a kölcsönös kizárás követelményét.

Petri-hálók viselkedését a PEP eszköz segítségével automatikusan is vizsgálhatjuk. Ez az eszköz képes egy Petri-doboz kifejezés alapján elkészíteni egy Petri-hálót, tudja szimulálni annak működését, és bizonyos tulajdonságok (például elérhetőség, holtpontmentesség) vizsgálatára is használható. Ezen felül lineáris, illetve elágazó idejű temporális logikai kifejezések teljesülését is képes ellenőrizni. Például a Peterson-algoritmus esetén a PEP eszköz segítségével automatikusan megvizsgálhatjuk, hogy teljesülhet-e a $\neg(\Diamond(p_4 \wedge q_4))$ temporális logikai kifejezés. A $\Diamond$ temporális logikai operátor jelentése informálisan: „valaha a jövőben teljesül”, így az előbbi formula pontosan az általunk vizsgált tulajdonságot írja le, hogy sohasem lehet a p_4 és q_4 helyeken egyszerre súly.

Gyakorlatok

24.10-1. Rajzoljuk fel, hogy a párhuzamos, illetve a szekvenciális kompozíciót alkalmazva a 24.36. ábrán látható hálókra milyen Petri-hálót kapunk. Ehhez használjuk fel a 24.40. ábra,

illetve a 24.41. ábra jobb oldalán található operátordobozokat.

24.10-2. Írjuk fel a *Razor* nyelven megadott Peterson-algoritmust Petri-doboz kifejezésekkel.

Megjegyzések a fejezethez

A Petri-háló fogalmát C. A. Petri vezette be 1962-ben [278]. Az eredeti modellt számos irányban továbbfejlesztették, például a színezett Petri-hálók, vagy a többszintű hálók bevezetésével. Ezen kiterjesztések közül többet is ismertet [266].

Az alapfogalmak bemutatásánál elsősorban Bagyinszkiné Orosz Anna jegyzetére [31] és T. Murata összefoglaló cikkére [257] támaszkodtunk. Az érdeklődő Olvasó megtalálja a 24.48. és 24.49. tételek bizonyítását [31]-ben. A Petri-hálók és a formális nyelvek kapcsolatát részletesen tárgyalja [32].

Az elosztott programok tulajdonságainak vizsgálatára bevezetett Petri-dobozok modelljét az azt kidolgozó E. Best és munkatársainak cikkei [42] és könyve [41], illetve az ezek alapján készült Peptool szimulációs és elemző rendszer alapján ismertettük. A fejezetben bemutatott algoritmusokat modellező Petri-hálókat is Peptool [275] segítségével készítettük. A hálókat leíró Peptool fájlokat a hálózaton keresztül hozzáférhetővé tesszük az érdeklődő Olvasó számára [287], aki ily módon az algoritmusok működését szimulációs környezetben maga is könnyen kipróbálhatja és elemezheti.

Elosztott algoritmusok Petri-hálókkal történő modellezése során gyakran teljesül az, hogy egy-egy helyen egynél több súly egyszerre sohasem lehet [295]. A kapacitáskorláttal rendelkező hálók azonban mindig helyettesíthetőek ekvivalens hálóval (24.6. tétel).

Helyinvariánsok (P-invariáns) és T-invariánsok lineáris algebrai eszközökkel történő meghatározását írja le [266].

Ez a fejezet a T037742 sz. OTKA pályázat és az MTA Bolyai János Kutatási Ösztöndíj keretében készült.

VIII. FOLYTONOS OPTIMALIZÁCIÓ

Bevezetés

Az első kötetben a *Játékelmélet* fejezet képviselte a folytonos optimalizációt.

Ebben a kötetben ehhez két újabb témakör járul: egyrészt az operációkutatás ígéretes új irányzata, a *belsőpontos programozás,* másrészt pedig a gyakorlatban széles körben alkalmazható *tömegkiszolgálsái módszerek*.

25. Belsőpontos algoritmusok

A lineáris programozás – napjainkban is – számos területen a legjobb modellezési eszközt biztosítja. Gazdasági, ipari, logisztikai és tudományos kérdések sokaságát lehet pontosan (vagy közelítőleg) lineáris egyenletekkel és egyenlőtlenségekkel modellezni.

A lineáris programozás hosszú múltra tekint vissza. Az egyik alapvető és napjainkban legtöbbször hivatkozott eredménye Farkas Gyula 1894-ben közölt lemmája. Farkas Gyula dolgozata a Fourier-féle mechanikai elv alkalmazásairól szólt. A lineáris programozás fejlődése során később is előfordult, hogy gyakorlati feladatok megoldása késztette a kutatókat újabb és újabb algoritmusok bevezetésére, illetve elméleti eredmények igazolására. Dantzig első – lineáris programozással kapcsolatos – közleménye csak 1948-ban jelent meg, annak ellenére, hogy a ***szimplex algoritmust*** már 1947-ben megfogalmazta és gyakorlati feladatok megoldására is kipróbálták. Egyes tudósok visszaemlékezései alapján tudjuk, hogy mind a brit, mind pedig az amerikai hadseregben használtak operációkutatási (lineáris programozási) módszereket a II. világháború alatt szállítási, logisztikai és más hasonló feladatok megoldására.

Hacsián[1] 1979-ben publikálta az ún. ***ellipszoid módszerét,*** amely akkoriban az elméletileg leghatékonyabb (polinomiális) módszer volt. Gyakorlati hatékonysága a szimplex algoritmusétól messze elmaradt. Karmarkar 1984-ben közölte ***projektív skálázású primál algoritmusát,*** amellyel elindította a belsőpontos algoritmusok fejlődésének az évtizedét. Számos, az 50-es és 60-as években bevezetett és akkoriban a gyakorlat szempontjából használhatatlan, lineáris programozási algoritmust (logaritmikus barrier módszer, affin skálázású algoritmus) fedeztek fel újra és tökéletesítettek. A 90-es évek közepére, a kor számítástechnikai lehetőségeit maximálisan kihasználva, a belsőpontos algoritmusok hatékony és numerikusan stabil számítógépes megvalósításait fejlesztették ki, amelyek napjaink optimalizálási szoftvereinek (CPLEX, XPRESS-MP stb.) nélkülözhetetlen részét képezik.

Ebben a fejezetben az a célunk, hogy megfogalmazzuk a lineáris programozás néhány primál-duál belsőpontos algoritmusát és elemezzük azok elméleti hatékonyságát. Ezt megelőzően röviden összefoglaljuk az ehhez szükséges lineáris programozási ismereteket. A fejezetben bemutatásra kerülő ismeretek megértéséhez elemi lineáris algebrai és analízisbeli ismeretekre lesz szükség. Természetesen az algoritmusok egy olyan variánsát is ismertetjük (nagy lépéses primál-duál logaritmikus barrier módszer), amelyet leggyakrabban valósítottak meg az elmúlt években.

[1] Hacsián nevének többféle írásmódjával találkozhatunk a szakirodalomban. A fejezetben a magyar megfelelőjét használjuk, ám az irodalomjegyzékben a Khacijan átírással találkozik az Olvasó, mivel az angol szakirodalomban ez az elfogadott.

A fejezet olvasásához a következő útmutatást nyújtjuk. Feltételezzük, hogy az Olvasó egyetemi szintű ismeretekkel rendelkezik lineáris algebrából és valós analízisből. A konkrét belsőpontos algoritmusokról szóló és a megvalósítást segítő megjegyzéseket tartalmazó 25.3.1.–25.3.4. pontok önmagukban is érthető egységet alkotnak. Így azon Olvasók, akik a belsőpontos algoritmusok egy hatékonyan megvalósítható változatát szeretnék a lehető leggyorsabban megismerni, ezen alfejezetek olvasására szorítkozhatnak. Akik szeretnének az elméleti alapokkal is, a dualitás elmélettel és a centrális út elméletével megismerkedni, a 25.1. alfejezet alapján betekintést kapnak ebbe. Azon olvasók, akik a belsőpontos algoritmusok egy egyszerű változatával, teljes lépésszám elemzéssel, az algoritmusok lépésszámára adott elméleti felső korlát levezetésével kívánnak megismerkedni, valamint a kerekítési eljárás iránt érdeklődőknek, ami leírja, hogy közelítő megoldásból miként kaphatunk pontos optimális megoldást, a Dikin-algoritmussal foglalkozó 25.2. alfejezetet ajánljuk. Végül az utolsó, 25.3.5. pont napjaink egyik fő kutatási irányát ismerteti.

25.1. A lineáris programozás alapvető tételei

Ebben a részben megfogalmazzuk az általános primál- és duál lineáris programozási feladatot, és a gyenge, illetve az erős dualitás tételeket bizonyítjuk. Ezután a Goldman–Tucker modellt és a speciális önduális lineáris programozási feladatot vezetjük be, majd a centrális út legfontosabb tulajdonságait tárgyaljuk. A Goldman–Tucker és a Sonnevend-tétel bizonyítása után kitérünk a lineáris programozási feladatokhoz tartozó Newton-rendszer megoldására is. Ezekkel az eredményekkel építjük fel a belsőpontos algoritmusok elméletét. Végezetül a több mint 100 éves Farkas-lemmára a Goldman–Tucker modell felhasználásával adunk elemi bizonyítást.

Gyenge dualitás tétel
Az ***általános primál*** (P) és ***duál*** (D) ***lineáris programozási feladatot*** az alábbi kanonikus alakban tekintjük:

$$\begin{aligned} &(P) \qquad \min\left\{\mathbf{c}^T\mathbf{x} \;:\; A\mathbf{x}\geq\mathbf{b},\;\; \mathbf{x}\geq\mathbf{0}\right\}, \\ &(D) \qquad \max\left\{\mathbf{b}^T\mathbf{y} \;:\; A^T\mathbf{y}\leq\mathbf{c},\;\; \mathbf{y}\geq\mathbf{0}\right\}, \end{aligned}$$

ahol $A\in\mathbb{R}^{m\times k}$ mátrix, $\mathbf{b},\mathbf{y}\in\mathbb{R}^m$ és $\mathbf{c},\mathbf{x}\in\mathbb{R}^k$. Legyen a ***primál,*** illetve a ***duál megengedett megoldások*** halmaza rendre

$$\mathcal{P}=\{\mathbf{x}\in\mathbb{R}^k_{\oplus} \;:\; A\mathbf{x}\geq\mathbf{b}\} \quad\text{és}\quad \mathcal{D}=\{\mathbf{y}\in\mathbb{R}^m_{\oplus} \;:\; A^T\mathbf{y}\leq\mathbf{c}\},$$

ahol $\mathbb{R}^k_{\oplus}$ a nemnegatív k-dimenziós vektorok halmazát, míg a későbbiekben előforduló $\mathbb{R}^k_{+}$ a pozitív k-dimenziós vektorok halmazát jelöli. Továbbá a ***primál,*** illetve a ***duál optimális megoldások halmazát*** a következőképpen jelöljük:

$$\begin{aligned} \mathcal{P}^* &= \{\mathbf{x}^*\in\mathcal{P} \;:\; \mathbf{c}^T\mathbf{x}^*\leq\mathbf{c}^T\mathbf{x},\;\; \forall\mathbf{x}\in\mathcal{P}\} \\ \text{és}\quad \mathcal{D}^* &= \{\mathbf{y}^*\in\mathcal{D} \;:\; \mathbf{b}^T\mathbf{y}^*\geq\mathbf{b}^T\mathbf{y},\;\; \forall\mathbf{y}\in\mathcal{D}\}. \end{aligned}$$

A gyenge dualitás tétel egyszerűen bizonyítható a kanonikus lineáris programozási feladatra.

25.1. tétel (gyenge dualitás tétel). *Tegyük fel, hogy* $\mathbf{x} \in \mathbb{R}^k$ *és* $\mathbf{y} \in \mathbb{R}^m$ *a primál* (P) *és duál* (D) *feladatok megengedett megoldásai. Ekkor*

$$\mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y} \,,$$

ahol egyenlőség akkor és csak akkor áll fenn, ha
(i) $x_i(\mathbf{c} - A^T\mathbf{y})_i = 0$ *minden* $i = 1, \ldots, k$ *és*
(ii) $y_j(A\mathbf{x} - \mathbf{b})_j = 0$ *minden* $j = 1, \ldots, m$.

Bizonyítás. Az $\mathbf{x}$ és $\mathbf{y}$ vektorok primál és duál megengedettségét használva kapjuk, hogy

$$(\mathbf{c} - A^T\mathbf{y})^T\mathbf{x} \geq 0 \qquad \text{és} \qquad \mathbf{y}^T(A\mathbf{x} - \mathbf{b}) \geq 0 \,,$$

ahol egyenlőség akkor és csak akkor áll fenn, ha (i) és (ii) teljesül (lásd 25.1-1. gyakorlat). Ezen két egyenlőtlenséget összeadva a kívánt

$$0 \leq (\mathbf{c} - A^T\mathbf{y})^T\mathbf{x} + \mathbf{y}^T(A\mathbf{x} - \mathbf{b}) = \mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y}$$

egyenlőtlenséget kapjuk. Tételünket bebizonyítottuk. ■

Az (i) és (ii) feltételeket ***komplementaritási feltételeknek*** nevezzük. Tehát a gyenge dualitás tétel szerint, komplementaritás és megengedettség a megoldások optimalitását garantálja.

Vektorok ***koordinátánkénti (Hadamard) szorzatát***[2] bevezetve, ahol $\mathbf{uv}$ szorzat $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ esetén azt az n-dimenziós vektort jelöli, melynek koordinátái az u_iv_i, $i = 1, \ldots, n$ értékek, a komplementaritási feltétel

$$\mathbf{x}(\mathbf{c} - A^T\mathbf{y}) = \mathbf{0} \quad \text{és} \quad \mathbf{y}(A\mathbf{x} - \mathbf{b}) = \mathbf{0}$$

alakban is írható. A továbbiakban a $\mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y}$ különbség értékét $\mathbf{x} \in \mathcal{P}$, $\mathbf{y} \in \mathcal{D}$ esetén ***dualitásrésnek*** nevezzük.

Könnyen bizonyítható az alábbi elégséges optimalitási feltétel (lásd 25.1-2. gyakorlat).

25.2. következmény (gyenge egyensúlyi tétel). *Legyenek* $\mathbf{x} \in \mathbb{R}^k$ *és* $\mathbf{y} \in \mathbb{R}^m$ *olyan primál és duál megengedett megoldások, amelyekre a dualitásrés nulla, azaz* $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$. *Ekkor* $\mathbf{x}$ *a* (P) *primál feladat egy optimális megoldása és* $\mathbf{y}$ *a* (D) *duál feladat egy optimális megoldása.*

Goldman–Tucker modell

Célunk a lineáris programozási feladat olyan megoldásainak előállítása, melyekre a dualitásrés nulla, így azt az egyenlőtlenségrendszert kell megoldanunk, amely a primál és a duál feltételekből áll, valamint előírjuk, hogy a duál célfüggvény értéke legalább akkora legyen, mint a primál célfüggvény értéke ($\mathbf{b}^T\mathbf{y} \geq \mathbf{c}^T\mathbf{x}$). Ekkor ugyanis a gyenge dualitás tétel miatt ezen rendszer minden megengedett megoldása primál és duál megengedett megoldást ad, amelyre a célfüggvényértékek szükségképpen egyenlők, így a 25.2. következmény szerint optimális megoldások.

[2] Hasonlóan, koordinátánkénti műveletként definiáljuk a vektorok hányadosát és tetszőleges hatványát is.

A szükséges eltérésváltozók $(\mathbf{t}, \mathbf{s}, \zeta)$ bevezetésével az alábbi egyenlőtlenségrendszerre jutunk:

$$\begin{aligned} A\mathbf{x} - \mathbf{t} &= \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}, \quad \mathbf{t} \geq \mathbf{0}, \\ A^T\mathbf{y} + \mathbf{s} &= \mathbf{c}, \quad \mathbf{y} \geq \mathbf{0}, \quad \mathbf{s} \geq \mathbf{0}, \\ \mathbf{b}^T\mathbf{y} - \mathbf{c}^T\mathbf{x} - \zeta &= 0, \quad \zeta \geq 0. \end{aligned}$$

Homogenizálva az egyenleteket a ***Goldman–Tucker feladatot*** kapjuk:

$$\left.\begin{array}{rrrrrcll} A\mathbf{x} & -\xi\mathbf{b} & -\mathbf{t} & & & = & \mathbf{0}, & \mathbf{x} \geq \mathbf{0}, \quad \mathbf{t} \geq \mathbf{0}, \\ -A^T\mathbf{y} & +\xi\mathbf{c} & & -\mathbf{s} & & = & \mathbf{0}, & \mathbf{y} \geq \mathbf{0}, \quad \mathbf{s} \geq \mathbf{0}, \\ \mathbf{b}^T\mathbf{y} & -\mathbf{c}^T\mathbf{x} & & & -\zeta & = & 0, & \xi \geq 0, \quad \zeta \geq 0. \end{array}\right\} \quad (GT)$$

A triviális, azonosan nulla megoldás kielégíti ezt a homogén rendszert, de ez a céljaink szempontjából érdektelen. A Goldman–Tucker rendszer bizonyos nemtriviális, speciális megoldását szeretnénk előállítani, és ennek segítségével nyerjük a (P) és (D) feladatok optimális megoldásait, illetve következtetünk a (P) és (D) feladatok megoldhatóságára az alábbi tétel alapján.

25.3. tétel. *Adott egy primál (P) és duál (D) lineáris programozási feladatpár. Az alábbi állítások igazak:*

1. *A (P) és (D) feladatok tetszőleges $(\mathbf{x}, \mathbf{y})$ optimális megoldás párja, melyre a dualitásrés nulla, a megfelelő Goldman–Tucker rendszer egy megoldását adja $\xi = 1$ és $\zeta = 0$ választással.*

2. *Ha $(\mathbf{y}, \mathbf{x}, \xi, \mathbf{t}, \mathbf{s}, \zeta)$ a Goldman–Tucker rendszer egy megoldása, akkor vagy $\xi = 0$ vagy $\zeta = 0$, azaz $\xi\zeta > 0$ nem teljesülhet.*

3. *Ha a Goldman–Tucker rendszer egy $(\mathbf{y}, \mathbf{x}, \xi, \mathbf{t}, \mathbf{s}, \zeta)$ megoldására $\xi > 0$ és $\zeta = 0$, akkor a $(\frac{\mathbf{x}}{\xi}, \frac{\mathbf{y}}{\xi})$ vektor a primál (P) és a duál (D) feladatok egy optimális megoldás párja.*

4. *Ha a Goldman–Tucker rendszernek van olyan $(\bar{\mathbf{y}}, \bar{\mathbf{x}}, \bar{\xi}, \bar{\mathbf{t}}, \bar{\mathbf{s}}, \bar{\zeta})$ megoldása, amelyre $\bar{\xi} = 0$ és $\bar{\zeta} > 0$, akkor megállapíthatjuk, hogy vagy a (P), vagy a (D) feladat, vagy mindkettő nem megengedett.*

Bizonyítás. Az első és a harmadik állítás behelyettesítéssel könnyen ellenőrizhető (lásd 25.1-3. gyakorlat).

A második állítást indirekt módon bizonyítjuk. Ha $\xi\zeta$ pozitív lenne, akkor

$$0 < \xi\zeta = \xi\mathbf{b}^T\mathbf{y} - \xi\mathbf{c}^T\mathbf{x} = \mathbf{x}^TA\mathbf{y} - \mathbf{t}^T\mathbf{y} - \mathbf{x}^TA^T\mathbf{y} - \mathbf{s}^T\mathbf{x} = -\mathbf{t}^T\mathbf{y} - \mathbf{s}^T\mathbf{x} \leq 0$$

egyenlőtlenséget kapnánk, ami nyilvánvaló ellentmondás.

Az utolsó állítás igazolásánál a $\bar{\xi} = 0$ feltételből következik, hogy $A\bar{\mathbf{x}} \geq \mathbf{0}$ és $A^T\bar{\mathbf{y}} \leq \mathbf{0}$. Továbbá, ha $\bar{\zeta} > 0$, akkor vagy $\mathbf{b}^T\bar{\mathbf{y}} > 0$, vagy $\mathbf{c}^T\bar{\mathbf{x}} < 0$, vagy mindkettő fennáll. Ha $\mathbf{b}^T\bar{\mathbf{y}} > 0$, akkor feltételezve, hogy a (P) feladatnak van egy $\mathbf{x} \geq \mathbf{0}$ megoldása a

$$0 < \mathbf{b}^T\bar{\mathbf{y}} \leq \mathbf{x}^TA^T\bar{\mathbf{y}} \leq 0$$

ellentmondáshoz jutunk. Tehát ha $\mathbf{b}^T\bar{\mathbf{y}} > 0$, akkor (P) nem megengedett.
Hasonlóan, ha $\mathbf{c}^T\bar{\mathbf{x}} < 0$, akkor a duál feladat nem megengedettségét kapjuk (lásd 25.1-4. gyakorlat). ■

Vegyük észre, hogy a Goldman–Tucker rendszer az alábbi kompakt alakban írható:

$$M\mathbf{u} \geq \mathbf{0}, \qquad \mathbf{u} \geq \mathbf{0}, \qquad \mathbf{z} = M\,\mathbf{u}, \tag{25.1}$$

ahol

$$\mathbf{u} = \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \\ \xi \end{pmatrix}, \qquad \mathbf{z} = \begin{pmatrix} \mathbf{t} \\ \mathbf{s} \\ \zeta \end{pmatrix} \quad \text{és} \quad M = \begin{pmatrix} 0 & A & -\mathbf{b} \\ -A^T & 0 & \mathbf{c} \\ \mathbf{b}^T & -\mathbf{c}^T & 0 \end{pmatrix}$$

egy ***ferdén szimmetrikus mátrix*** ($M^T = -M$, lásd 25.1-5. gyakorlat), azaz tetszőleges (P) és (D) feladatpárhoz tartozó Goldman–Tucker rendszer felírható, mint egy speciális struktúrájú, kanonikus lineáris programozási feladat, azaz

$$(\overline{SP}) \qquad \min\left\{\mathbf{0}^T\mathbf{u} \;:\; M\mathbf{u} \geq \mathbf{0},\;\; \mathbf{u} \geq \mathbf{0}\right\}.$$

25.1.1. A ferdén szimmetrikus önduális feladat alaptulajdonságai

Az $(\overline{SP})$ feladatnál tekintsünk egy kicsit általánosabb feladatot. Ez az $(\overline{SP})$ feladattól abban különbözik, hogy nem homogén, azaz a jobb oldala nem feltétlenül a nulla vektor, és célfüggvény együttható vektora a jobb oldali vektor negatívja. A következő (speciális) lineáris programozási feladattal ***(önduális feladat)*** foglalkozunk a továbbiakban

$$\left.\begin{array}{rcl} \min & \mathbf{q}^T\mathbf{u} & \\ M\mathbf{u} & \geq & -\mathbf{q} \\ \mathbf{u} & \geq & \mathbf{0} \end{array}\right\} \quad (SP)\,,$$

ahol $M \in \mathbb{R}^{n\times n}$ ferdén szimmetrikus mátrix és $\mathbf{q} \in \mathbb{R}^n_{\oplus}$ vektor. Jelölje

$$\mathcal{F} = \{\mathbf{u} \in \mathbb{R}^n_{\oplus} \;:\; M\mathbf{u} \geq -\mathbf{q}\}$$

az (SP) feladat ***megengedett megoldásainak halmazát***. Felhasználva, hogy az M mátrix ferdén szimmetrikus és hogy a jobb oldalon álló vektor $(-\mathbf{q})$ a célfüggvényvektor negatívja, az Olvasó könnyen ellenőrizheti, hogy (SP) duálja ekvivalens az (SP) feladattal, azaz az (SP) ***önduális*** feladat (lásd 25.1-6. feladat). Az önduális tulajdonság miatt a következő eredmény triviális (lásd 25.1-7. feladat).

25.4. lemma. *Az (SP) feladat optimum értéke nulla, továbbá az azonosan nulla vektor, $\mathbf{u} = \mathbf{0}$, megengedett és egyben optimális megoldása az (SP) feladatnak.*

Ha $\mathbf{u}$ megoldása az (SP) feladatnak, és $z(\mathbf{u}) = M\mathbf{u} + \mathbf{q}$ az $\mathbf{u}$ vektorhoz tartozó ***eltérésvektor***, akkor M ferdén szimmetrikussága miatt $\mathbf{u}^T M\mathbf{u} = 0$, így

$$\mathbf{q}^T\mathbf{u} = \mathbf{u}^T(z(\mathbf{u}) - M\mathbf{u}) = \mathbf{u}^T z(\mathbf{u}) = \mathbf{e}^T(\mathbf{u}\,z(\mathbf{u}))\,,$$

ahol $\mathbf{e} \in \mathbb{R}^n$ a csupa egyes vektor. A fenti egyenlőség miatt tetszőleges optimális megoldásra $\mathbf{e}^T(\mathbf{u}\,z(\mathbf{u})) = 0$, azaz $\mathbf{u}\,z(\mathbf{u}) = 0$, amiből az is következik, hogy az $\mathbf{u}$ és $z(\mathbf{u})$ vektorok komplementárisak. Jelölje

$$\begin{aligned} \mathcal{F}^* &= \{\mathbf{u}^* \in \mathcal{F} \;:\; \mathbf{q}^T\mathbf{u}^* \leq \mathbf{q}^T\mathbf{u},\ \forall \mathbf{u} \in \mathcal{F}\} \\ &= \{\mathbf{u}^* \in \mathcal{F} \;:\; \mathbf{q}^T\mathbf{u}^* = 0\} = \{\mathbf{u}^* \in \mathcal{F} \;:\; \mathbf{u}^* z(\mathbf{u}^*) = \mathbf{0}\} \end{aligned}$$

az ***(SP) feladat optimális megoldásainak a halmazát.*** Adott $\mathbf{u} \in \mathcal{F}$ esetén az $\mathbf{u}^T z(\mathbf{u}) = \mathbf{q}^T \mathbf{u}$ értéket ***dualitásrésnek*** nevezzük.[3] Az $\mathcal{F}^*$ halmaz definíciója alapján világos, hogy optimális megoldás esetén a dualitásrés nulla.

Az optimális megoldások egy, a továbbiakban gyakran használt tulajdonságát fogalmazzuk meg az alábbi lemmában.

25.5. lemma. *Legyen* $\mathbf{u}$ *és* $\hat{\mathbf{u}}$ *az (SP) feladat megengedett megoldása. Az* $\mathbf{u}$ *és* $\hat{\mathbf{u}}$ *vektorok akkor és csak akkor optimálisak, ha*

$$\mathbf{u}\, z(\hat{\mathbf{u}}) = \hat{\mathbf{u}}\, z(\mathbf{u}) = \mathbf{u}\, z(\mathbf{u}) = \hat{\mathbf{u}}\, z(\hat{\mathbf{u}}) = \mathbf{0}\ .$$

Bizonyítás. Mivel M ferdén szimmetrikus, így $(\mathbf{u}-\hat{\mathbf{u}})^T M(\mathbf{u}-\hat{\mathbf{u}}) = 0$, amiből következik, hogy $(\mathbf{u}-\hat{\mathbf{u}})^T(z(\mathbf{u}) - z(\hat{\mathbf{u}})) = 0$. Ekkor $\mathbf{u}^T z(\hat{\mathbf{u}}) + \hat{\mathbf{u}}^T z(\mathbf{u}) = \mathbf{u}^T z(\mathbf{u}) + \hat{\mathbf{u}}^T z(\hat{\mathbf{u}})$ és ez akkor és csak akkor nulla, ha $\mathbf{u}$ és $\hat{\mathbf{u}}$ is optimális, de ekkor

$$\mathbf{u}^T z(\hat{\mathbf{u}}) + \hat{\mathbf{u}}^T z(\mathbf{u}) = 0\ . \tag{25.2}$$

Figyelembe véve az $\mathbf{u}, \hat{\mathbf{u}} \in \mathcal{F}$ feltételt $\mathbf{u}^T z(\hat{\mathbf{u}}) \geq 0$ és $\hat{\mathbf{u}}^T z(\mathbf{u}) \geq 0$ teljesül, amelyből, az (25.2) összefüggés alapján

$$\mathbf{u}^T z(\hat{\mathbf{u}}) = \mathbf{e}^T(\mathbf{u}\, z(\hat{\mathbf{u}})) = 0 \qquad \text{és} \qquad \hat{\mathbf{u}}^T z(\mathbf{u}) = \mathbf{e}^T(\hat{\mathbf{u}}\, z(\mathbf{u})) = 0$$

következik. Tehát $\mathbf{u}\, z(\hat{\mathbf{u}}) = \hat{\mathbf{u}}\, z(\mathbf{u}) = \mathbf{0}$ adódik. ■

Megállapíthatjuk, hogy az optimális megoldások általános értelemben is komplementárisak, azaz nem csak saját eltérésvektorukkal, hanem bármelyik más optimális megoldás eltérésvektorával is komplementáris párt alkotnak.

Az (*SP*) feladat ***optimalitási kritériumát*** a következő alakban is megadhatjuk

$$\left.\begin{aligned} -M\mathbf{u} + \mathbf{z} &= \mathbf{q} \\ \mathbf{u} \geq \mathbf{0}, \mathbf{z} &\geq \mathbf{0} \\ \mathbf{u}\ \mathbf{z} &= \mathbf{0} \end{aligned}\right\} (SP_{OPT})\ .$$

Az összes eddigi eredmény, egy triviális optimális megoldás létezését is beleértve, szinte magától értetődő volt az (*SP*) feladatra. Ebből talán arra következtethetnénk, hogy az (*SP*) feladat a lineáris programozási feladatok érdektelen, nagyon speciális esete. Tekintettel arra, hogy az (*SP*) feladat homogén változatát a kanonikus primál- és duál feladatokból vezettük le, fel sem merül annak a lehetősége, hogy egy érdektelen feladattal foglalkoznánk.

Az első felvetődő kérdés az, hogy létezik-e a triviálison kívül másmilyen optimális megoldása a feladatnak, és hogyan tudjuk azt előállítani. Ennek a kérdésnek a megválaszolása elvezet az optimális megoldások bizonyos komplementaritási tulajdonságainak kérdéséhez, azaz a lineáris programozás elméletének egyik alapvető jelentőségű tételéhez, a Goldman–Tucker-tételhez.

Szükségünk lesz a következő definícióra.

25.6. definíció. *Legyen* $\mathbf{u} \in \mathcal{F}^*$. *Az* $\mathbf{u}$ *és* $z(\mathbf{u})$ *vektorokat* ***szigorúan komplementárisnak*** *nevezzük, ha* $\mathbf{u} + z(\mathbf{u}) > \mathbf{0}$ *feltétel teljesül.*

[3] Tesszük ezt annak ellenére, hogy az önduális feladat esetén az $\mathbf{u}^T z(\mathbf{u}) = \mathbf{q}^T \mathbf{u}$ nyilván csak a fele a klasszikus értelemben vett dualitásrésnek (lásd 1232. o.).

Az $\mathbf{u}$ és $z(\mathbf{u})$ vektorok szigorú komplementaritásának egyszerű következménye az, hogy minden $i = 1, 2, \ldots, n$ index esetén az $u_i = 0$ és $z_i(u) = 0$ feltételek közül pontosan az egyik teljesül. A 25.3. tétel 3. és 4. állításai bizonyítják, hogy amennyiben az (*SP*) feladat a (*GT*) rendszer alapján egy lineáris programozási feladatpárból származik, bármely szigorúan komplementáris megoldása vagy egy optimális megoldás párt ad az eredeti (*P*) és (*D*) lineáris programozási feladatokra, vagy azok valamelyikének nem-megoldhatóságát bizonyítja. Az (*SP*) feladat egy szigorúan komplementáris megoldásának az előállítása a lineáris programozás belsőpontos módszereinek a bevezetését igényeli.

Vezessük be a ***belsőpontos megoldások halmazát***

$$\mathcal{F}^0 = \{\mathbf{u} \in \mathcal{F} \ : \ (\mathbf{u}, z(\mathbf{u})) > \mathbf{0}\} \ .$$

Ekkor az ún. ***belső pont feltételt*** az alábbi módon fogalmazhatjuk meg:

$$\mathcal{F}^0 \neq \emptyset \ ,$$

amelyet másként úgy is kimondhatunk, hogy létezik olyan $\bar{\mathbf{u}} \in \mathcal{F}$ vektor, amelyre

$$(\bar{\mathbf{u}}, \ z(\bar{\mathbf{u}})) > \mathbf{0}$$

teljesül. (A 25.1-11. gyakorlat ad példát olyan lineáris programozási feladatra, amelyikből elkészített Goldman–Tucker feladatnak van belső pontja.)

Newton-lépés és tulajdonságai

Legyen adott $(\mathbf{u}, \mathbf{z}) > \mathbf{0}$, melyre $\mathbf{z} = M\mathbf{u}+\mathbf{q}$. Adott $\mathbf{w} \in \mathbb{R}^n, \ \mathbf{w} > \mathbf{0}$ vektor esetén szeretnénk[4] olyan $(\triangle\mathbf{u}, \triangle\mathbf{z})$ elmozdulás vektort („lépést") meghatározni, melyre

$$\begin{aligned} M(\mathbf{u} + \triangle\mathbf{u}) + \mathbf{q} &= \mathbf{z} + \triangle\mathbf{z} \ , \\ (\mathbf{u} + \triangle\mathbf{u})(\mathbf{z} + \triangle\mathbf{z}) &= \mathbf{w} \end{aligned}$$

egyenletrendszer teljesül. Ez nemlineáris egyenletrendszer, így direkt megoldása nem lehetséges. Átrendezve a $(\Delta\mathbf{u}, \Delta\mathbf{z})$ ismeretlenekre vonatkozóan a

$$\begin{aligned} M\Delta\mathbf{u} - \Delta\mathbf{z} &= \mathbf{0} \ , \\ \mathbf{u}\Delta\mathbf{z} + \mathbf{z}\Delta\mathbf{u} + \Delta\mathbf{u}\Delta\mathbf{z} &= \mathbf{w} - \mathbf{u}\mathbf{z} \end{aligned}$$

egyenletrendszert kapjuk, amely még mindig nemlineáris. A másodrendű $\Delta\mathbf{u}\Delta\mathbf{z}$ tag elhagyásával az

$$\begin{aligned} M\Delta\mathbf{u} - \Delta\mathbf{z} &= \mathbf{0} \ , \\ Z\Delta\mathbf{u} + U\Delta\mathbf{z} &= \mathbf{w} - \mathbf{u}\mathbf{z} \end{aligned} \tag{25.3}$$

lineáris egyenletrendszer adódik, ahol $U = \mathrm{diag}(\mathbf{u})$ és $Z = \mathrm{diag}(\mathbf{z})$ pozitív diagonális mátrixok. A (25.3) egyenletrendszert ***Newton-rendszernek***[5] nevezzük, amelynek az együtthatómátrixa

$$\bar{M} = \begin{pmatrix} M & -I \\ Z & U \end{pmatrix} \in \mathbb{R}^{2n \times 2n} \tag{25.4}$$

---

[4] Célunk a

$$\begin{aligned} -M\bar{\mathbf{u}} + \bar{\mathbf{z}} &= \mathbf{q} \\ \bar{\mathbf{u}}\,\bar{\mathbf{z}} &= \mathbf{0} \end{aligned}$$

rendszert megoldani, ahol a megoldást $\bar{\mathbf{u}} = \mathbf{u} + \Delta\mathbf{u}$;s $\bar{\mathbf{u}} = \mathbf{u} + \Delta\mathbf{u}$ alakban keressük.

[5] Ez a rendszer a numerikus analízisből és a többdimenziós nemlineáris egyenletrendszerek megoldásához is használt jól ismert Newton-módszer megfelelője.

alakú. A $\Delta\mathbf{z} = M\,\Delta\mathbf{u}$ behelyettesítése után a Newton rendszer

$$(Z + U\,M)\,\Delta\mathbf{u} \;=\; \mathbf{w} - \mathbf{u}\,\mathbf{z} \tag{25.5}$$

alakra egyszerűsödik, amelyet ***normál egyenletrendszernek*** nevezünk.

25.7. állítás. *Legyenek $U, Z, I, M \in \mathbb{R}^{n\times n}$ mátrixok. U és Z pozitív diagonális, I egység, míg M ferdén szimmetrikus mátrix. Ekkor a (25.4) összefüggéssel definiált $\bar{M}$ és a $Z + UM$ mátrix reguláris, tehát a (25.5) lineáris egyenletrendszernek létezik egyértelmű megoldása.*

A bizonyítás lineáris algebrai gondolatmenetet használ, és az Olvasóra bízzuk (lásd 25.1-12., illetve 25.1-13. gyakorlat).

A $\Delta\mathbf{u}$ és $\Delta\mathbf{z}$ elmozdulás vektorokat ***Newton-irányoknak*** nevezzük, és az alábbi formában[6] adhatók meg:

$$\begin{aligned} \Delta\mathbf{u} &= (Z + U\,M)^{-1}\,(\mathbf{w} - \mathbf{u}\,\mathbf{z}) = (U^{-1}Z + M)^{-1}(U^{-1}\mathbf{w} - \mathbf{z}) \quad \text{és} \\ \Delta\mathbf{z} &= M(U^{-1}Z + M)^{-1}(U^{-1}\mathbf{w} - \mathbf{z}) = M\Delta\mathbf{u}\;. \end{aligned} \tag{25.6}$$

A lépéshossz követelménye, hogy az új pontunkra is teljesüljön az előjel feltétel. Miután a Newton-irányban egy α lépéshosszú lépést teszünk, az új $(\mathbf{u}(\alpha),\ \mathbf{z}(\alpha)) = (\mathbf{u}+\alpha\Delta\mathbf{u},\ \mathbf{z}+\alpha\Delta\mathbf{z})$ megoldásra a következő kifejezést kapjuk:

$$\begin{aligned} \mathbf{u}(\alpha)\,\mathbf{z}(\alpha) &= (\mathbf{u} + \alpha\Delta\mathbf{u})(\mathbf{z} + \alpha\,\Delta\mathbf{z}) = \mathbf{u}\,\mathbf{z} + \alpha\,(\mathbf{u}\,\Delta\mathbf{z} + \mathbf{z}\,\Delta\mathbf{u}) + \alpha^2\,\Delta\mathbf{u}\,\Delta\mathbf{z} \\ &= \mathbf{u}\,\mathbf{z} + \alpha\,(\mathbf{w} - \mathbf{u}\,\mathbf{z}) + \alpha^2\,\Delta\mathbf{u}\,\Delta\mathbf{z}\;. \end{aligned}$$

Ez az összefüggés világossá teszi, hogy az $\mathbf{u}\,\mathbf{z}$ vektor lokális megváltozását a $\mathbf{w} - \mathbf{u}\,\mathbf{z}$ vektor határozza meg. Szerencsére ezt a vektort explicit ismerjük, amikor a Newton-lépést alkalmazzuk. Így ha α elegendően kicsi, akkor pontosan tudjuk, hogy $\mathbf{u}\,\mathbf{z}$ mely koordinátái csökkennek lokálisan (pontosan azok, amelyekre a $\mathbf{w} - \mathbf{u}\,\mathbf{z}$ vektor megfelelő koordinátái negatívak), és $\mathbf{u}\,\mathbf{z}$ mely koordinátái növekednek lokálisan (pontosan azok, amelyekre a $\mathbf{w} - \mathbf{u}\,\mathbf{z}$ vektor megfelelő koordinátái pozitívak).

A 25.7. állítást illusztrálja a 25.1-11. gyakorlat. Ebben a feladatban a Newton-irányok kiszámítására, az α lépéshossz előállítására és a megengedett pozitív $\mathbf{u}^+$, illetve $\mathbf{z}^+$ meghatározására láthatunk példát.

Ezeket az észrevételeket fejezi ki pontosabban a következő állítás is.

25.8. állítás. *Legyen az M ferdén szimmetrikus mátrix, és $\mathbf{u} \in \mathcal{F}^0$, azaz $(\mathbf{u},\ z(\mathbf{u})) > \mathbf{0}$. Legyen továbbá, $\hat{\mathbf{w}} \in \mathbb{R}^n$, $\hat{\mathbf{w}} > \mathbf{0}$ és $\mathbf{w} = \mathbf{u}\,z(\mathbf{u})$. Definiáljuk a*

$$\mathcal{T}(\hat{\mathbf{w}},\ \mathbf{w}) = \{\,\mathbf{u} \in \mathbb{R}^n \;:\; \hat{w}_i \le u_i \le w_i \;\; \textit{vagy} \;\; w_i \le u_i \le \hat{w}_i,\ \forall\, i\,\}$$

tégla *halmazt. Ha $\operatorname{int}\mathcal{T}(\hat{\mathbf{w}},\ \mathbf{w}) \neq \emptyset$, akkor a $\hat{\mathbf{w}}$ vektorra kiszámított $(\Delta\mathbf{u},\ \Delta\mathbf{z})$ Newton-irányra létezik olyan $\alpha_* \in (0, 1)$ lépéshossz, hogy az*

$$\mathbf{u}^+ = \mathbf{u} + \alpha_*\Delta\mathbf{u}\;, \quad \mathbf{z}^+ = \mathbf{z} + \alpha_*\,\Delta\mathbf{z}\;, \quad \mathbf{w}^+ = \mathbf{u}^+\mathbf{z}^+\;,$$

vektorokra a $\mathbf{w}^+ \in \operatorname{int}\mathcal{T}(\hat{\mathbf{w}},\ \mathbf{w})$, valamint $\mathbf{u}^+ \in \mathcal{F}^0$ teljesül.

Az előző állítás bizonyításának a legfontosabb részeit két feladatra bontjuk (lásd 25.1-14. gyakorlat), amelyek megoldását az Olvasóra bízzuk.

---

[6] A Newton-irányok meghatározásánál a numerikus szempontokból hatékonyabb, ha a (25.5) lineáris egyenletrendszer egyértelmű megoldásaként határozzuk meg a $\Delta\mathbf{u}$ vektort.

Az (*SP*) feladat szinthalmazai és tulajdonságaik
Először vezessük be a

$$\mathcal{L}_K = \{\mathbf{u} \in \mathcal{F} \ : \ \mathbf{u}^T z(\mathbf{u}) \leq K\} = \{\mathbf{u} \in \mathcal{F} \ : \ \mathbf{q}^T \mathbf{u} \leq K\} \ ,$$

szinthalmazt és bármely $\mathbf{w} \in \mathbb{R}^n_+$ vektor esetén az

$$\mathcal{L}_{\mathbf{w}} = \{(\mathbf{u}, \mathbf{z}) \in \mathbb{R}^{2n}_{\oplus} \ : \ \mathbf{z} = M\mathbf{u} + \mathbf{q} \text{ és } \mathbf{u}\,\mathbf{z} \leq \mathbf{w}\} \ .$$

általánosított szinthalmazt.

25.9. lemma. *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Ekkor bármely $K \in \mathbb{R}$, $K > 0$ szám esetén az $\mathcal{L}_K$ szinthalmaz korlátos és zárt, azaz kompakt.*

Az előző lemma bizonyítását (lásd 25.1-16. gyakorlat) az Olvasóra bízzuk.

A 25.9. lemmához hasonló eredmények igazak az $\mathcal{L}_{\mathbf{w}}$ szinthalmazokra is. A szinthalmaz definíciójában szereplő nemlineáris kifejezés miatt kicsit bonyolultabb a zártság bizonyítása (lásd 25.1-17. gyakorlat).

25.10. lemma. *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Ekkor bármely $\mathbf{w} \in \mathbb{R}^n_+$, vektor esetén az $\mathcal{L}_{\mathbf{w}}$ szinthalmaz korlátos és zárt, azaz kompakt.*

A szinthalmazokkal kapcsolatos lemmákra a következő állítás bizonyításakor lesz szükségünk. Ebben a lemmában azt vizsgáljuk, hogy milyen halmazt alkotnak azok a $\mathbf{w} \in \mathbb{R}^n_{\oplus}$ vektorok, amelyek előállnak az (*SP*) feladat valamely megengedett megoldásának és a hozzátartozó eltérésváltozónak a szorzataként.

25.11. lemma. *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Definiáljuk a következő halmazt*

$$\mathcal{G} = \{\mathbf{w} \in \mathbb{R}^n_{\oplus} \ : \ \exists \mathbf{u} \in \mathcal{F}, \quad \textit{amelyre} \quad \mathbf{u}\, z(\mathbf{u}) = \mathbf{w}\} \ .$$

Ekkor a $\mathcal{G}$ halmaz nem üres és zárt.

A lemma bizonyítását (lásd 25.1-18. gyakorlat) az Olvasóra bízzuk. Az előző eredmény segítségével belátjuk, hogy belső pont létezése mellett minden $\mathbf{w} > 0$ esetén az $\mathcal{L}_{\mathbf{w}}$ szinthalmaz nem üres.

25.12. tétel. *Legyen adott az (SP) feladat, és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Ekkor bármely $\mathbf{w} \in \mathbb{R}^n_+$ vektor esetén az $\mathcal{L}_{\mathbf{w}}$ szinthalmaz nem üres.*

Bizonyítás. Az 25.10 lemmában már igazoltuk, hogy az $\mathcal{L}_{\mathbf{w}}$ halmaz kompakt. Indirekt módon tegyük fel, hogy létezik egy $\hat{\mathbf{w}} \in \mathbb{R}^n_+$ vektor, amelyre $\mathcal{L}_{\hat{\mathbf{w}}} = \emptyset$.

Az $\mathcal{F}^0 \neq \emptyset$ feltevés miatt létezik egy $\bar{\mathbf{u}} \in \mathcal{F}^0$ vektor és egy $\bar{\mathbf{w}} = \bar{\mathbf{u}}\,\bar{\mathbf{z}} > \mathbf{0}$, ahol $\bar{\mathbf{z}} = M\bar{\mathbf{u}} + \mathbf{q} > \mathbf{0}$ vektor, amelyek esetén az $\mathcal{L}_{\bar{\mathbf{w}}}$ szinthalmaz nem üres és kompakt.

Legyen $\mathbf{w}' > \bar{\mathbf{w}}$ és $\mathbf{w}' > \hat{\mathbf{w}}$. Ekkor az $\mathcal{A} = \{\mathbf{w} \in \mathbb{R}^n_{\oplus} \ : \ \mathbf{e}^T\mathbf{w} \leq \mathbf{e}^T\mathbf{w}'\}$ nem üres és kompakt halmaz. Továbbá az $\mathcal{A} \cap \mathcal{G}$ halmaz sem üres és kompakt.

Definiáljuk az $f : \mathcal{A} \cap \mathcal{G} \to \mathbb{R}^n_{\oplus}$ függvényt a következő kifejezéssel

$$f_i(\mathbf{w}) = \begin{cases} 0, & \text{ha } \ w_i \leq \hat{w}_i \ , \\ w_i - \hat{w}_i & \text{különben} \ . \end{cases}$$

A 25.1-19. gyakorlat szerint $\|f\|_\infty$ felveszi a minimumát, azaz létezik

$$\gamma = \|f(\tilde{\mathbf{w}})\|_\infty = \min_{\mathbf{w}\in\mathcal{A}\cap\mathcal{G}} \|f(\mathbf{w})\|_\infty \le \|f(\bar{\mathbf{w}})\|_\infty \ .$$

Mivel az $\mathcal{L}_{\tilde{\mathbf{w}}} = \emptyset$ (indirekt feltevés), ezért $\gamma = \|f(\tilde{\mathbf{w}})\|_\infty > 0$.

Ha $\operatorname{int}\mathcal{T}(\tilde{\mathbf{w}}, \hat{\mathbf{w}}) \neq \emptyset$, akkor létezik $\alpha \in (0, 1)$, amelyre $\mathbf{u}^+ = \bar{\mathbf{u}} + \alpha\,\Delta\mathbf{u}$ és $\mathbf{z}^+ = \bar{\mathbf{z}} + \alpha\,\Delta\mathbf{z}$ olyan vektorok, amelyekre $\mathbf{w}^+ = \mathbf{u}^+\,\mathbf{z}^+ \in \operatorname{int}\mathcal{T}(\tilde{\mathbf{w}}, \hat{\mathbf{w}})$, azaz

$$\|f(\mathbf{w}^+)\|_\infty < \|f(\tilde{\mathbf{w}})\|_\infty = \gamma \ ,$$

ami ellentmond Weierstrass tételének.

Ha $\operatorname{int}\mathcal{T}(\tilde{\mathbf{w}}, \hat{\mathbf{w}}) = \emptyset$, akkor legyen $\mathbf{w}^* \in \mathcal{B}_{\gamma/3}(\hat{\mathbf{w}}) \cap \operatorname{int}\mathcal{T}(\mathbf{0}, \hat{\mathbf{w}})$ vektor[7], amely esetén $\operatorname{int}\mathcal{T}(\tilde{\mathbf{w}}, \mathbf{w}^*) \neq \emptyset$ és megismételhetjük az előző gondolatmenetet. ■

25.1.2. Centrális út

Az (*SP*) feladat optimalitási kritériumának relaxáltja a

$$\left.\begin{aligned} -M\mathbf{u} + \mathbf{z} &= \mathbf{q} \\ \mathbf{u} \ge \mathbf{0},\ \mathbf{z} &\ge \mathbf{0} \\ \mathbf{u}\,\mathbf{z} &= \mu\mathbf{e} \end{aligned}\right\} \quad (CP) \ ,$$

az úgynevezett ***centrális út feladat.*** Belátjuk, hogy adott $\mu > 0$ paraméter esetén a (*CP*) feladatnak egyértelmű megoldása van. Ezt a megoldást ***μ-centrumnak*** nevezzük, amelyet az $(\mathbf{u}(\mu), \mathbf{z}(\mu))$ vektorral jelölünk.

25.13. definíció. *A*

$$\mathcal{C} = \{(\mathbf{u}(\mu), z(\mathbf{u}(\mu))) : \mathbf{u}(\mu) \in \mathcal{F}^0,\ \mathbf{u}(\mu)\mathbf{z}(\mu) = \mu\mathbf{e}, \quad \textit{valamely } \mu \in \mathbb{R}_+ \textit{ paraméterre}\}$$

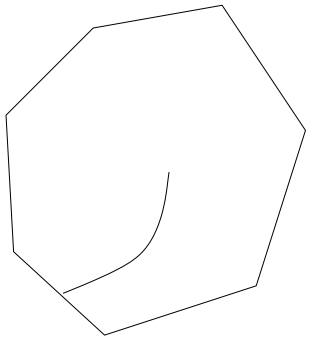
halmazt az (SP) feladat ***centrális útjának*** *nevezzük.*

A centrális út tehát a μ-centrumok által alkotott görbe (lásd 25.1. ábra). Bizonyítható, hogy belső pont létezése esetén a centrális út létezik és egyértelmű, sőt a centrális út egy környezete is egyértelmű. Ennél többet bizonyítunk be a következő tételben, mégpedig a fenti állítások ekvivalenciáját.

25.14. tétel. *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F} \neq \emptyset$. Ekkor a következő állítások ekvivalensek:*

1. $\mathcal{F}^0 \neq \emptyset$;
2. $\forall \mathbf{w} \in \mathbb{R}^n_+$ *esetén* $\exists!\ (\mathbf{u}, \mathbf{z}) > 0\ :\ M\mathbf{u} + \mathbf{q} = \mathbf{z}$ *és* $\mathbf{u}\,\mathbf{z} = \mathbf{w}$;
3. $\forall \mu > 0$ *esetén* $\exists!\ (\mathbf{u}, \mathbf{z}) > 0\ :\ M\mathbf{u} + \mathbf{q} = \mathbf{z}$ *és* $\mathbf{u}\,\mathbf{z} = \mu\mathbf{e}$.

[7] $\mathcal{B}_\gamma(\mathbf{w}) = \{\mathbf{u} \in \mathbb{R}^n\ :\ \|\mathbf{u} - \mathbf{w}\| < \gamma\}$, azaz a $\mathbf{w}$ középpontú, γ sugarú gömböt jelöli.



25.1. ábra. Centrális út a primál változók terében.

Bizonyítás. A 3. állítás a 2. speciális esete, továbbá a 3. állításból következik az 1., ugyanis $\mathbf{u}(\mu)\mathbf{z}(\mu) = \mu\mathbf{e}$ esetén $\mathbf{u}(\mu) \in \mathcal{F}^0$.
Az 1. $\Rightarrow$ 2. implikációt indirekt bizonyítjuk. Tegyük fel, hogy $\exists\, \hat{\mathbf{w}} \in \mathbb{R}^n_+$ úgy, hogy $\nexists \hat{\mathbf{u}} \in \mathcal{F}^0$ és $\hat{\mathbf{z}} = M\hat{\mathbf{u}} + \mathbf{q}$, amire teljesülne $\hat{\mathbf{w}} = \hat{\mathbf{u}}\,\hat{\mathbf{z}}$. Mivel fennáll a belső pont feltétel, a 25.12. tétel és a 25.10. lemma miatt $\mathcal{L}_{\hat{\mathbf{w}}} \neq \emptyset$ és kompakt.

Legyen $f : \mathcal{L}_{\hat{\mathbf{w}}} \to \mathbb{R}_+$ függvény, $f(\mathbf{u}, \mathbf{z}) = \mathbf{u}^T\mathbf{z}$. Az indirekt feltevésünk miatt $f(\mathbf{u}, \mathbf{z}) = \mathbf{u}^T\mathbf{z} < \mathbf{e}^T\hat{\mathbf{w}}$ minden $(\mathbf{u}, \mathbf{z}) \in \mathcal{L}_{\hat{\mathbf{w}}}$. Az f folytonos függvény, így Weierstrass tétele miatt felveszi a maximumát az $\mathcal{L}_{\hat{\mathbf{w}}}$ kompakt halmazon, azaz

$$\mathbf{e}^T\overline{\mathbf{w}} = \overline{\mathbf{u}}^T\overline{\mathbf{z}} = f(\overline{\mathbf{u}}, \overline{\mathbf{z}}) = \max_{(\mathbf{u},\mathbf{z})\in\mathcal{L}_{\hat{\mathbf{w}}}} f(\mathbf{u}, \mathbf{z}) = \max_{(\mathbf{u},\mathbf{z})\in\mathcal{L}_{\hat{\mathbf{w}}}} \mathbf{u}^T\mathbf{z} < \mathbf{e}^T\hat{\mathbf{w}}\,, \tag{25.7}$$

ahol $\overline{\mathbf{w}} = \overline{\mathbf{u}}\overline{\mathbf{z}}$ és $\overline{\mathbf{w}} \leq \hat{\mathbf{w}}$, de $\overline{\mathbf{w}} \neq \hat{\mathbf{w}}$ az indirekt feltevésünk miatt.
Két eset lehetséges:

1. Ha $\operatorname{int}\mathcal{T}(\overline{\mathbf{w}}, \hat{\mathbf{w}}) \neq \emptyset$, akkor $\exists\, \mathbf{w}^+ \in \mathcal{T}(\overline{\mathbf{w}}, \hat{\mathbf{w}}) : \overline{\mathbf{w}} < \mathbf{w}^+ < \hat{\mathbf{w}}$ és $\mathbf{w}^+ = \mathbf{u}^+\mathbf{z}^+$, ahol $\mathbf{u}^+ \in \mathcal{F}^0$. Így $\mathbf{e}^T\overline{\mathbf{w}} < \mathbf{e}^T\mathbf{w}^+$, és ez ellentmond a (25.7) egyenlőtlenségnek, hiszen $(\mathbf{u}^+, \mathbf{z}^+) \in \mathcal{L}_{\hat{\mathbf{w}}}$.

2. Ha $\operatorname{int}\mathcal{T}(\overline{\mathbf{w}}, \hat{\mathbf{w}}) = \emptyset$, akkor legyen $\delta = \mathbf{e}^T(\hat{\mathbf{w}} - \overline{\mathbf{w}}) > 0$, és legyen

$$\tilde{\mathbf{w}} \in \mathbb{R}^n_+ \;:\; \hat{\mathbf{w}} > \tilde{\mathbf{w}} > \hat{\mathbf{w}} - \frac{\delta}{n}\mathbf{e}\,.$$

Ekkor $\tilde{\mathbf{w}}$ választási szabálya miatt $\mathcal{T}(\overline{\mathbf{w}}, \tilde{\mathbf{w}}) \neq \emptyset$, azaz létezik $\mathbf{w}^+ \in \mathcal{T}(\overline{\mathbf{w}}, \tilde{\mathbf{w}})$ úgy, hogy $\mathbf{w}^+ = \mathbf{u}^+\mathbf{z}^+$, $\mathbf{u}^+ = \overline{\mathbf{u}} + \alpha\Delta\mathbf{u}$, $\mathbf{z}^+ = \overline{\mathbf{z}} + \alpha\Delta\mathbf{z}$, ahol $\mathbf{u}^+ \in \mathcal{F}^0$ és $\alpha \in (0, 1)$. A továbbiakban megmutatjuk, hogy $\mathbf{e}^T\mathbf{w}^+ > \mathbf{e}^T\overline{\mathbf{w}}$, ami ellentmondás (25.7) miatt. Felhasználva a $\Delta\mathbf{u}$, $\Delta\mathbf{z}$ vektorok ortogonalitását, illetve a $\overline{\mathbf{z}}\,\Delta\mathbf{u} + \overline{\mathbf{u}}\,\Delta\mathbf{z} = \tilde{\mathbf{w}} - \overline{\mathbf{u}}\,\overline{\mathbf{z}}$ egyenlőséget, a következőt kapjuk:

$$\begin{aligned} \mathbf{e}^T(\mathbf{w}^+ - \overline{\mathbf{w}}) &= \overline{\mathbf{u}}^T\overline{\mathbf{z}} + \alpha(\overline{\mathbf{z}}^T\Delta\mathbf{u} + \overline{\mathbf{u}}^T\Delta\mathbf{z}) + \alpha^2(\Delta\mathbf{u})^T\Delta\mathbf{z} - \overline{\mathbf{u}}^T\overline{\mathbf{z}} \\ &= \alpha(\overline{\mathbf{z}}^T\Delta\mathbf{u} + \overline{\mathbf{u}}^T\Delta\mathbf{z}) = \alpha\,\mathbf{e}^T(\tilde{\mathbf{w}} - \overline{\mathbf{w}})\,. \end{aligned} \tag{25.8}$$

$\tilde{\mathbf{w}}$ definíciója miatt

$$\tilde{\mathbf{w}} - \overline{\mathbf{w}} > (\hat{\mathbf{w}} - \overline{\mathbf{w}}) - \frac{\delta}{n}\mathbf{e}, \quad \text{így} \quad \mathbf{e}^T(\tilde{\mathbf{w}} - \overline{\mathbf{w}}) > \mathbf{e}^T(\hat{\mathbf{w}} - \overline{\mathbf{w}}) - \frac{\delta}{n}\mathbf{e}^T\mathbf{e} = \delta - \delta = 0\,.$$

A (25.8) egyenlőséget figyelembe véve $\mathbf{e}^T\mathbf{w}^+ > \mathbf{e}^T\overline{\mathbf{w}}$, amit bizonyítani szerettünk volna, azaz ellentmondásra jutottunk.
A fentiekben beláttuk, hogy $\forall \mathbf{w} \in \mathbb{R}^n_+$ esetén $\exists\,(\mathbf{u},\mathbf{z}) > 0:\ M\mathbf{u} + \mathbf{q} = \mathbf{z},\ \mathbf{w} = \mathbf{u}\,\mathbf{z}$.
Az egyértelműség bizonyítását az Olvasóra bízzuk (lásd 25.1-22. gyakorlat). ■

A következőkben megmutatjuk, hogy a centrális út limesz pontja az (*SP*) feladat egy optimális, sőt szigorúan komplementáris megoldása.

25.15. tétel. *Legyen adott az (SP) feladat, amelyre teljesül a belső pont feltétel. Ekkor létezik* $(\mathbf{u}^*,\mathbf{z}^*)$ *az alábbi tulajdonságokkal:*

1. $(\mathbf{u}^*,\mathbf{z}^*) = \lim\limits_{\mu\to 0}(\mathbf{u}(\mu),\mathbf{z}(\mu))$*;*
2. $\mathbf{u}^* \in \mathcal{F}^*$*;*
3. $(\mathbf{u}^*,\mathbf{z}^*)$ *szigorúan komplementáris megoldás.*

Bizonyítás. Az 1. és a 2. állítások bizonyítását (lásd 25.1-23. gyakorlat) az Olvasóra bízzuk.

Az utolsó állítás bizonyításához vezessük be a következő jelölést. Legyen $\mathbf{u} \in \mathbb{R}^n_\oplus$ és jelölje $\sigma(\mathbf{u}) = \{i\ :\ u_i > 0\}$ az $\mathbf{u}$ vektor tartóját. Felhasználva az M mátrix ferdén szimmetrikusságát

$$0 = (\mathbf{u}^* - \mathbf{u}(\mu))^T(\mathbf{z}^* - \mathbf{z}(\mu)) = (\mathbf{u}^*)^T\mathbf{z}^* + \mathbf{u}(\mu)^T\mathbf{z}(\mu) - (\mathbf{u}^*)^T\mathbf{z}(\mu) - (\mathbf{z}^*)^T\mathbf{u}(\mu)$$

egyenlet adódik. Figyelembe véve a második állítást és az $u(\mu)_i z(\mu)_i = \mu$ összefüggést

$$\begin{aligned}
\mu n &= (\mathbf{u}^*)^T\mathbf{z}(\mu) + (\mathbf{z}^*)^T\mathbf{u}(\mu) = \sum_{i\,:\,u_i^*>0} u_i^*(z(\mu))_i + \sum_{i\,:\,z_i^*>0} z_i^*(u(\mu))_i \\
n &= \sum_{i\,:\,u_i^*>0} u_i^*\frac{(z(\mu))_i}{\mu} + \sum_{i\,:\,z_i^*>0} z_i^*\frac{(u(\mu))_i}{\mu} = \sum_{i\,:\,u_i^*>0}\frac{u_i^*}{(u(\mu))_i} + \sum_{i\,:\,z_i^*>0}\frac{z_i^*}{(z(\mu))_i}\,.
\end{aligned}$$

Határátmenettel, amikor $\mu \to 0$, a

$$|\sigma(\mathbf{u}^*)| + |\sigma(\mathbf{z}^*)| = n$$

egyenletet kapjuk, ami csak úgy teljesülhet, ha az $(\mathbf{u}^*,\mathbf{z}^*)$ egy szigorúan komplementáris megoldás. ■

Az előző tétel harmadik állítása Goldman és Tucker eredménye, ami a következő formában ismert.

25.16. következmény (Goldman–Tucker-tétel belső pont feltevés mellett). *Legyen adott az (SP) feladat és tegyük fel, hogy* $\mathcal{F}^0 \neq \emptyset$*. Ekkor létezik szigorúan komplementáris megoldás.*

Optimális partíció, analitikus centrum
Az alábbiakban a centrális út limesz pontját tanulmányozzuk. Ehhez definiáljuk a vektorok indexhalmazának egy partícióját, melyről a következőkben további tulajdonságokat látunk be.

Vezessük be a következő jelöléseket:

$$\mathcal{B} = \{i \ : \ u_i > 0, \ \text{ valamely } \ u \in \mathcal{F}^* \ \text{ esetén}\} \, ,$$
$$\mathcal{N} = \{i \ : \ z(u)_i > 0, \ \text{ valamely } \ u \in \mathcal{F}^* \ \text{ esetén}\} \, .$$

A fent szereplő $(\mathcal{B}, \mathcal{N})$ felbontás az $\mathcal{I} = \{1, 2, \ldots, n\}$ indexhalmaz egy partíciója, amit az alábbiakban bizonyítunk be.

25.17. állítás. *Legyen adott az (SP) feladat, és tegyük fel, hogy teljesül a belső pont feltétel. Ekkor $\mathcal{I}$ indexhalmaznak a $(\mathcal{B}, \mathcal{N})$ valóban partíciója.*

Bizonyítás. $\mathcal{B} \cup \mathcal{N} = \mathcal{I}$, mert a 25.16. következmény miatt létezik $(\mathbf{u}^*, \mathbf{z}^*)$ szigorúan komplementáris megoldás. A 25.5. lemma miatt pedig $\mathcal{B} \cap \mathcal{N} = \emptyset$ is teljesül. ■

Ezt követően már jogos a következő definíció.

25.18. definíció. *Az $\mathcal{I}$ indexhalmaz $(\mathcal{B}, \mathcal{N})$ felbontását **optimális partíciónak** nevezzük.*

A centrális út az optimális megoldások halmazának egy speciális pontjához tart, amikor μ tart nullához, ez az úgynevezett analitikus centrum, amit a következőképpen definiálunk

25.19. definíció. *Jelölje $\overline{\mathbf{u}} \in \mathcal{F}^*$, $\overline{\mathbf{z}} = z(\overline{\mathbf{u}})$ azon vektorpárt, melyek maximalizálja a*

$$\prod_{i \in \mathcal{B}} u_i \prod_{i \in \mathcal{N}} z_i$$

*szorzatot az $\mathcal{F}^*$ optimális halmazon. Ekkor az $\overline{\mathbf{u}}$ vektort az $\mathcal{F}^*$ optimális halmaz **analitikus centrumának** nevezzük.*

A 25.1-20. gyakorlat szerint belső pont feltevés mellett az $\mathcal{F}^*$ halmaz korlátos, így ekkor létezik az optimális halmaz analitikus centruma. Továbbá létezik szigorúan komplementáris megoldás a 25.16. következmény alapján, így a definícióban szereplő szorzat maximumértéke pozitív, tehát az analitikus centrum egyben szigorúan komplementáris megoldás is.

25.20. tétel (Sonnevend). *Legyen a centrális út határpontja $(\mathbf{u}^*, \mathbf{z}^*)$. Ekkor $\mathbf{u}^*$ az $\mathcal{F}^*$ halmaz analitikus centruma.*

Bizonyítás. Legyen $\overline{\mathbf{u}} \in \mathcal{F}^*$ tetszőleges, és $\overline{\mathbf{z}} = z(\overline{\mathbf{u}})$. A 25.15. tétel bizonyításában leírtakhoz hasonlóan az alábbi összefüggésre jutunk:

$$\sum_{i \in \mathcal{B}} \frac{\overline{u}_i}{u_i^*} + \sum_{i \in \mathcal{N}} \frac{\overline{z}_i}{z_i^*} = n \, .$$

Mivel $u_i^* > 0\ \forall i \in \mathcal{B}$ és $z_i^* > 0\ \forall i \in \mathcal{N}$, alkalmazhatjuk a számtani-mértani közép közti egyenlőtlenséget:

$$\left(\prod_{i\in\mathcal{B}} \frac{\overline{u}_i}{u_i^*} \prod_{i\in\mathcal{N}} \frac{\overline{z}_i}{z_i^*}\right)^{\frac{1}{n}} \le \frac{1}{n}\left(\sum_{i\in\mathcal{B}} \frac{\overline{u}_i}{u_i^*} + \sum_{i\in\mathcal{N}} \frac{\overline{z}_i}{z_i^*}\right) = 1\ .$$

Így megkapjuk a kívánt

$$\prod_{i\in\mathcal{B}} \overline{u}_i \prod_{i\in\mathcal{N}} \overline{z}_i \le \prod_{i\in\mathcal{B}} u_i^* \prod_{i\in\mathcal{N}} z_i^*$$

egyenlőtlenséget. ■

25.1.3. Erős dualitás tétel

Az előzőekben az (*SP*) feladatra belső pont feltétel teljesülése mellett már bizonyítottuk a Goldman–Tucker-tételt (25.16. következmény). Célunk általános esetben is belátni a tételt. Vegyük észre, hogy a (25.1) Goldman–Tucker modell nem teljesítheti a belső pont feltételt a 25.3. tétel miatt. Ezért szükségünk van a (25.1) homogén feladat beágyazására olyan ekvivalens (*SP*) feladatba, mely teljesíti a belső pont feltételt.

Legyen $\mathbf{u}^0 = \mathbf{z}^0 = \mathbf{e}$. Ezek a vektorok pozitívak, de általában nem elégítik ki a (25.1) feltételeket. Definiáljuk az $\mathbf{r}$ hibavektort

$$\mathbf{r} = \mathbf{e} - M\mathbf{e}\ , \quad \text{és legyen} \quad \lambda = n + 1\ .$$

Ekkor

$$\begin{pmatrix} M & \mathbf{r} \\ -\mathbf{r}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{e} \\ 1 \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \lambda \end{pmatrix} = \begin{pmatrix} M\mathbf{e} + \mathbf{r} \\ -\mathbf{r}^T\mathbf{e} + \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{e} \\ 1 \end{pmatrix}\ .$$

A fenti konstrukció alapján definiáljuk a beágyazási ($\overline{SP}$) feladatot a következőképpen:

$$\min\left\{\lambda\vartheta\ :\ \begin{pmatrix} M & \mathbf{r} \\ -\mathbf{r}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{u} \\ \vartheta \end{pmatrix} - \begin{pmatrix} \mathbf{z} \\ \nu \end{pmatrix} = -\begin{pmatrix} \mathbf{0} \\ \lambda \end{pmatrix};\ \begin{pmatrix} \mathbf{u} \\ \vartheta \end{pmatrix}, \begin{pmatrix} \mathbf{z} \\ \nu \end{pmatrix} \ge \mathbf{0}\right\}\ .$$

Ez a feladat kielégíti a belső pont feltételt, mivel a csupa egyesből álló vektor megengedett megoldást ad. Ez a feladat (*SP*) alakú, ahol

$$\overline{M} = \begin{pmatrix} M & \mathbf{r} \\ -\mathbf{r}^T & 0 \end{pmatrix}\ , \qquad \overline{\mathbf{u}} = \begin{pmatrix} \mathbf{u} \\ \vartheta \end{pmatrix} \qquad \text{és} \qquad \overline{\mathbf{z}} = \begin{pmatrix} \mathbf{z} \\ \nu \end{pmatrix}\ .$$

Mivel az ($\overline{SP}$) feladat célfüggvénye a ϑ változó egy pozitív többszöröse, így ennek a változónak minden optimális megoldásban a 25.4. lemma miatt nulla az értéke. Tehát az $(\mathbf{u}, \vartheta, \mathbf{z}, \nu)$ akkor és csak akkor szigorúan komplementáris megoldása az ($\overline{SP}$) feladatnak, ha $(\mathbf{u}, \mathbf{z})$ szigorúan komplementáris megoldása a (25.1) Goldman–Tucker modellnek. Figyelembe véve, hogy az ($\overline{SP}$) feladat kielégíti a belső pont feltételt, a 25.15. tétel miatt létezik szigorúan komplementáris megoldása, ezzel beláttuk a Goldman–Tucker-tételt általános esetben is.

Az eddigieket összefoglalva: Minden lineáris programozási feladat beágyazható egy, az (*SP*) alakban adott önduális ($\overline{SP}$) feladatba oly módon, hogy $(\overline{\mathbf{u}}, \overline{\mathbf{z}}) = (\mathbf{e}, 1, \mathbf{e}, 1)$, azaz a

csupa egyesből álló vektor az ($\overline{SP}$) feladat megengedett megoldása. Továbbá a (25.1) ferdén szimmetrikus feladat bármely erősen komplementáris megoldása vagy az eredeti lineáris programozási feladat egy optimális megoldását adja, vagy bizonyítja, hogy az adott lineáris programozási feladatnak nincs optimális megoldása.

A fenti eredmények segítségével könnyen bebizonyítható az úgynevezett erős dualitás tétel.

25.21. tétel (erős dualitás tétel). *Legyenek a* (P) *és a* (D) *feladatok adottak. Ekkor a következő két alternatíva valamelyike teljesül:*

1. $\mathcal{P} \neq \emptyset$ *és* $\mathcal{D} \neq \emptyset$, *valamint létezik* $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ *szigorúan komplementáris megoldás.*
2. $\mathcal{P}^* = \emptyset$ *és* $\mathcal{D}^* = \emptyset$, *ez pontosan akkor fordulhat elő, ha létezik* $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, *melyre*

$$A\overline{\mathbf{x}} \geq \mathbf{0}\,, \quad A^T\overline{\mathbf{y}} \leq \mathbf{0}\,, \quad \mathbf{c}^T\overline{\mathbf{x}} \leq \mathbf{b}^T\overline{\mathbf{y}}\,.$$

Bizonyítás. A 25.16. következmény miatt a lineáris programozási feladat Goldman–Tucker rendszerének van egy erősen komplementáris megoldása. Egy ilyen megoldásban vagy $\xi > 0$, és ebben az esetben a 25.3. tétel 3. pontjából következik, hogy létezik optimális megoldás pár nulla dualitásréssel, vagy $\zeta > 0$ a Goldman–Tucker rendszer erősen komplementáris megoldásában. Az utóbbi esetben a 25.3. tétel 4. pontja miatt (P) vagy (D) vagy mindkettő nemmegengedett. ■

A lineáris programozás fontos és alapvető eredménye a több mint 100 éves Farkas-lemma, melyre a Goldman–Tucker-tétel segítségével most egy újabb bizonyítást adunk.

25.22. lemma (Farkas-lemma). *A következő két egyenlőtlenségrendszer közül pontosan az egyiknek van megoldása*

$$\left.\begin{array}{rl} A\mathbf{x} & \geq \mathbf{b}\,, \\ \mathbf{x} & \geq \mathbf{0} \end{array}\right\} \qquad \textit{és} \qquad \left.\begin{array}{rl} A^T\mathbf{y} & \leq \mathbf{0}\,, \\ \mathbf{b}^T\mathbf{y} & > 0\,, \\ \mathbf{y} & \geq \mathbf{0} \end{array}\right\}\,,$$

ahol $A \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$.

Bizonyítás. Tekintsük a következő primál és duál lineáris programozási feladatot

$$\begin{array}{ll} (P) & \min\left\{\mathbf{0}^T\mathbf{x} \;:\; A\mathbf{x} \geq \mathbf{b},\;\; \mathbf{x} \geq \mathbf{0}\right\}\,, \\ (D) & \max\left\{\mathbf{b}^T\mathbf{y} \;:\; A^T\mathbf{y} \leq \mathbf{0},\;\; \mathbf{y} \geq \mathbf{0}\right\}\,. \end{array}$$

Elkészíthető a (P) és (D) feladathoz az (SP) ferdén szimmetrikus önduális feladat

$$\left.\begin{array}{rrrl} \min\ \mathbf{0}^T\mathbf{y} & +\mathbf{0}^T\mathbf{x} & +0\xi & \\ & A\mathbf{x} & -\mathbf{b}\xi & \geq \mathbf{0}\,, \\ -A^T\mathbf{y} & & +\mathbf{0}\xi & \geq \mathbf{0}\,, \\ \mathbf{b}^T\mathbf{y} & -\mathbf{0}^T\mathbf{x} & & \geq \mathbf{0}\,, \\ & & \mathbf{y},\mathbf{x},\xi & \geq \mathbf{0} \end{array}\right\} \quad (SP)\,.$$

Az előző tételek alapján ennek a feladatnak létezik szigorúan komplementáris megoldása, $(\overline{\mathbf{y}}, \overline{\mathbf{x}}, \overline{\xi})$. Vezessük be a következő jelöléseket:

$$s(\overline{\mathbf{y}}) = -A^T\overline{\mathbf{y}}\,, \qquad s(\overline{\mathbf{x}}) = A\overline{\mathbf{x}} - \mathbf{b}\overline{\xi}\,, \qquad s(\overline{\xi}) = \mathbf{b}^T\overline{\mathbf{y}}\,.$$

Ekkor két eset lehetséges: (i) $\overline{\xi} > 0$, illetve (ii) $\overline{\xi} = 0$.
Az (i) esetben $\overline{\mathbf{x}} \geq \mathbf{0}$, $s(\overline{\mathbf{x}}) \geq \mathbf{0}$, így $A\overline{\mathbf{x}} > \mathbf{b}\overline{\xi}$, tehát az $\overline{\mathbf{x}}/\overline{\xi}$ megoldása az első egyenletrendszernek.
Az (ii) esetben – mivel $\overline{\xi} = 0 - s(\overline{\xi}) > 0$, tehát $\mathbf{b}^T\overline{\mathbf{y}} > 0$, $\overline{\mathbf{y}} \geq \mathbf{0}$. Valamint $s(\overline{\mathbf{y}}) \geq \mathbf{0}$ miatt $A^T\overline{\mathbf{y}} \leq 0$, azaz ebben az esetben az $\overline{\mathbf{y}}$ vektor megoldása a második rendszernek. A két rendszernek nem lehet egyszerre megoldása, ami elemien belátható. ■

Gyakorlatok

25.1-1. Legyen $\mathbf{x}$ és $\mathbf{y}$ primál, illetve duál megengedett megoldás, ekkor a következő két állítás ekvivalens:

1. $\mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y} = (\mathbf{c} - A^T\mathbf{y})^T\mathbf{x} + \mathbf{y}^T(A\,\mathbf{x} - \mathbf{b}) = 0$, illetve
2. $(\mathbf{c} - A^T\mathbf{y})\,\mathbf{x} = \mathbf{0}$ és $\mathbf{y}\,(A\,\mathbf{x} - \mathbf{b}) = \mathbf{0}$.

25.1-2. Bizonyítsuk be a *gyenge egyensúlyi tételt*.
25.1-3. Bizonyítsuk be a 25.3. tétel 1. és 3. állítását.
25.1-4. Legyen $A \in \mathbb{R}^{m\times k}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^k$ adott mátrix, illetve vektorok. Tekintsük a (P) és (D) kanonikus lineáris programozási feladatokat. Tegyük fel, hogy létezik $\bar{\mathbf{x}}$ megengedett megoldása a (P) feladatnak, amelyre az $A\,\bar{\mathbf{x}} \geq \mathbf{0}$ és $\mathbf{c}^T\bar{\mathbf{x}} < 0$ összefüggések is teljesülnek. Ekkor nem létezik $\mathbf{y} \in \mathbb{R}^m$ megengedett megoldása a (D) feladatnak.
25.1-5. Legyen $A \in \mathbb{R}^{m\times k}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^k$ adott mátrix, illetve vektorok. Bizonyítsuk be, hogy az

$$M = \begin{pmatrix} 0 & A & -\mathbf{b} \\ -A^T & 0 & \mathbf{c} \\ \mathbf{b}^T & -\mathbf{c}^T & 0 \end{pmatrix},$$

mátrix ferdén szimmetrikus, azaz $M = -M^T$.
25.1-6. Bizonyítsuk be, hogy az (SP) feladat önduális.
25.1-7. Bizonyítsuk be a 25.4. lemmát.
25.1-8. Tekintsük a következő lineáris programozási feladatot

$$\begin{array}{lrrrrcr} \min & -17\,x_1 & +13\,x_2 & +2\,x_3 & -8\,x_4 & & \\ & 2\,x_1 & - \;\; x_2 & & - \;\; x_4 & \leq & 4 \\ & -\;\; x_1 & & + \;\; x_3 & - \;\; x_4 & \geq & -3 \\ & x_1 & +3\,x_2 & -2\,x_3 & & \leq & 4\,, \end{array}$$

ahol $x_1,\ x_2,\ x_3 \geq 0$.

a. Írjuk fel a primál és duál lineáris programozási feladatot kanonikus alakban.
b. Adjuk meg a komplementaritási feltételeket.
c. Írjuk fel a Goldman–Tucker feladatot. Határozzuk meg az M mátrixot.

25.1-9. Tekintsük a következő lineáris programozási feladatot

$$\begin{array}{lrrrrrcl} \min & & & & & x_6 & & \\ & & 2x_2 & +x_3 & +\ x_4 & -x_5 & = & 0 \\ & 3x_1 & -2x_2 & -x_3 & +2x_4 & & \leq & 15 \\ & x_1 & & +x_3 & & & \leq & 5 \\ & -9x_1 & +8x_2 & +x_3 & -2x_4 & -x_6 & = & 0\,, \end{array}$$

ahol $x_1, x_2, \ldots, x_5 \geq 0$ és az x_6 előjelkötetlen változó.

a. Írjuk fel a primál és duál lineáris programozási feladatot kanonikus alakban.

b. Adjuk meg a komplementaritási feltételeket.

c. Írjuk fel a Goldman–Tucker feladatot. Határozzuk meg az M mátrixot.

25.1-10. Legyenek a következő adatok egy kanonikus primál feladat adatai

$$\mathbf{c} = \begin{pmatrix} -9 \\ 8 \\ 1 \\ -2 \end{pmatrix}, \qquad A = \begin{pmatrix} 3 & -2 & -1 & 2 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 15 \\ 5 \end{pmatrix}.$$

a. Írjuk fel a kanonikus primál feladatot.

b. Keressünk egy pozitív primál megoldást.

c. Írjuk fel a duál feladatot és keressünk egy (lehetőleg pozitív) duál megoldást. *Útmutatás.* Ha nem tudunk duál megengedett megoldást találni, akkor próbáljunk meg olyan primál megengedett megoldást találni, amelyikre $A\mathbf{x} \geq \mathbf{0}$ és $\mathbf{c}^T\mathbf{x} < 0$ teljesül. Mit jelent ez?

d. Írjuk fel a Goldman–Tucker feladatot és keressünk egy pozitív megoldást, mely az első hat feltételt kielégíti. Mit jelent az, ha ilyen nem létezik? Vizsgáljuk meg, hogy teljesüle a hetedik feltétel is?

25.1-11. Legyenek a következő adatok egy kanonikus primál feladat adatai

$$\mathbf{c} = \begin{pmatrix} \frac{19}{10} \\ \frac{9}{5} \\ \frac{2}{5} \end{pmatrix}, \qquad A = \begin{pmatrix} \frac{1}{10} & \frac{11}{10} & -\frac{3}{5} \\ 1 & -\frac{3}{10} & -\frac{4}{5} \\ -\frac{1}{5} & 0 & \frac{4}{5} \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} -\frac{2}{5} \\ -\frac{11}{10} \\ -\frac{2}{5} \end{pmatrix}.$$

a. Írjuk fel a kanonikus primál feladatot.

b. Keressünk egy pozitív primál megoldást.

c. Írjuk fel a duál feladatot és keressünk egy pozitív duál megoldást.

d. Mi következik abból, hogy létezik belső pont mind a primál, mind pedig a duál feladat esetén?

e. Írjuk fel a Goldman–Tucker feladatot és keressünk egy olyan pozitív megoldást, amely kielégíti az első hat feltételt, ám a ζ értéke -6 lesz. (Ha a dualitásrést a következő alakban írjuk fel, akkor $\mathbf{x}^T\mathbf{s} + \mathbf{y}^T\mathbf{t} = 6$ kell, hogy teljesüljön a megoldására.)

25.1-12. Bizonyítsuk be a 25.7. állítást. *Útmutatás.* Alkalmazzunk indirekt bizonyítást. Tegyük fel, hogy az $\bar{M}$ mátrix szinguláris.

25.1-13. Bizonyítsuk be, hogy ha U, Z pozitív diagonális, továbbá M ferdén szimmetrikus mátrix, akkor $Z + UM$ reguláris mátrix.

25.1-14. Tekintsük a 25.8. állítást.

a. Határozzuk meg a legnagyobb α lépéshosszt, melyre az $\mathbf{x}^+ > \mathbf{0}$ és $\mathbf{s}^+ > \mathbf{0}$ teljesül.

b. A $\mathbf{w}^+ \in \operatorname{int} \mathcal{T}$ elvárásból, koordinátánként a következő két összefüggés valamelyikét nyerjük:

$$w_i < w_i^+ < \hat{w}_i \quad \text{vagy} \quad \hat{w}_i < w_i^+ < w_i \,.$$

Milyen lépéshossz korlátok számíthatók ki az első, illetve második egyenlőtlenségből?

25.1-15. Tekintsük a 25.1-11. gyakorlatban szereplő lineáris programozási feladatot.

a. Írjuk fel a centrális út feladatot arra a megoldásra, amelyikhez tartozó dualitásrés értéke 6 és határozzuk meg a μ paraméter értékét.

b. Legyenek $\mathbf{w}^T = (1, 1, 1, 1, 1, 1)$ és $\hat{\mathbf{w}}^T = (1/2, 1/2, 1/2, 2, 2, 2)$ adott pontok. Tegyünk egy Newton-lépést a $\mathbf{u}\,\mathbf{z} = \mathbf{w}$ pontból a $\hat{\mathbf{w}}$ pont felé, azaz számítsuk ki a $\Delta\mathbf{u}$ és $\Delta\mathbf{z}$ vektorokat. Legyen az M mátrix a Goldman–Tucker feladatnál, az előző adatokkal meghatározott mátrixnak, a 6×6-os bal felső részmátrixa, míg a $\mathbf{q}^T = (\mathbf{b}^T\,, -\mathbf{c}^T)$. *Útmutatás.* A számításokat a MATLAB programcsomaggal végezzük, mert úgy gyorsabb.

c. Határozzuk meg az α lépéshosszt úgy, hogy az új vektorok a feladat szigorúan pozitív megoldásai legyenek.

25.1-16. Bizonyítsuk be a 25.9. lemmát. *Útmutatás.* Számoljuk ki az $(\mathbf{u} - \bar{\mathbf{u}})^T M\,(\mathbf{u} - \bar{\mathbf{u}})$ értéket, ahol $\bar{\mathbf{u}} \in \mathcal{F}^0$ és $(\mathbf{u}, \mathbf{z}) \in \mathcal{L}_K$, majd pedig számítsunk ki korlátokat az u_j és z_j koordinátákra.

25.1-17. Bizonyítsuk be a 25.10. lemmát. *Útmutatás.* Hasonló az előző gyakorlathoz, de itt vegyük figyelembe az $\mathcal{L}_{\mathbf{w}}$ általánosított szinthalmaz definícióját és azt is, hogy az $f : \mathbb{R}_+^{2n} \to \mathbb{R}_+^n$, $f(\mathbf{x}, \mathbf{s}) = \mathbf{x}\,\mathbf{s}$ függvény folytonos. Milyen az f függvény ősképe?

25.1-18. Bizonyítsuk be a 25.11. lemmát. *Útmutatás.* Azt kell belátni, hogy a $\mathcal{G}$ halmaz tartalmazza az összes limeszpontját.

25.1-19. Bizonyítsuk be, hogy a 25.12. tétel bizonyításában szereplő f függvény esetén $\|f\|_\infty$ felveszi a minimumát az $\mathcal{A} \cap \mathcal{G}$ halmazon. *Útmutatás.* Bizonyítsuk, be, hogy $\|f\|_\infty$ függvény folytonos, majd alkalmazzuk rá a Weierstrass-tételt.

25.1-20. Legyen adott az (*SP*) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Bizonyítsuk be, hogy az $\mathcal{F}^*$ halmaz kompakt.

25.1-21. Legyen $M \in \mathbb{R}^{n \times n}$ egy ferdén szimmetrikus mátrix. Bizonyítsuk be, hogy bármely $\mathbf{z} \in \mathbb{R}^n \setminus \{0\}$ vektor esetén létezik olyan j index, melyre $z_j \neq 0$ és $z_j(M\mathbf{z})_j \geq 0$. *Útmutatás.* Legegyszerűbb indirekt bizonyítást adni az állításra.

25.1-22. Bizonyítsuk be a 25.14. tétel 2. állításában szereplő $(\mathbf{u}, \mathbf{z})$ vektorok egyértelműségét. *Útmutatás.* Alkalmazzunk indirekt bizonyítást és használjuk fel az előző gyakorlat állítását.

25.1-23. Bizonyítsuk be a 25.15. tétel 1. és 2. állítását.

25.2. Dikin-féle affin skálázású algoritmus

Az előző részben megmutattuk, hogy ha létezik belső pont, akkor a centrális utat követve a ferdén szimmetrikus önduális feladat megoldásához tartunk. Ennek a résznek a célja bizonyítani, hogy ezt a gondolatmenetet követve *polinom időben* a ferdén szimmetrikus önduális

feladat megoldásához tetszőlegesen közeli pontot tudunk előállítani, azaz tetszőleges $\varepsilon > 0$ számhoz olyan $(\mathbf{u}, \mathbf{z})$ pontpárt, amely a megengedett tartományban van és $\mathbf{u}^T\mathbf{z} \le \varepsilon$. E célból egy konkrét algoritmust, egy affin skálázású primál-duál módszert ismertetünk, melynek első változatát Dikin orosz matematikus 1967-ben fogalmazta meg. Ezt követően a $(\mathcal{B}, \mathcal{N})$ partíció meghatározásáról ejtünk pár szót, majd ennek segítségével ismertetjük a kerekítési eljárást, azaz azt, hogy a Dikin-módszerrel kapott jó közelítő megoldásból milyen módon lehet egy szigorúan komplementáris megoldást előállítani (a szakirodalomban például az ellipszoid módszer estén is találkozhatunk kerekítési eljárással). Ezzel eloszlatjuk azt a tévhitet, ami a mai napig a köztudatban él, miszerint lineáris optimalizálási feladat pontos megoldása belsőpontos módszerrel polinom időben nem állítható elő.

Csökkenési irány meghatározása
Tekintsük az (SP) feladatot, és legyen $\mathbf{u} \in \mathcal{F}^0,\ \mathbf{z} = z(\mathbf{u}) > 0$. Keressük $(\Delta\mathbf{u}, \Delta\mathbf{z}) \in \mathbb{R}^{2n}$:

$$\begin{aligned} -M\Delta\mathbf{u} + \Delta\mathbf{z} &= \mathbf{0}\ , \\ \mathbf{u} + \Delta\mathbf{u},\ \mathbf{z} + \Delta\mathbf{z} &\ge \mathbf{0}\ . \end{aligned}$$

Ennél mi többet szeretnénk elérni, mégpedig: $\mathbf{u}^+ = \mathbf{u} + \Delta\mathbf{u} > \mathbf{0}$ és $\mathbf{z}^+ = \mathbf{z} + \Delta\mathbf{z} > \mathbf{0}$, valamint, hogy közben csökkenjen a dualitásrés is. Felhasználva, hogy M ferdén szimmetrikus mátrix, az új dualitásrés értéke:

$$\begin{aligned} \mathbf{q}^T\mathbf{u}^+ &= (\mathbf{u}^+)^T\mathbf{z}^+ = (\mathbf{u} + \Delta\mathbf{u})^T(\mathbf{z} + \Delta\mathbf{z}) = \mathbf{u}^T\mathbf{z} + \mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T\Delta\mathbf{z} + (\Delta\mathbf{u})^T\Delta\mathbf{z} = \\ &= \mathbf{u}^T\mathbf{z} + \mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T\Delta\mathbf{z} = \mathbf{q}^T\mathbf{u} + \mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T\Delta\mathbf{z}\ . \end{aligned}$$

A dualitásrés csökkenésének a feltétele az, hogy

$$\mathbf{q}^T\mathbf{u}^+ < \mathbf{q}^T\mathbf{u}, \qquad \text{azaz} \qquad \mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T\Delta\mathbf{z} < 0\ .$$

Ebben az esetben a $(\Delta\mathbf{u}, \Delta\mathbf{z}) \in \mathbb{R}^{2n}$ vektort ***csökkenési iránynak*** nevezzük. Célunk a (lokálisan) legjobb csökkenési irány keresése, ami a következőképpen fogalmazható meg:

$$\left.\begin{aligned} \min\{\mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T\Delta\mathbf{z}\} \\ -M\Delta\mathbf{u} + \Delta\mathbf{z} = \mathbf{0} \\ \mathbf{u} + \Delta\mathbf{u} \ge \mathbf{0}\ ,\ \mathbf{z} + \Delta\mathbf{z} \ge \mathbf{0} \end{aligned}\right\} \quad (IF)\ .$$

A fenti ***iránymeghatározási segédfeladatot*** szeretnénk megoldani, de ez egy – az eredetivel azonos „nehézségű” – lineáris programozási feladat. Nyilvánvaló, hogy egy olyan számítási eljárás, amelynek egy-egy iterációjában az eredeti feladattal ekvivalens segédfeladatot kellene megoldani, nem igazán kecsegtet sikerrel. Régóta köztudott viszont, hogy lineáris függvényt könnyen lehet minimalizálni (zárt) gömb felett (lásd 25.2-1. gyakorlat), hiszen legyen $\mathbf{c} \in \mathbb{R}^n$, akkor a

$$\min_{\|\mathbf{u}\| \le 1} \mathbf{c}^T\mathbf{u} \tag{25.9}$$

feladat optimális megoldása az $\mathbf{u} = -\mathbf{c}/\|\mathbf{c}\|$. Másfelől egy tetszőleges adott dimenziós, zárt ellipszoid, egy ugyanolyan dimenziós zárt gömbbé transzformálható át (lásd 25.2-2. gyakorlat).

Dikin ötlete (1967): Keressünk egy könnyebben megoldható feladatot, amely dualitásrés csökkenést biztosít. Tegyük ezt úgy, hogy a pozitivitási megkötést „relaxáljuk”. A közelítést

egy (konvex) zárt ellipszoiddal oldjuk meg.
Legyen $\mathbf{u},\ \mathbf{z} > 0$, és ekkor a ***Dikin-ellipszoid:***

$$\mathcal{E}_D(\mathbf{u},\mathbf{z}) = \left\{(\bar{\mathbf{u}},\ \bar{\mathbf{z}}) \in \mathbb{R}^{2n}\ :\ \bar{\mathbf{u}} = \mathbf{u} + \Delta\mathbf{u},\ \bar{\mathbf{z}} = \mathbf{z} + \Delta\mathbf{z},\ \left\|\frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}}\right\| \leq 1\right\}\ .$$

Legyen

$$\mathcal{F}_z^0 = \{(\mathbf{u},\mathbf{z}) \in \mathbb{R}^{2n}\ :\ \mathbf{u} \in \mathcal{F}^0\ ,\ \mathbf{z} > \mathbf{0}\}\ .$$

Ekkor $\mathcal{F}_z^0 \cap \mathcal{E}_D(\mathbf{u},\mathbf{z}) \neq \emptyset$, mert $(\mathbf{u},\mathbf{z}) \in \mathcal{F}_z^0 \cap \operatorname{int}\mathcal{E}_D(\mathbf{u},\mathbf{z})$. Belátható, hogy $\mathcal{F}_z^0 \cap \mathcal{E}_D(\mathbf{u},\mathbf{z})$ egy nem üres kompakt halmaz.

Legyen $(\mathbf{u}+\Delta\mathbf{u},\mathbf{z}+\Delta\mathbf{z}) \in \mathcal{F}_z^0 \cap \mathcal{E}_D(\mathbf{u},\mathbf{z})$, ekkor

$$1 \geq \left\|\frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}}\right\| = \left\|\frac{\mathbf{z}\Delta\mathbf{u} + \mathbf{u}\Delta\mathbf{z}}{\mathbf{u}\mathbf{z}}\right\| = \left\|\frac{\mathbf{z}\Delta\mathbf{u} + \mathbf{u}M\Delta\mathbf{u}}{\mathbf{u}\mathbf{z}}\right\| =$$

$$= \|(XS)^{-1}(S + XM)\Delta\mathbf{u}\| = \|S^{-1}(X^{-1}S + M)\Delta\mathbf{u}\|\ .$$

A 25.2-3. és 25.2-4. gyakorlat alapján $S^{-1}(X^{-1}S + M)$ pozitív definit mátrix, ezért az

$$\|S^{-1}(X^{-1}S + M)\Delta\mathbf{u}\|^2$$

a $\Delta\mathbf{u}$ konvex kvadratikus függvénye, azaz

$$\{\Delta\mathbf{u} : \|S^{-1}(X^{-1}S + M)\Delta\mathbf{u}\| \leq 1\}$$

korlátos konvex nemüres zárt halmaz. Vagyis az (IF) feladat relaxálható a következő módon:

$$\left.\begin{array}{r}\min\{\mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T M\Delta\mathbf{u}\} \\ \|S^{-1}(X^{-1}S + M)\Delta\mathbf{u}\| \leq 1\end{array}\right\} \quad (\overline{IF})\ .$$

Az $(\overline{IF})$ lineáris célfüggvény minimalizálása konvex kompakt nem üres halmaz felett, így a halmaz korlátosságát is figyelembe véve, a Weierstrass-tétel miatt létezik megoldása.

Átskálázás
Az $(\overline{IF})$ feladat helyett tekinthetjük a

$$\left.\begin{array}{r}\min\{\mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T\Delta\mathbf{z}\} \\ -M\Delta\mathbf{u} + \Delta\mathbf{z} = \mathbf{0}, \\ \|\frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}}\| \leq 1\end{array}\right\} \quad (SF)$$

segédfeladatot, amelyik esetén az első feltétel egy homogén lineáris egyenletrendszer, míg a második egy zárt elliptikus henger, amellyel a nemnegativitási feltételeket relaxáltuk.

A célunk az, hogy olyan ***skálázást*** (koordináta transzformációt) vezessünk be, amely után (SF) lineáris célfüggvényét egy gömbön kell minimalizálnunk.
Legyen

$$\mu = \frac{\mathbf{u}^T\mathbf{z}}{n}\ ,\quad \mathbf{d} = \sqrt{\frac{\mathbf{u}}{\mathbf{z}}}\ ,\quad \mathbf{v} = \sqrt{\frac{\mathbf{u}\mathbf{z}}{\mu}}\ .$$

Ekkor $\sqrt{\mathbf{uz}} = \sqrt{\mu}\mathbf{v}$ és $\mathbf{d}^{-1}\mathbf{u} = \sqrt{\mu}\mathbf{v} = \mathbf{dz}$. Vezessük be a $\mathbf{p}_u$ és $\mathbf{p}_z$ vektorokat úgy, hogy

$$\mathbf{p}_u = \frac{\mathbf{d}^{-1}\Delta\mathbf{u}}{\sqrt{\mu}} \quad \text{és} \quad \mathbf{p}_z = \frac{\mathbf{d}\Delta\mathbf{z}}{\sqrt{\mu}} .$$

Ekkor

$$\frac{\Delta\mathbf{u}}{\mathbf{u}} = \frac{\mathbf{d}^{-1}\Delta\mathbf{u}}{\mathbf{d}^{-1}\mathbf{u}} = \frac{\sqrt{\mu}\mathbf{p}_u}{\sqrt{\mu}\mathbf{v}} = \frac{\mathbf{p}_u}{\mathbf{v}} \quad \text{és} \quad \frac{\Delta\mathbf{z}}{\mathbf{z}} = \frac{\mathbf{d}\Delta\mathbf{u}}{\mathbf{du}} = \frac{\sqrt{\mu}\mathbf{p}_z}{\sqrt{\mu}\mathbf{v}} = \frac{\mathbf{p}_z}{\mathbf{v}} ,$$

amiből a

$$\frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}} = \frac{\mathbf{p}_u + \mathbf{p}_z}{\mathbf{v}}$$

összefüggés következik. Bevezetve a $\mathbf{p} = \mathbf{p}_u + \mathbf{p}_z$ jelölést a

$$\frac{\mathbf{p}}{\mathbf{v}} = \frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}} ,$$

illetve a

$$\mathbf{z}\Delta\mathbf{u} + \mathbf{u}\Delta\mathbf{z} = (\mathbf{zd})(\mathbf{d}^{-1}\Delta\mathbf{u}) + (\mathbf{ud}^{-1})(\mathbf{d}\Delta\mathbf{z}) = \sqrt{\mu}\mathbf{v}\sqrt{\mu}\mathbf{p}_u + \sqrt{\mu}\mathbf{v}\sqrt{\mu}\mathbf{p}_z = \mu\mathbf{vp}$$

egyenleteket kapjuk. Tehát az (SF) átskálázott alakja rögzített μ paraméter mellett

$$\left.\begin{array}{l} \min\ \mu\mathbf{v}^T\mathbf{p} \\ \|\frac{\mathbf{p}}{\mathbf{v}}\| \le 1 \ . \end{array}\right\}$$

Legyen $\bar{\mathbf{p}} = \mathbf{p}/\mathbf{v}$. Ekkor az (SF) a következő alakot ölti:

$$\left.\begin{array}{l} \min\ \mu(\mathbf{v}^2)^T\bar{\mathbf{p}} \\ \|\bar{\mathbf{p}}\| \le 1 \ , \end{array}\right\}$$

ami gömb felett egy lineáris függvény minimalizálása, tehát létezik egyértelmű minimuma:

$$\bar{\mathbf{p}} = -\frac{\mathbf{v}^2}{\|\mathbf{v}^2\|} , \quad \text{és ekkor} \quad \mathbf{p} = -\frac{\mathbf{v}^3}{\|\mathbf{v}^2\|} .$$

A $(\Delta\mathbf{u}, \Delta\mathbf{z})$ csökkenési irány meghatározása
Mivel $\mathbf{0} = -M\Delta\mathbf{u} + \Delta\mathbf{z} = -MD(D^{-1}\Delta\mathbf{u}) + D^{-1}(D\Delta\mathbf{z}) = -\sqrt{\mu}(MD\mathbf{p}_u - D^{-1}\mathbf{p}_z)$, ezért

$$\mathbf{p}_z = DMD\mathbf{p}_u , \quad \text{és így a} \quad \mathbf{p} = \mathbf{p}_u + \mathbf{p}_z = (I + DMD)\mathbf{p}_u .$$

A 25.2-3. gyakorlat szerint $I + DMD$ pozitív definit mátrix, így

$$\mathbf{p}_u = (I + DMD)^{-1}\mathbf{p} \quad \text{és} \quad \mathbf{p}_z = DMD(I + DMD)^{-1}\mathbf{p} .$$

Figyelembe véve a $\mathbf{d}^{-1}\Delta\mathbf{u} = \sqrt{\mu}\mathbf{p}_u$ és $\mathbf{d}\Delta\mathbf{z} = \sqrt{\mu}\mathbf{p}_z$ összefüggéseket

$$\Delta\mathbf{u} = \sqrt{\mu}D(I + DMD)^{-1}\mathbf{p} \quad \text{és} \quad \Delta\mathbf{z} = \sqrt{\mu}MD(I + DMD)^{-1}\mathbf{p}$$

adódik. Az így kapott $(\Delta\mathbf{u}, \Delta\mathbf{z})$ vektort ***Dikin-iránynak*** nevezzük.
Ezt meglépve a dualitásrés csökkenése:

$$\mu(\mathbf{v}^2)^T\bar{\mathbf{p}} = \mu(\mathbf{v}^2)^T\left(-\frac{\mathbf{v}^2}{\|\mathbf{v}^2\|}\right) = -\mu\frac{\|\mathbf{v}^2\|^2}{\|\mathbf{v}^2\|} = -\mu\|\mathbf{v}^2\| = -\mu\left\|\frac{\mathbf{uz}}{\mu}\right\| = -\|\mathbf{uz}\| .$$

A $\Delta\mathbf{u}$ vektorra kapott kifejezésbe behelyettesítve $\mathbf{d}$, $\mathbf{p}$, illetve $\mathbf{p}$ képletébe $\mathbf{v}$ definícióját, könnyen adódik a következő állítás.

25.23. állítás. *A fenti jelölésekkel*

$$\Delta\mathbf{u} = \sqrt{\mu}D(I + DMD)^{-1}\mathbf{p} = -(S + XM)^{-1}\frac{\mathbf{u}^2\mathbf{z}^2}{\|\mathbf{uz}\|} \qquad \textit{teljesül.}$$

A 25.23. állítás első egyenlőségét már bizonyítottuk, a másodiknak a bizonyítását, amelyik az eredeti adatokat használja, az Olvasóra bízzuk (lásd 25.2-5. gyakorlat).

Az algoritmus során szükségünk van a pontunk pozitivitásának megtartására, melyet a következő ***centralitási mérték***

$$\partial_c(\mathbf{u}) = \frac{\max(\mathbf{u}z(\mathbf{u}))}{\min(\mathbf{u}z(\mathbf{u}))} = \frac{\max(\mathbf{v}^2)}{\min(\mathbf{v}^2)} = \delta_c(\mathbf{v})\,, \qquad \mathbf{u} \in \mathcal{F}^0$$

segítségével teszünk meg, ahol $\max(\mathbf{v})$, illetve $\min(\mathbf{v})$ a $\mathbf{v}$ vektor legnagyobb, illetve legkisebb koordinátáját jelöli. Ekkor $\partial_c(\mathbf{u}) \geq 1$ teljesül bármely $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} = z(\mathbf{u}) > \mathbf{0}$ esetén, valamint $\partial_c(\mathbf{u}) = 1$ pontosan akkor, ha $\mathbf{u} \in \mathcal{F}^0 :\ \mathbf{u}\,z(\mathbf{u}) = \mu\,\mathbf{e}$, azaz a minimális centralitási mértékű megoldások a centrális úton vannak.

Nevezzünk egy α ***lépéshosszt megengedettnek,*** ha az új pontunkra is teljesül a pozitivitási megkötés, azaz $\mathbf{u} + \alpha\Delta\mathbf{u} > \mathbf{0},\ \mathbf{z} + \alpha\Delta\mathbf{z} > \mathbf{0}$.

Mind az algoritmus elméleti elemzése során, mind pedig a gyakorlati számítógépes megvalósításakor két célunk van: az $\alpha > 0$ lépéshosszt úgy meghatározni, hogy (i) az $\mathbf{u} + \alpha\Delta\mathbf{u} > \mathbf{0}$ és $\mathbf{z} + \alpha\Delta\mathbf{z} > \mathbf{0}$, valamint (ii) a pontunk centralitási mértéke egy adott korlát alatt maradjon, azaz $\partial_c(\mathbf{u}) \leq \tau$ teljesüljön[8], valamely adott $\tau > 1$ szám esetén.

Legyen $\tau > 1,\ \tau \in \mathbb{R}$ és $\mathbf{u} \in \mathcal{F},\ \partial_c(\mathbf{u}) \leq \tau$. Továbbá mivel $\mathbf{v} = \sqrt{\mathbf{u}\,\mathbf{z}/\mu}$, ekkor a

$$\partial_c(\mathbf{u}) = \frac{\max(\mathbf{v}^2)}{\min(\mathbf{v}^2)} \leq \tau \iff \max(\mathbf{v}^2) \leq \tau\min(\mathbf{v}^2)$$

összefüggésből következik, hogy létezik $\tau_1, \tau_2 > 0$ és $\tau_2 = \tau\tau_1$, amelyekre

$$\tau_1\mathbf{e} \leq \mathbf{v}^2 \leq \tau_2\mathbf{e}\ .$$

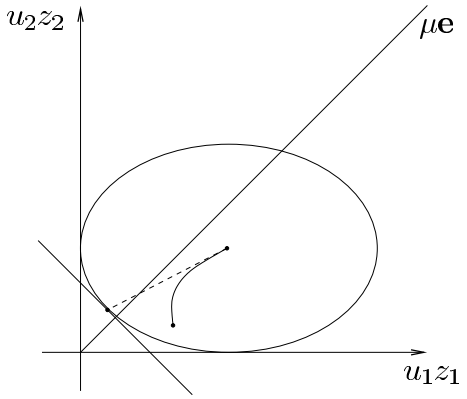
Az eddigieket összefoglalva, az algoritmusunk a következő:
Bemenő adatként meg kell adnunk az $\varepsilon > 0$ megállási paramétert, a $\tau \geq 1$ környezeti paramétert, az $\alpha > 0$ lépés paramétert, illetve egy $\mathbf{u}^0$ belső pontot, ami a megadott τ környezeten belül helyezkedik el, azaz $\mathbf{u}^0 \in \mathcal{F}^0$, $z(\mathbf{u}^0) > \mathbf{0}$ és $\partial_c(\mathbf{u}^0) \leq \tau$.

DIKIN-FÉLE-AFFIN-SKÁLÁZÁS

```
1  u ← u^0
2  z ← z(u^0)
3  while u^T z ≥ ε
4        do Δu ← −(S + XM)^{−1} u^2 z^2 / ||u z||
5           számítsuk ki az α > 0 megengedett lépéshosszt
6           u ← u + αΔu
7           z ← z(u)
```

Az algoritmus egy lépését a 25.2. ábra szemlélteti.

[8] A $\partial_c(\mathbf{u}) \leq \tau$ egyenlőtlenség a centrális útnak egy ún. ***τ-környezetét*** jelöli ki.



25.2. ábra. A Dikin-algoritmus egy iterációja az átskálázott térben, ahol $u_1z_1 = \mu v_1^2$ és $u_2z_2 = \mu v_2^2$.

A DIKIN-FÉLE-AFFIN-SKÁLÁZÁS fenti változata kétféleképpen is elemezhető: gyakorlati, illetve elméleti szempontból. A folytonos optimalizálásban gyakran megfigyelt jelenség, hogy az iteratív algoritmusok gyakorlatban megfigyelt lépésszáma messze felülmúlja az elméletileg bizonyítható lépésszámot főleg a hosszú lépéses és nagy környezetes belsőpontos algoritmusok esetén. Az elméleti hatékonyság esetén az ún. *legkedvezőtlenebb eset elemzése* a cél. Ezzel szemben a gyakorlatban az a cél, hogy minél jobban kihasználjuk a megoldandó feladat összes tulajdonságát, még akkor is, ha az általános elméleti elemzések során nem minden apró részletet tudunk figyelembe venni.

25.2.1. Dikin-algoritmus: gyakorlati szempontok

A DIKIN-FÉLE-AFFIN-SKÁLÁZÁSI-ALGORITMUS számítógépes megvalósítása során két paraméternek van jelentős hatása: a környezeti paraméter, $\tau \geq 1$ megválasztásának, illetve a τ rögzítése után, a lehető legnagyobb α megengedett lépéshossz meghatározásának.

A gyakorlati alkalmazások során akár $\tau = 10\,000$ is lehet. Ekkor például $\tau_1 = 1/100$ és $\tau_2 = 100$ is lehetséges.[9]

A megengedett lépéshossz, $\alpha > 0$ megválasztásánál, mohó módon járunk el, azaz minden iterációban, miután meghatároztuk a csökkenési irányt, kiszámoljuk a maximális megengedett $\alpha > 0$ lépéshosszt.[10] Először azt szeretnénk elérni, hogy az új megoldás vektorok pozitívak legyenek, azaz $\mathbf{u} + \alpha\Delta\mathbf{u} \geq \mathbf{0}$ és $\mathbf{z} + \alpha\Delta\mathbf{z} \geq \mathbf{0}$, ami

$$0 < \alpha < \bar{\alpha} = \min\left\{\min_{i\in\mathcal{J}}\left\{-\frac{u_i}{\Delta u_i} : \Delta u_i < 0\right\}, \min_{i\in\mathcal{J}}\left\{-\frac{z_i}{\Delta z_i} : \Delta z_i < 0\right\}, 1\right\} \tag{25.10}$$

[9] Később az elméleti elemzésekből szembetűnő lesz, hogy elméletileg minél nagyobb a τ értéke, annál rosszabb lesz az iteráció szám. Ezt a gyakorlat nem igazolta vissza, de annak érdekében, hogy elméletileg elemezhető legyen az algoritmus, és hogy a polinomiális lépésszámot az algoritmus egy változatára általánosan bizonyítani tudjuk, a legrosszabb eset elemzést kell használnunk. A legrosszabb esettől a gyakorlati problémák sokszor eltérnek, de nem mindig.

[10] Ez az elképzelés is jelentősen eltér az elméletileg bizonyítható helyzetektől, amikor is közvetlenül τ függvényében adjuk meg a korlátokat a megengedett lépéshosszra, nem törődve a lokálisan legjobb, meghatározható érték kiszámításával. Cserébe elméletileg is elemezhető algoritmus változatot kapunk, amelynek a lépésszáma polinomiális.

esetén biztosan teljesülni fog, ahol $\mathcal{I} = \{1, 2, \ldots, n\}$. (Ezen állítás helyességét a 25.2-6. gyakorlat eredménye garantálja.) Másfelől fontos, hogy a centralitási mértékünk a megadott érték alatt maradjon az új megoldások esetén is. Mivel az

$$\mathbf{u}^+ = \mathbf{u} + \alpha\Delta\mathbf{u} = \mathbf{d}(\mathbf{d}^{-1}\mathbf{u}) + \alpha\mathbf{d}(\mathbf{d}^{-1}\Delta\mathbf{u}) = \mathbf{d}\sqrt{\mu}\,\mathbf{v} + \alpha\mathbf{d}\sqrt{\mu}\,\mathbf{p}_u = \sqrt{\mu}\,\mathbf{d}(\mathbf{v} + \alpha\mathbf{p}_u)\,,$$

és

$$z(\mathbf{u}^+) = \mathbf{z}^+ = \mathbf{z} + \alpha\Delta\mathbf{z} = \sqrt{\mu}\,\mathbf{d}^{-1}(\mathbf{v} + \alpha\mathbf{p}_z),$$

ezért

$$\begin{aligned}(\mathbf{u}^+)z(\mathbf{u}^+) &= (\mathbf{u} + \alpha\Delta\mathbf{u})\,(\mathbf{z} + \alpha\Delta\mathbf{z}) = \mu\,(\mathbf{v} + \alpha\mathbf{p}_u)(\mathbf{v} + \alpha\mathbf{p}_z)\\ &= \mu\,(\mathbf{v}^2 + \alpha(\mathbf{p}_u + \mathbf{p}_z)\mathbf{v} + \alpha^2\mathbf{p}_u\mathbf{p}_z)\,.\end{aligned}$$

A

$$\mathbf{p} = \mathbf{p}_u + \mathbf{p}_z = -\frac{\mathbf{v}^3}{\|\mathbf{v}^2\|}$$

összefüggést figyelembe véve

$$\mathbf{u}^+ z(\mathbf{u}^+) = \mu\left(\mathbf{v}^2 - \alpha\frac{\mathbf{v}^4}{\|\mathbf{v}^2\|} + \alpha^2\mathbf{p}_u\mathbf{p}_z\right) \tag{25.11}$$

adódik. A $\mu > 0$ és az $\bar{\alpha}$ korlát miatt az $\mathbf{u}^+ z(\mathbf{u}^+) > 0$ teljesül, de mi még azt is szeretnénk, hogy

$$\tau_1\,\mathbf{e} \le \mathbf{v}^2 - \alpha\frac{\mathbf{v}^4}{\|\mathbf{v}^2\|} + \alpha^2\mathbf{p}_u\mathbf{p}_z \le \tau_2\,\mathbf{e}$$

is igaz legyen, azaz bármely i index esetén

$$\tau_1 \le u_i^2 - \alpha\frac{u_i^4}{\|\mathbf{v}^2\|} + \alpha^2(\mathbf{p}_u\mathbf{p}_z)_i \le \tau_2\,. \tag{25.12}$$

A bal oldali, az α ismeretlenre nézve kvadratikus, egyenlőtlenségből az alábbi korlátokat kapjuk:

$$\alpha_1 = \min_{i\in\mathcal{J}}\left\{\frac{\frac{u_i^4}{\|\mathbf{v}^2\|} + \sqrt{\vartheta_1}}{2\,(\mathbf{p}_u\mathbf{p}_z)_i} :\ \vartheta_1 \ge 0,\ (\mathbf{p}_u\mathbf{p}_z)_i > 0\right\} \quad \text{és}$$

$$\alpha_2 = \min_{i\in\mathcal{J}}\left\{\frac{\frac{u_i^4}{\|\mathbf{v}^2\|} - \sqrt{\vartheta_1}}{2\,(\mathbf{p}_u\mathbf{p}_z)_i} :\ \vartheta_1 \ge 0,\ (\mathbf{p}_u\mathbf{p}_z)_i < 0\right\}\,,$$

ahol $\vartheta_1 = u_i^8/\left\|\mathbf{v}^2\right\|^2 - 4\,(\mathbf{p}_u\mathbf{p}_z)_i\left(u_i^2 - \tau_1\right)$. Hasonlóan a jobb oldali egyenlőtlenségből a következő korlátok adódnak α értékére

$$\alpha_3 = \min_{i\in\mathcal{J}}\left\{\frac{-\frac{u_i^4}{\|\mathbf{v}^2\|} - \sqrt{\vartheta_2}}{-2\,(\mathbf{p}_u\mathbf{p}_z)_i} :\ \vartheta_2 \ge 0,\ (\mathbf{p}_u\mathbf{p}_z)_i > 0\right\} \quad \text{és}$$

$$\alpha_4 = \min_{i\in\mathcal{J}}\left\{\frac{-\frac{u_i^4}{\|\mathbf{v}^2\|} + \sqrt{\vartheta_2}}{-2\,(\mathbf{p}_u\mathbf{p}_z)_i} :\ \vartheta_2 \ge 0,\ (\mathbf{p}_u\mathbf{p}_z)_i < 0\right\}\,,$$

ahol $\vartheta_2 = u_i^8/\left\|\mathbf{v}^2\right\|^2 - 4\,(\mathbf{p}_u\mathbf{p}_z)_i\left(u_i^2 - \tau_2\right)$.

25.2.2. Dikin-algoritmus: elméleti elemzés

Az iterációk során csak egy tompított[11] Dikin-lépést teszünk meg. A lépéshossz megfelelő megválasztásával biztosítjuk az új pontunk pozitivitását, így a megengedettségét, valamint a pontunkat nem engedjük ki a centrális út τ környezetéből. Ezen elvárásoknak megfelelő α lépéshosszra ad elégséges feltételt a következő lemma, amelynek bizonyítását az Olvasóra bízzuk (lásd 25.2-9. gyakorlat).

25.24. lemma. *Legyen $\tau \in \mathbb{R}$, $\tau > 1$. Tegyük fel, hogy $\mathbf{u} > \mathbf{0}$, $\mathbf{z} = z(\mathbf{u}) > \mathbf{0}$, $\mathbf{u} \in \mathcal{F}$ és $\partial_c(\mathbf{u}) \le \tau$. Ekkor bármely α lépéshossz, amely kielégíti az*

$$\alpha \le \frac{\|\mathbf{v}^2\|}{2\tau_2} \quad \text{és} \quad \alpha < \frac{4\tau_1}{\|\mathbf{v}^2\|}$$

egyenlőtlenségeket, megengedett lépéshossz, és az $\mathbf{u}^+ = \mathbf{u} + \alpha\Delta\mathbf{u}$ vektorra $\partial_c(\mathbf{u}^+) \le \tau$.

Tekintettel arra, hogy az előző lemma számunkra egy olyan lépéshosszt biztosít, amelyik esetén az új megoldásunk megengedett lesz, és rá a megfelelő centralitási elvárás is teljesül, rátérhetünk annak a vizsgálatára, hogy ez mekkora dualitásrés csökkenést fog eredményezni.

25.25. lemma. *Ha az α lépéshossz megengedett, akkor*

$$(\mathbf{u}^+)^T\mathbf{z}^+ \le \left(1 - \frac{\alpha}{\sqrt{n}}\right)\mathbf{u}^T\mathbf{z} \,.$$

Bizonyítás. Induljunk ki a (25.11) egyenlőségből. Mivel az α megengedett lépéshossz és $\mu > 0$, ezért

$$\mathbf{v}^2 - \alpha\frac{\mathbf{v}^4}{\|\mathbf{v}^2\|} + \alpha^2\mathbf{p}_u\mathbf{p}_z > \mathbf{0} \,.$$

Az M mátrix ferdén szimmetrikus, ezért $(\Delta\mathbf{u})^T\Delta\mathbf{z} = 0$, azaz

$$\mathbf{p}_u^T\mathbf{p}_z = \mathbf{e}^T(\mathbf{d}^{-1}\Delta\mathbf{u}\,\mathbf{d}\,\Delta\mathbf{z})\frac{1}{\mu} = \frac{1}{\mu}(\Delta\mathbf{u})^T\Delta\mathbf{z} = 0 \,. \tag{25.13}$$

Számítsuk ki az új dualitásrés és a μ paraméter hányadosát:

$$\frac{(\mathbf{u}^+)^T\mathbf{z}^+}{\mu} = \mathbf{e}^T\mathbf{v}^2 - \alpha\frac{\mathbf{e}^T\mathbf{v}^4}{\|\mathbf{v}^2\|} + \alpha^2\mathbf{p}_u^T\mathbf{p}_z = \|\mathbf{v}\|^2 - \alpha\|\mathbf{v}^2\| \,,$$

ahol figyelembe vettük a $\mathbf{p}_u$ és $\mathbf{p}_z$ vektorok ortogonalitását és az $\mathbf{e}^T\mathbf{v}^4 = (\mathbf{v}^2)^T\mathbf{v}^2 = \|\mathbf{v}^2\|^2$ összefüggést. A Cauchy–Schwarz egyenlőtlenség alapján

$$\|\mathbf{v}^2\| = \frac{1}{\sqrt{n}}\|\mathbf{e}\|\,\|\mathbf{v}^2\| \ge \frac{\mathbf{e}^T\mathbf{v}^2}{\sqrt{n}} = \frac{\|\mathbf{v}\|^2}{\sqrt{n}} \,,$$

[11] Legyen az új pontunk $(\mathbf{u} + \alpha\Delta\mathbf{u}, \mathbf{z} + \alpha\Delta\mathbf{z})$, ha $0 < \alpha < 1$, akkor α paraméterrel tompított lépésről, míg $\alpha = 1$ esetén teljes lépésről beszélünk.

behelyettesítve

$$(\mathbf{u}^+)^T\mathbf{z}^+ \le \mu\left(1-\frac{\alpha}{\sqrt{n}}\right)\|\mathbf{v}\|^2 = \left(1-\frac{\alpha}{\sqrt{n}}\right)\mathbf{u}^T\mathbf{z}$$

adódik. ■

A 25.24. lemma szerint az $\alpha = 1/(\tau\sqrt{n})$ választás[12] megfelelő, azaz ebben az esetben az algoritmus jól definiált. Az alábbi tételben megmutatjuk, hogy ezen érték mellett az algoritmus polinomiális.

25.26. tétel. *Legyen* $\tau = \max(2, \partial_c(\mathbf{u}^0))$ *és* $\alpha = 1/(\tau\sqrt{n})$. *Ha* $n \ge 2$, *akkor a Dikin-féle affin skálázású algoritmus a ferdén szimmetrikus lineáris programozási feladatot legfeljebb*

$$\left\lceil \tau n \lg \frac{\mathbf{q}^T\mathbf{u}^0}{\varepsilon} \right\rceil$$

lépésben oldja meg. A megoldás $\mathbf{u} \in \mathcal{F}$, *amelyre* $\partial_c(\mathbf{u}) \le \tau$ *és* $\mathbf{q}^T\mathbf{u} \le \varepsilon$.

Bizonyítás. Az $\mathbf{u}^0 \in \mathcal{F}^0$ induló megoldásra $\partial_c(\mathbf{u}^0) \le \tau$ összefüggés teljesült. A 25.24. lemma alapján a lépéshossz megválasztható úgy, hogy minden egyes lépésben az új pont pozitív, megengedett megoldás legyen és a centralitási mértéke τ értékénél kisebb maradjon. Azt kell megmutatni, hogy a tétel feltételeiben megadott α érték kielégíti a 25.24. lemmában adott korlátokat. Mivel $n \ge 2$, ezért

$$\alpha = \frac{1}{\tau\sqrt{n}} = \frac{\tau_1}{\tau_2\sqrt{n}} \le \frac{\tau_1\sqrt{n}}{2\tau_2} = \frac{\|\tau_1\mathbf{e}\|}{2\tau_2} \le \frac{\|\mathbf{v}^2\|}{2\tau_2}\,,$$

ahol felhasználtuk azt is, hogy $\mathbf{0} \le \tau_1\mathbf{e} \le \mathbf{v}^2$. A $\|\mathbf{v}^2\| \le \tau_2\sqrt{n}$ miatt

$$\frac{4\tau_1}{\|\mathbf{v}^2\|} \ge \frac{4\tau_1}{\tau_2\sqrt{n}} = \frac{4}{\tau\sqrt{n}} > \alpha$$

adódik, azaz a tétel feltételeiben megadott lépéshossz megengedett, mert teljesíti a 25.24. lemmában adott korlátokat.

Az induló megoldásnál a célfüggvény értéke $\mathbf{q}^T\mathbf{u}^0$ volt. (Könnyen megmutatható, hogy $\mathbf{q}^T\mathbf{u}^0 = (\mathbf{u}^0)^T z(\mathbf{u}^0)$, ami pedig a dualitásrés.) A 25.25. lemma alapján a dualitásrés minden lépésben $(1-1/(n\tau))$ szorzóval csökken. Így k lépés után a célfüggvény értéke ε alá csökken, ha

$$\left(1-\frac{1}{n\tau}\right)^k \mathbf{q}^T\mathbf{u}^0 \le \varepsilon\,.$$

Szeretnénk meghatározni a k értékét. Vegyük az előző kifejezés logaritmusát. Ekkor

$$k \lg\left(1-\frac{1}{n\tau}\right) + \lg(\mathbf{q}^T\mathbf{u}^0) \le \lg\varepsilon\,.$$

[12] Mivel az α paraméter függ az n számtól, azaz a feladat dimenziójától, rövid lépéses algoritmusról beszélünk.

Átrendezés után

$$\lg \frac{\mathbf{q}^T\mathbf{u}^0}{\varepsilon} \leq -k \lg\left(1 - \frac{1}{n\tau}\right)$$

adódik. Mivel $-\lg(1-t) \geq t$, ezért a következő egyenlőtlenség a fentinél erősebb feltétel

$$\frac{k}{n\tau} \geq \lg \frac{\mathbf{q}^T\mathbf{u}^0}{\varepsilon} \ .$$

Ebből pedig már következik a tétel állítása. ■

25.2.3. Az optimális partíció meghatározása

Ahhoz, hogy a változók koordinátáira a centrális út mentén egy korlátot tudjunk adni, szükségünk lesz egy, az optimális megoldások halmazát jellemző mérőszámra. Az előzőek alapján az $\mathbf{u} \in \mathcal{F}$ vektorra pontosan akkor igaz, hogy $\mathbf{u} \in \mathcal{F}^*$, ha $\mathbf{u}_\mathcal{N} = \mathbf{0}$ és $\mathbf{z}_\mathcal{B} = \mathbf{0}$ teljesül[13], ahol $(\mathcal{B}, \mathcal{N})$ a feladat optimális partícióját jelöli (lásd a 25.18. definíciót). Tetszőleges $\mathbf{u} \in \mathcal{F}^*$ optimális megoldásra fennáll az

$$\mathbf{u}\, z(\mathbf{u}) = \mathbf{0} \quad \text{és} \quad \mathbf{u} + z(\mathbf{u}) \geq \mathbf{0}$$

összefüggés. Első lépésként az optimális megoldások nemnulla koordinátáinak nagyságára keresünk egy jellemző mennyiséget. Definiáljuk az (SP) feladat kondíciószámát a alábbi módon.

25.27. definíció. *Legyen*

$$\sigma^u_{SP} = \begin{cases} \min\limits_{i\in\mathcal{B}} \max\limits_{(\mathbf{u},\mathbf{z})\in\mathcal{F}^*} \{u_i\}, & \text{ha } \mathcal{B} \neq \emptyset\ , \\ \infty & \text{különben}\ , \end{cases}$$

és

$$\sigma^z_{SP} = \begin{cases} \min\limits_{i\in\mathcal{N}} \max\limits_{(\mathbf{u},\mathbf{z})\in\mathcal{F}^*} \{z(u)_i\}, & \text{ha } \mathcal{N} \neq \emptyset\ , \\ \infty & \text{különben}\ . \end{cases}$$

Az (SP) feladat ***kondíciószáma*** $\sigma_{SP} = \min\{\sigma^u_{SP}, \sigma^z_{SP}\}$.

Mivel a $\mathcal{B}$ és $\mathcal{N}$ halmazok egyszerre nem lehetnek üresek, az $\mathcal{F}^*$ halmaz pedig korlátos, σ^u_{SP} és σ^z_{SP} számok közül legalább az egyik véges. Továbbá $\sigma^u_{SP}, \sigma^z_{SP} > 0$, azaz a kondíciószám egy véges pozitív szám. Valamint mivel erősen komplementáris megoldás létezik és minden $(\mathbf{u}, \mathbf{z}) \in \mathcal{F}^*$ komplementáris megoldás, következésképpen

$$\sigma_{SP} = \min_i \max_{(\mathbf{u},\mathbf{z})\in\mathcal{F}^*} \{u_i + z(u)_i\}\ .$$

Első célunk a σ_{SP} értékére egy alsó korlát meghatározása.

25.28. tétel. *Legyen adott az (SP) feladat, melyre teljesül a belső pont feltétel. Ha az M mátrix és a $\mathbf{q}$ vektor egészértékű, akkor az (SP) feladat kondíciószámára a következő alsó*

[13] Jelölje $\mathbf{u}_\mathcal{N}$ az $\mathbf{u}$ vektor $\mathbf{u}_\mathcal{N}$ halmazbeli indexekből álló részvektorát. Hasonlóan definiáljuk a $\mathbf{z}_\mathcal{B}$ részvektort.

becslés teljesül:

$$\sigma_{SP} \geq \frac{1}{\prod_{j=1}^{n} \|\mathbf{m}_j\|} ,$$

ahol $\mathbf{m}_j$ *az* M *mátrix* j*-edik oszlopa.*

Bizonyítás. Tegyük fel, hogy a $(\mathcal{B}, \mathcal{N})$ partíciót ismerjük. Ekkor

$$\begin{pmatrix} \mathbf{z}_\mathcal{B} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & M_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & M_{\mathcal{N}\mathcal{N}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{u}_\mathcal{N} \end{pmatrix} + \begin{pmatrix} \mathbf{q}_\mathcal{B} \\ \mathbf{q}_\mathcal{N} \end{pmatrix} .$$

Legyen $(\mathbf{u}, \mathbf{z}) \in \mathcal{F}$ szigorúan komplementáris megoldás. A komplementaritás miatt az $(\mathbf{u}, \mathbf{z})$ vektor optimális (25.2. következmény), azaz $\mathbf{q}^T\mathbf{u} = 0$. Továbbá felhasználva a $(\mathcal{B}, \mathcal{N})$ partíció definícióját ($\mathbf{u}_\mathcal{N} = \mathbf{0}$) azt kapjuk, hogy $\mathbf{q}_\mathcal{B}^T\mathbf{u}_\mathcal{B} = 0$. Figyelembe véve a szigorú komplementaritást $\mathbf{u}_\mathcal{B} > \mathbf{0}$, így szükségszerűen $\mathbf{q}_\mathcal{B} = \mathbf{0}$. Ezeket felhasználva a fenti rendszerünk a következő alakú:

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & M_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & M_{\mathcal{N}\mathcal{N}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{q}_\mathcal{N} \end{pmatrix} ,$$

így

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}}\mathbf{u}_\mathcal{B} \\ M_{\mathcal{N}\mathcal{B}}\mathbf{u}_\mathcal{B} + \mathbf{q}_\mathcal{N} \end{pmatrix}.$$

Tehát

$$\begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & 0_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & -I_{\mathcal{N}\mathcal{N}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} \mathbf{0}_\mathcal{B} \\ -\mathbf{q}_\mathcal{N} \end{pmatrix}, \qquad \mathbf{u}_\mathcal{B} \geq \mathbf{0},\ \mathbf{u}_\mathcal{N} = \mathbf{0},\ \mathbf{z}_\mathcal{B} = \mathbf{0},\ \mathbf{z}_\mathcal{N} \geq \mathbf{0} .$$

Az utóbbi rendszer éppen az $\mathcal{F}^*$ leírása, ami egy nem üres, kompakt halmaz és szigorúan komplementáris megoldást is tartalmaz, így az

$$R = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & 0_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & -I_{\mathcal{N}\mathcal{N}} \end{pmatrix} , \quad \mathbf{y} = \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} , \quad \mathbf{r} = \begin{pmatrix} \mathbf{0}_\mathcal{B} \\ -\mathbf{q}_\mathcal{N} \end{pmatrix}$$

választással a 25.2-11. gyakorlat eredményét alkalmazhatjuk, és így

$$\max_{\mathbf{u}} u_i \geq \frac{1}{\prod_{j\in\mathcal{B}} \|\mathbf{m}_j\|}, \quad \forall i \in \mathcal{B} \qquad \text{és} \qquad \max_{\mathbf{z}} z_i \geq \frac{1}{\prod_{j\in\mathcal{B}} \|\mathbf{m}_j\|}, \quad \forall i \in \mathcal{N}$$

adódik. Ezekből pedig a tétel állítását,

$$\sigma_{SP} \geq \frac{1}{\prod_{j\in\mathcal{B}} \|\mathbf{m}_j\|} \geq \frac{1}{\prod_{j=1}^{n} \|\mathbf{m}_j\|}$$

nyerjük. ■

Természetesen az előző tétel akkor is igaz marad, ha az M mátrix és a $\mathbf{q}$ vektor adatai racionálisak. Ebben az esetben a nevezők legkisebb közös többszörösével kell megszoroznunk a mátrix és a vektor elemeit. Ezután az előző tétel alkalmazásával a kondíciószámra egy alsó korlátot tudunk kiszámolni.

A változók mérete a centrális út mentén

A $\sigma = \sigma_{SP}$ kondíciószám segítségével alsó és felső korlátokat adhatunk a változókra a centrális út mentén. Legyen $(\mathcal{B}, \mathcal{N})$ az (SP) feladat optimális partíciója.

25.29. lemma. *Minden* $(\mathbf{u}(\mu), \mathbf{z}(\mu)) \in \mathcal{C}$, $\mu > 0$ *esetén az alábbi egyenlőtlenségek igazak:*

$$u_i(\mu) \geq \frac{\sigma}{n} \quad i \in \mathcal{B}\,, \qquad u_i(\mu) \leq \frac{n\mu}{\sigma} \quad i \in \mathcal{N}\,,$$
$$z_i(\mu) \leq \frac{n\mu}{\sigma} \quad i \in \mathcal{B}\,, \qquad z_i(\mu) \geq \frac{\sigma}{n} \quad i \in \mathcal{N}\,.$$

A lemma, amelynek a bizonyítását az Olvasóra bízzuk (lásd 25.2-12. gyakorlat), alapján természetesen adódnak a következő elnevezések:

25.30. definíció. *Legyen* $(\mathbf{u}(\mu), \mathbf{z}(\mu)) \in \mathcal{C}$, $\mu > 0$. *Ekkor az* $u_i(\mu)$ *koordinátát* ***nagynak*** *mondjuk, ha* $i \in \mathcal{B}$, *illetve a* $z_i(\mu)$ *koordinátát* ***nagynak*** *mondjuk, ha* $i \in \mathcal{N}$. *Ellenkező esetben a koordinátákat* ***kicsinek*** *nevezzük.*

Az elnevezés is mutatja, hogy az $\mathbf{u}$ vektor nagy koordinátáinak indexei alkotják a $\mathcal{B}$ halmazt, míg a $\mathbf{z}$ nagy koordinátáinak indexei az $\mathcal{N}$ halmazt. A lemma alapján ha a μ paraméter elég kicsi, akkor egyértelműen eldönthető, hogy az $\mathbf{u}(\mu)$ és a $\mathbf{z}(\mu)$ vektorok mely koordinátái nagyok és melyek kicsik, azaz a $(\mathcal{B}, \mathcal{N})$ partíció egyértelműen meghatározható.

25.31. következmény. *Ha a* $\mu < \sigma^2/n^2$ *centrális út paraméterhez ismert az* $(\mathbf{u}(\mu), \mathbf{z}(\mu)) \in \mathcal{C}$ *megoldás, akkor meghatározhatjuk a* $(\mathcal{B}, \mathcal{N})$ *optimális partíciót.*

Bizonyítás. A megadott μ érték mellett $n\mu/\sigma < \sigma/n$ teljesül, így az előző lemma alapján a

$$\mathcal{B} = \left\{ i \;:\; u_i(\mu) \geq \frac{\sigma}{n} \right\}\,, \qquad \mathcal{N} = \left\{ i \;:\; z_i(\mu) \geq \frac{\sigma}{n} \right\}$$

szétosztás egyértelmű és megfelelő. ■

Tekintettel arra, hogy a belsőpontos módszerek, az iterációk során, a legritkábban állítanak elő a centrális úton lévő pontot, ezért az előző eredmények megfelelőit a centrális út egy környezetében is bizonyítanunk kell.

A változók mérete a centrális út egy környezetében

Legyen adott $\mathbf{u}^0 \in \mathcal{F}^0$, $\mathbf{z}^0 > 0$, melyek a $\mu^0 = 1$ paraméterhez tartoznak, azaz $(\mathbf{u}^0)^T\mathbf{z}^0 = \mathbf{q}^T\mathbf{u}^0 = n\mu^0 = n$. Legyen $\tau = 2$, ekkor a 25.26. tétel alapján a Dikin-lépéses algoritmus $\lceil 2n \lg \frac{n}{\varepsilon} \rceil$ iterációban előállít egy $\mathbf{u} \in \mathcal{F}^0$ pontot, melyre $\partial_c(\mathbf{u}) \leq 2$ és $\mathbf{q}^T\mathbf{u} \leq \varepsilon$.

Fontos lenne a centrális út környezetében tudni a változók nagyságrendjét. Erre szolgál a következő lemma, amelynek bizonyítását az Olvasóra bízzuk (lásd 25.2-13. gyakorlat).

25.32. lemma. *Legyen* $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} > 0$, *melyekre* $\partial_c(\mathbf{u}) \leq \tau$. *Ekkor*

$$u_i \geq \frac{\sigma}{\tau n} \quad i \in \mathcal{B}\,, \qquad u_i \leq \frac{\mathbf{u}^T\mathbf{z}}{\sigma} \quad i \in \mathcal{N}\,,$$
$$z_i \leq \frac{\mathbf{u}^T\mathbf{z}}{\sigma} \quad i \in \mathcal{B}\,, \qquad z_i \geq \frac{\sigma}{\tau n} \quad i \in \mathcal{N}\,,$$

ahol σ *a feladat kondíciószáma.*

Megfelelő feltétel mellett a 25.32. lemma segítségével az $\mathbf{u}$, illetve az $\mathbf{z}$ vektor koordinátáit nagyság szerint szétválogatva meghatározhatjuk a $(\mathcal{B},\mathcal{N})$ partíciót.

25.33. lemma. *Legyen $\mathbf{u}\in\mathcal{F}^0$, $\mathbf{z}>0$, melyekre $\partial_c(\mathbf{u})\le\tau$. Ha $\mathbf{u}^T\mathbf{z}\le\sigma^2/(\tau n)$, akkor az (SP) feladat $(\mathcal{B},\mathcal{N})$ partíciója a következő módon kapható meg:*

$$\mathcal{B}=\{i\ :\ u_i>z_i\}\qquad\text{és}\qquad\mathcal{N}=\{i\ :\ u_i<z_i\}\ .$$

Bizonyítás. A kicsi és nagy változókat szét tudjuk választani, ha $\mathbf{u}^T\mathbf{z}/\sigma<\sigma/(\tau n)$, azaz $\mathbf{u}^T\mathbf{z}<\sigma^2/(\tau n)$. Tehát az előző lemma szerint a fent megadott $\mathcal{B}$ és $\mathcal{N}$ halmazok jól definiáltak, és az indexhalmaz optimális partícióját adják. ■

A 25.33. lemma feltételeit kielégítő $(\mathbf{u},\mathbf{z})$ pontpár a 25.26. tétel szerint $\lceil 2n\lg\frac{2n^2}{\sigma^2}\rceil$ lépésben a Dikin-algoritmussal előállítható, ennek tükrében az alábbi következmény könnyen adódik.

25.34. következmény. *Legyen $\mathbf{u}^0\in\mathcal{F}^0$, $\mathbf{z}^0>0$, melyekre $\partial_c(\mathbf{u}^0)\le\tau$ és $(\mathbf{u}^0)^T\mathbf{z}^0=n$, $\tau=2$. Ekkor a Dikin-lépéses algoritmus $\lceil 2n\lg\frac{2n^2}{\sigma^2}\rceil$ lépésben egy olyan $\mathbf{u}\in\mathcal{F}^0$ megoldást ad, amely esetén a $(\mathcal{B},\mathcal{N})$ partíció meghatározható.*

Mint már az első részben említettük az $\mathbf{u}=\mathbf{0}$ mindig optimális megoldása az (*SP*) feladatnak. A következőkben megmutatjuk, hogy ez akkor és csak akkor az egyetlen optimális vektor, ha a $\mathcal{B}$ indexhalmaz üres. (Ezzel a speciális, triviális esettel a továbbiakban nem foglalkozunk.)

25.35. tétel. *Legyen $\mathcal{F}^0\neq\emptyset$. Ekkor $\mathbf{u}=\mathbf{0}$ az (SP) feladat egyetlen optimális megoldása akkor és csak akkor, ha $\mathbf{q}>\mathbf{0}$. Ez az eset pontosan akkor fordul elő, ha $\mathcal{B}=\emptyset$.*

Bizonyítás. Ha $\mathcal{B}=\emptyset$, akkor $\mathbf{u}=\mathbf{0}$ optimális, sőt szigorúan komplementáris is $\mathcal{F}^0\neq\emptyset$ mellett, azaz $z(\mathbf{u})=\mathbf{q}>\mathbf{0}$ kell, hogy teljesüljön.
Másfelől $z(\mathbf{0})=\mathbf{q}$ és $z(\mathbf{0})>\mathbf{0}$ miatt $\mathbf{q}>\mathbf{0}$, így $\mathbf{u}=\mathbf{0}$ egyértelmű optimum. ■

Vizsgáljuk meg az $\mathbf{u}\in\mathcal{F}^0$, $z(\mathbf{u})=\mathbf{z}>\mathbf{0}$ ε-optimális megoldást. Ha az ε kellően kicsi, akkor a $(\mathcal{B},\mathcal{N})$ partíció ismert, így a rendszerünk a következő alakú:

$$\begin{pmatrix}\mathbf{z}_\mathcal{B}\\ \mathbf{z}_\mathcal{N}\end{pmatrix}=\begin{pmatrix}M_{\mathcal{B}\mathcal{B}} & M_{\mathcal{B}\mathcal{N}}\\ M_{\mathcal{N}\mathcal{B}} & M_{\mathcal{N}\mathcal{N}}\end{pmatrix}\begin{pmatrix}\mathbf{u}_\mathcal{B}\\ \mathbf{u}_\mathcal{N}\end{pmatrix}+\begin{pmatrix}\mathbf{q}_\mathcal{B}\\ \mathbf{q}_\mathcal{N}\end{pmatrix}\ .$$

Az előző tétel szerint a $\mathcal{B}=\emptyset$ eset triviális, ezért tegyük fel, hogy $\mathcal{B}\neq\emptyset$, ekkor az optimalitás miatt $\mathbf{q}_\mathcal{B}=\mathbf{0}$.

$$\mathbf{z}_\mathcal{B}=M_{\mathcal{B}\mathcal{B}}\,\mathbf{u}_\mathcal{B}+M_{\mathcal{B}\mathcal{N}}\,\mathbf{u}_\mathcal{N}+\mathbf{q}_\mathcal{B}\ ,\quad\text{tehát}$$

$$M_{\mathcal{B}\mathcal{B}}\,\mathbf{u}_\mathcal{B}=\mathbf{z}_\mathcal{B}-M_{\mathcal{B}\mathcal{N}}\,\mathbf{u}_\mathcal{N}\ .\tag{25.14}$$

Ha $\tilde{\mathbf{u}}\in\mathcal{F}^*$, $\tilde{\mathbf{z}}=z(\tilde{\mathbf{u}})$, akkor $\tilde{\mathbf{u}}_\mathcal{N}=\mathbf{0}$, $\tilde{\mathbf{z}}_\mathcal{B}=\mathbf{0}$, és a szigorú komplementaritás miatt $\tilde{\mathbf{u}}_\mathcal{B}>\mathbf{0}$, valamint a (25.14) egyenlőség miatt $M_{\mathcal{B}\mathcal{B}}\,\tilde{\mathbf{u}}_\mathcal{B}=\mathbf{0}$, azaz az $M_{\mathcal{B}\mathcal{B}}$ mátrix szinguláris. Ennek következtében az

$$M_{\mathcal{B}\mathcal{B}}\,\xi=\mathbf{z}_\mathcal{B}-M_{\mathcal{B}\mathcal{N}}\,\mathbf{u}_\mathcal{N}\tag{25.15}$$

egyenlet megoldása nem egyértelmű, egy lehetséges megoldása a $\xi = \mathbf{u}_{\mathcal{B}}$. Legyen ξ tetszőleges megoldása a (25.15) egyenletnek, ekkor definiálhatjuk az $\overline{\mathbf{u}}_{\mathcal{B}} = \mathbf{u}_{\mathcal{B}} - \xi$, $\overline{\mathbf{u}}_{\mathcal{N}} = \mathbf{0}$ megoldást és a $\overline{\mathbf{z}} = z(\overline{\mathbf{u}})$ eltérésváltozót, ahol $\overline{\mathbf{z}}_{\mathcal{B}} = M_{\mathcal{B}\mathcal{B}}\,\overline{\mathbf{u}}_{\mathcal{B}} + M_{\mathcal{B}\mathcal{N}}\,\overline{\mathbf{u}}_{\mathcal{N}} + \mathbf{q}_{\mathcal{B}} = M_{\mathcal{B}\mathcal{B}}\,(\mathbf{u}_{\mathcal{B}} - \xi) = \mathbf{0}$. Tehát az $(\overline{\mathbf{u}}, \overline{\mathbf{z}})$ komplementáris megoldás, de $\mathbf{u}$ és $\mathbf{u}$ nem feltétlenül pozitív vektorok. Az $(\overline{\mathbf{u}}, \overline{\mathbf{z}})$ pozitivitásához szükséges:

$$\overline{\mathbf{u}}_{\mathcal{B}} = \mathbf{u}_{\mathcal{B}} - \xi > \mathbf{0}\,, \quad \text{illetve}$$

$$\begin{aligned}\overline{\mathbf{z}}_{\mathcal{N}} &= M_{\mathcal{N}\mathcal{B}}\,\overline{\mathbf{u}}_{\mathcal{B}} + M_{\mathcal{N}\mathcal{N}}\,\overline{\mathbf{u}}_{\mathcal{N}} + \mathbf{q}_{\mathcal{N}} = M_{\mathcal{N}\mathcal{B}}\,\mathbf{u}_{\mathcal{B}} - M_{\mathcal{N}\mathcal{B}}\,\xi + \mathbf{q}_{\mathcal{N}} \\ &= \mathbf{z}_{\mathcal{N}} - M_{\mathcal{N}\mathcal{N}}\,\mathbf{u}_{\mathcal{N}} - M_{\mathcal{N}\mathcal{B}}\,\xi > \mathbf{0}\,.\end{aligned}$$

Tehát ε megfelelő értéke mellett elő tudjuk állítani az $(\overline{\mathbf{u}}, \overline{\mathbf{z}})$ vektort, ami az előző feltételek teljesülése mellett szigorúan komplementáris megoldás lesz. A következő részben ezen megoldás előállításával foglalkozunk.

Szigorúan komplementáris megoldás előállítása
Vezessük be a következő jelöléseket:

$$w = \|M\|_\infty\,, \qquad \mathcal{B}^* = \{i \in \mathcal{B} \;:\; M_{\mathcal{B}i} \text{ oszlopvektor nem azonosan nulla}\}.$$

Definiáljuk a $\pi_{\mathcal{B}} \in \mathbb{N}$ számot a következő módon

$$\pi_{\mathcal{B}} = \begin{cases} 1, & \text{ha } \mathcal{B}^* = \emptyset\,, \\ \prod\limits_{j \in \mathcal{B}^*} \|M_{\mathcal{B}j}\| & \text{különben}\,. \end{cases}$$

Célunk a belsőpontos algoritmussal meghatározott, kellően pontos közelítő megoldásból az (*SP*) feladat egy szigorúan komplementáris megoldását előállítani. A következő algoritmus bemeneti adataként adjunk meg egy $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} > 0$ pontpárt, melyre $\partial_c(\mathbf{u}) \le \tau = 2$ és $\mathbf{u}^T\mathbf{z} \le \sigma^2 / \left(6nw^2\sqrt{|\mathcal{B}|\pi_{\mathcal{B}}}\right)$ teljesül.

SZIGORÚAN-KOMPLEMENTÁRIS-MEGOLDÁS

1 határozzuk meg a $(\mathcal{B}, \mathcal{N})$ partíciót az $(\mathbf{u}, \mathbf{z})$ pontpárból
2 oldjuk meg Gauss-eliminációval az
  $M_{\mathcal{B}\mathcal{B}}\,\xi = \mathbf{z}_{\mathcal{B}} - M_{\mathcal{B}\mathcal{N}}\,\mathbf{u}_{\mathcal{N}}$ egyenletrendszert ▷ Kerekítési eljárás.
3 $\bar{\mathbf{u}}_{\mathcal{B}} \leftarrow \mathbf{u}_{\mathcal{B}} - \xi$
4 $\bar{\mathbf{u}}_{\mathcal{N}} \leftarrow \mathbf{0}$
5 $\bar{\mathbf{u}} \leftarrow (\bar{\mathbf{u}}_{\mathcal{B}}, \bar{\mathbf{u}}_{\mathcal{N}})$
6 $\bar{\mathbf{z}} \leftarrow z(\bar{\mathbf{u}})$

A fenti eljárással kapott $(\bar{\mathbf{u}}, \bar{\mathbf{z}})$ pontpár szigorúan komplementáris megoldása a feladatnak, mint ahogy ezt a következő lemmában ki is mondjuk.

25.36. lemma. *Legyen adott egy egész együtthatós (SP) feladat. Legyen* $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} > 0$, *melyekre* $\partial_c(\mathbf{u}) \le \tau = 2$. *Ha* $\mathbf{u}^T\mathbf{z} \le \varepsilon$, *ahol* $\varepsilon = \sigma^2 / \left(6nw^2\sqrt{|\mathcal{B}|\pi_{\mathcal{B}}}\right)$, *akkor a kerekítési eljárás* $O(|\mathcal{B}^*|^3)$ *aritmetikai művelettel előállítja az (SP) feladat egy szigorúan komplementáris megoldását.*

A 25.36. lemma, amelynek az bizonyítását az Olvasóra bízzuk (lásd 25.2-14. gyakorlat), állítása a korábban már említett okok miatt racionális együtthatók esetén is igaz marad. Az eddigi eredményeket a következő tételben foglaljuk össze.

25.37. tétel. *A Dikin-féle affin skálázási algoritmussal egy* $\mathbf{u}^0 \in \mathcal{F}^0$ $(\mathbf{z}^0 > \mathbf{0})$ *pontból, amelyre* $\partial_c(\mathbf{u}^0) \leq \tau = 2$ *és* $(\mathbf{u}^0)^T\mathbf{z}^0 = n$ *teljesül, előállítható egy olyan* $\mathbf{u} \in \mathcal{F}^0$ $(\mathbf{z} > \mathbf{0})$ *pont, amelyből a kerekítési eljárással szigorúan komplementáris megoldás számolható ki. Az* $(\mathbf{u}, \mathbf{z})$ *pont előállításához* $\left\lceil 2n \lg\left(6n^2w^2\sqrt{|\mathcal{B}|}\pi_{\mathcal{B}}/\sigma^2\right)\right\rceil$ *iteráció szükséges.*

Gyakorlatok

25.2-1. Bizonyítsuk be, hogy a (25.9) optimalizálási feladatnak a $-\mathbf{c}/\|\mathbf{c}\|$ vektor az optimális megoldása.

25.2-2. Tekintsük az egyszerűség kedvéért a $B(\mathbf{0}, 1) = \{\mathbf{u} \in \mathbb{R}^n : \|\mathbf{u}\| \leq 1\}$ gömböt és az $E = \{\mathbf{y} \in \mathbb{R}^n : \mathbf{y}^T Q\mathbf{y} \leq 1\}$ ellipszoidot, ahol a $Q \in \mathbb{R}^{n\times n}$ szimmetrikus pozitív szemidefinit mátrix. Bizonyítsuk be, hogy létezik egy-egy értelmű megfeleltetés a $B(\mathbf{0}, 1)$ és az E halmazok között.

25.2-3. Legyen $D \in \mathbb{R}^{n\times n}$ pozitív definit, $M \in \mathbb{R}^{n\times n}$ ferdén szimmetrikus mátrix. Bizonyítsuk be, hogy a $D + M$ és az $I + DMD$ pozitív definit mátrixok.

25.2-4. Legyenek $A, B \in \mathbb{R}^{n\times n}$ pozitív definit mátrixok. Bizonyítsuk be, hogy az $A^{-1}B$ is pozitív definit mátrix.

25.2-5. Bizonyítsuk be a 25.23. állítás második egyenlőségét.

25.2-6. Ellenőrizzük, hogy az $\bar{\alpha}$ valóban megfelelő felső korlát a megengedett lépéshosszra. *Útmutatás.* Az $\bar{\alpha}$ értéket a (25.10) képlettel definiáltuk.

25.2-7. Ellenőrizzük az (25.12) egyenlőtlenségből nyerhető $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ korlátokat (lásd 1253. oldal).

25.2-8. A skálázásnál bevezetett jelöléseket felhasználva bizonyítsuk be, hogy

$$\|\mathbf{p}_u\,\mathbf{p}_z\|_\infty \leq \frac{1}{4}\|\mathbf{p}_u + \mathbf{p}_z\|^2 = \frac{1}{4}\|\mathbf{p}\|^2 .$$

25.2-9. Bizonyítsuk be a 25.24. lemmát. *Útmutatás.* Induljunk ki a (25.11) egyenlőség jobb oldalán álló kifejezésből és vezessük be az

$$f: \quad t \mapsto t - \alpha\frac{t^2}{\|\mathbf{v}^2\|}$$

függvényt, ahol az $\mathbf{v}^2$ vektor helyett írtuk be a t változót. A függvény elemzéséből kapjuk az első korlátot. A második korlát kiszámításakor használnunk kell a centralitási mértékre való elvárásokat és a 25.2-8. gyakorlat eredményét.

25.2-10. Legyen az $R \in \mathbb{Z}^{m\times m}$ reguláris mátrix és $\mathbf{r} \in \mathbb{Z}^m$ vektor. Bizonyítsuk be, hogy az $R\mathbf{u} = \mathbf{r}$ egyenlet bármely megoldása esetén, ha $y_i \neq 0$, akkor

$$y_i \geq \frac{1}{\prod_{j=1}^{n}\|\mathbf{r}_j\|} ,$$

ahol $\mathbf{r}_j$ az R mátrix j. oszlopa. *Útmutatás.* Alkalmazzuk a Cramer-szabályt és a Hadamard-egyenlőtlenséget.

25.2-11. Tekintsük a következő megengedettségi feladatot:

$$\mathcal{M} = \{\mathbf{y} \in \mathbb{R}^n \ ; \ R\mathbf{y} = \mathbf{r}, \ \mathbf{y} \geq \mathbf{0}\} \ ,$$

ahol $R \in \mathbb{Z}^{m \times n}$ és $\mathbf{r} \in \mathbb{Z}^m$, $\mathbf{r} \neq \mathbf{0}$. Legyen az $\mathcal{M}$ korlátos halmaz és tegyük fel, hogy tartalmaz egy pozitív vektort. Bizonyítsuk be, hogy bármely $i \in \mathcal{I}$ index esetén

$$\max_{\mathbf{y} \in \mathcal{M}} y_i \geq \frac{1}{\prod_{j=1}^{n} \|\mathbf{r}_j\|} \ .$$

Útmutatás. Vegyük figyelembe az R mátrix rangjának az értékét – legfeljebb m – és alkalmazzuk az előző gyakorlatot.

25.2-12. Bizonyítsuk be a 25.29. lemmát. *Útmutatás.* Számoljuk ki a következő összefüggést $(\mathbf{u}(\mu) - \mathbf{u}^*)^T(\mathbf{z}(\mu) - \mathbf{z}^*)$, ahol $(\mathbf{u}^*, \mathbf{z}^*)$ optimális megoldása az (SP) feladatnak.

25.2-13. Bizonyítsuk be a 25.32. lemmát.

25.2-14. Bizonyítsuk be a 25.36. lemmát. *Útmutatás.* Bizonyítsuk be, hogy az (25.15) egyenletnek van olyan megoldása, amelyre $\mathbf{u}_\mathcal{B} - \xi > \mathbf{0}$ és $\overline{\mathbf{z}}_\mathcal{N} = \mathbf{z}_\mathcal{N} - M_{\mathcal{N}\mathcal{N}}\,\mathbf{u}_\mathcal{N} - M_{\mathcal{N}\mathcal{B}}\,\xi > \mathbf{0}$ teljesül, az ε megállási paraméterre tett feltevés mellett.

25.2-15. Részletezzük a 25.37. tétel bizonyítását. *Útmutatás.* Alkalmazzuk a 25.36. lemmát és a 25.26. tételt.

25.2-16. Készítsük el a Dikin-algoritmus gyakorlati változatának MATLAB megvalósítását. Használjuk fel a MATLAB adta numerikus lehetőségeket (Cholesky-faktorizáció, ritka mátrixok használata stb.) és körültekintően (figyeljünk a beágyazásra, az induló belsőpontos megoldás meghatározására, az algoritmus paramétereinek a kellő megválasztására, az eredmény megfelelő értelmezésére stb.) végezzük el a számítógépes megvalósítást. Ha gondosan járunk el, akkor akár néhány száz feltételes és több száz változós gyakorlati feladatot – általában – száznál nem több iterációval meg tudunk oldani.

25.3. Primál-duál belsőpontos algoritmusok

A primál-duál belsőpontos algoritmusok a gyakorlatban is hatékonyan használható módszerek. Az ilyen algoritmusoknak két sarkalatos pontja van. Az egyik a μ centralitási paraméter iterálási módja, a másik pedig, hogy milyen centralitási mértéket használunk a pontunk és a centrális út távolságának mérésére. Az első kérdés szempontjából három csoportba oszthatjuk a primál-duál belsőpontos algoritmusokat. Általánosan a centralitási paramétert a $\mu^+ = (1 - \theta)\mu$ szabály szerint frissítjük, ahol $0 < \theta < 1$, és a θ értékének megfelelően osztályozunk.

Ha θ egy konstans szám, akkor az algoritmus hosszú lépéses, ha θ a feladat dimenziójától, azaz n értékétől függ, akkor pedig rövid lépéses módszerről beszélünk. A harmadik csoportot az adaptív algoritmusok alkotják, ahol a θ paraméter ellentétben az előző két esettel az iterációk során változik, függ az aktuális ponttól. A rövid lépéses módszerek $\mathcal{O}\left(\sqrt{n} \lg \frac{n}{\varepsilon}\right)$, míg a hosszú lépéses módszerek esetén $\mathcal{O}\left(n \lg \frac{n}{\varepsilon}\right)$ a bizonyított lépésszám felső korlát. Ez ellentmond a gyakorlati tapasztalatoknak, miszerint a hosszú lépéses algoritmusok hatékonyabbak.

A másik lényeges kérdés, a centralitási mérték megválasztása esetén is többfajta megoldással találkozunk a szakirodalomban. A centrális úton az $\mathbf{v}$ vektor minden koordinátája

egyenlő, így kézenfekvő a

$$\delta_c(\mathbf{v}) = \frac{\max(\mathbf{v}^2)}{\min(\mathbf{v}^2)}$$

módon definiálni a centralitási mértéket, ezt használtuk a Dikin-féle affin skálázású algoritmus elemzése során is. Ez a fajta centralitási mérték nem mindig felel meg céljainknak, mert a vektornak csak a maximális, illetve minimális elemét veszi figyelembe. Az alábbi mérték már a vektor összes koordinátájától függ, valamint a centrális úton lévő pontok esetén nulla az értéke, ami jobban tükrözi célunkat, miszerint a centrális út és az aktuális pont távolságát szeretnénk jellemezni vele.

$$\delta_0(\mathbf{v}) = \frac{1}{2}\left\|\mathbf{e} - \mathbf{v}^2\right\| .$$

Az ebben a részben használt centralitási mérték a szakirodalomban egyik leggyakrabban alkalmazott mérték, amelyet a következő képlettel definiálunk:

$$\delta_1(\mathbf{v}) = \frac{1}{2}\left\|\mathbf{v} - \mathbf{v}^{-1}\right\| .$$

A $\delta_1(\mathbf{v})$ centralitási mérték még inkább megfelel az elvárásainknak, mint a $\delta_0(\mathbf{v})$. Nem csak hogy eltűnik a centrális úton, hanem ha egy bizonyos korlát alatt tartjuk, akkor nem engedi a pontunkat a megengedett tartomány határhoz túl közel, ugyanis akár az $\mathbf{x}$, akár az $\mathbf{s}$ vektor valamely koordinátája tart nullához, vagy végtelenbe, akkor a pont centralitási mértéke is tart végtelenbe. (Ennek a tulajdonságnak csak a fele teljesül a δ_0 mérték esetén.) A δ_1 centralitási mérték általánosításának tekinthetők a legújabb, önreguláris függvények segítségével definiált mértékek (lásd a 25.3.5. pontot).

A fő célunk, amelyet a következő részben fejtünk ki, a gyakorlatban is hatékony belsőpontos algoritmusok bemutatása. Először a ***primál-duál Newton-lépéses belsőpontos algoritmust*** ismertetjük, mely teljes Newton-lépések megtételével konvergál a megoldáshoz. Egyszerűen bizonyítható az eljárás polinomialitása, belátható, hogy az algoritmus lépésszáma $O(\sqrt{n}\lg\frac{n}{\varepsilon})$. A primál-duál Newton-lépéses belsőpontos algoritmusok továbbfejlesztésének tekinthetjük a prediktor-korrektor algoritmusokat, melyek egyik első változatát Sonnevend, Stoer és Zhao vezette be lineáris programozási feladatokra. Sonnevendék algoritmusának a prediktor lépés után több korrektor lépésre volt szüksége ahhoz, hogy a centrális út megfelelő, előírt környezetébe visszajusson. Mizuno, Todd és Ye olyan ***prediktor-korrektor belsőpontos algoritmust*** publikált lineáris programozási feladatra, amelynél egy prediktor lépést csupán egy korrektor lépés követett, és az algoritmus lépésszámára adott felső korlát a lineáris programozás szakirodalmából ismert legjobb. Ez utóbbi algoritmus egy változatát fogjuk ismertetni. A rövid lépéses prediktor-korrektor algoritmus elemzése után egy adaptív változatot is bemutatunk, amely esetén már kvadratikus konvergencia is bizonyítható.

25.3.1. Primál-duál Newton-lépéses belsőpontos algoritmus

A lineáris programozási feladatok különböző, egymással ekvivalens formában adhatók meg. Azt, hogy éppen melyik alakját használjuk a lineáris programozási feladatnak, elsősorban az határozza meg, hogy milyen algoritmussal szeretnénk megoldani, illetve a feladat milyen megadása segíti legjobban elő elemzéseink egyszerű bemutatását. A lineáris programozás

belsőpontos szakirodalmában a ***primál*** és ***duál feladatokat*** leggyakrabban a következő alakban adjuk meg:

$$\left.\begin{array}{lcl} \min & \mathbf{c}^T\mathbf{x} & \\ A\mathbf{x} & = & \mathbf{b} \\ \mathbf{x} & \geq & \mathbf{0} \end{array}\right\} \quad (P), \qquad \left.\begin{array}{lcl} \max & \mathbf{b}^T\mathbf{y} & \\ A^T\mathbf{y} & \leq & \mathbf{c} \end{array}\right\} \quad (D)\,,$$

ahol $A \in \mathbb{R}^{m\times n}$, $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$és $\mathbf{y}, \mathbf{b} \in \mathbb{R}^m$. Az általánosság korlátozása nélkül feltehető, hogy $\operatorname{rang}(A) = m$. Legyen a ***primál,*** illetve a ***duál megengedett megoldások*** halmaza rendre

$$\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n_{\oplus} \,:\, A\mathbf{x} = \mathbf{b}\} \quad \text{és} \quad \mathcal{D} = \{(\mathbf{y}, \mathbf{s}) \in \mathbb{R}^{m+n} \,:\, A^T\mathbf{y} + \mathbf{s} = \mathbf{c},\ \mathbf{s} \geq \mathbf{0}\}\,.$$

Jelölje

$$\mathcal{P}_+ = \{\mathbf{x} \in \mathcal{P} :\ \mathbf{x} > \mathbf{0}\} \qquad \text{és} \qquad \mathcal{D}_+ = \{(\mathbf{y}, \mathbf{s}) \in \mathcal{D} :\ \mathbf{s} > \mathbf{0}\}$$

a ***primál,*** illetve ***duál belső pontok halmazát***.
Belső pont feltétel: létezik $\bar{\mathbf{x}} \in \mathcal{P}_+$ és létezik $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$.

A belső pont feltétel nem jelent megszorítást a 25.1. alfejezetben bemutatott beágyazási feladat (1243. oldal) segítségével bármely lineáris programozási feladat beágyazható egy olyan önduális feladatba, mely teljesíti a belső pont feltételt, és a beágyazási feladat megoldása az eredeti feladat egy megoldását adja. A belsőpontos feltételek teljesülése esetén, azaz ha $\mathcal{P}_+ \neq \emptyset$ és $\mathcal{D}_+ \neq \emptyset$, akkor az erős dualitás tétel alapján létezik primál (és duál) optimális megoldás.
A ***primál,*** illetve a ***duál optimális megoldások halmazát*** a következőképpen jelöljük:

$$\mathcal{P}^* = \{\mathbf{x}^* \in \mathcal{P} \,:\, \mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\mathbf{x},\ \ \forall \mathbf{x} \in \mathcal{P}\}$$
$$\text{és} \quad \mathcal{D}^* = \{\mathbf{y}^* \in \mathcal{D} \,:\, \mathbf{b}^T\mathbf{y}^* \geq \mathbf{b}^T\mathbf{y},\ \ \forall \mathbf{y} \in \mathcal{D}\}\,.$$

Dualitásrés: $\mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y} = (A^T\mathbf{y} + \mathbf{s})^T\mathbf{x} - \mathbf{y}^T(A\,\mathbf{x}) = \mathbf{s}^T\mathbf{x}$.
A következő állítás könnyen belátható, ezért az Olvasóra bízzuk (lásd 25.3-2. gyakorlat).

25.38. állítás. *Legyenek az $\mathbf{x}, \mathbf{s} \in \mathbb{R}^n$ olyan vektorok, amelyekre $\mathbf{x} \geq \mathbf{0}$ és $\mathbf{s} \geq \mathbf{0}$ teljesül. Az $\mathbf{x}$ és $\mathbf{s}$ vektorok pontosan akkor ortogonálisak, ha $x_i\, s_i = 0$ teljesül $i = 1, 2, \ldots, n$ esetén.*

Felhasználva az előző állítást az ***optimalitási kritériumok*** az alábbi formában írhatók

$$\begin{array}{rrcll} & A\,\mathbf{x} & = & \mathbf{b}, & \mathbf{x} \geq \mathbf{0}\,, \\ A^T\mathbf{y}+ & \mathbf{s} & = & \mathbf{c}, & \mathbf{s} \geq \mathbf{0}\,, \\ & \mathbf{x}\,\mathbf{s} & = & \mathbf{0}\,. & \end{array}$$

25.39. állítás. *Ha az $\bar{\mathbf{x}} \in \mathcal{P}_+$ és $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$, akkor $\bar{\mathbf{x}}\,\bar{\mathbf{s}} = \bar{\mathbf{w}} \in \mathbb{R}^n_+$, azaz az optimum belső pontban nem vétetik fel.*

A 25.39. állítás miatt, amelynek az bizonyítását az Olvasóra bízzuk (lásd 25.3-3. gyakorlat), az optimalitási kritériumokat relaxáljuk, olyan primál és duál belsőpontos megoldáspárt szeretnénk előállítani, amelyre

$$\begin{array}{rrcll} & A\,\mathbf{x} & = & \mathbf{b}, & \mathbf{x} > \mathbf{0}\,, \\ A^T\mathbf{y}+ & \mathbf{s} & = & \mathbf{c}, & \mathbf{s} > \mathbf{0}\,, \\ & \mathbf{x}\,\mathbf{s} & = & \mu\,\mathbf{e} & \end{array}$$

teljesül, ahol $\mu > 0$ adott. Az $(\bar{\mathbf{x}}, \bar{\mathbf{s}}) > 0$ primál-duál megoldásból szeretnénk az előző ***relaxált optimalitási kritériumok*** egy megoldását előállítani oly módon, hogy valamilyen ismeretlen $\Delta\mathbf{x}$, $\Delta\mathbf{y}$, $\Delta\mathbf{s}$ értékkel módosítjuk a megengedett megoldást:

$$\begin{array}{rrcll} A(\bar{\mathbf{x}}+\Delta\mathbf{x}) & & = & \mathbf{b}, & \bar{\mathbf{x}}+\Delta\mathbf{x} > \mathbf{0}\ , \\ A^T(\bar{\mathbf{y}}+\Delta\mathbf{y})+ & (\bar{\mathbf{s}}+\Delta\mathbf{s}) & = & \mathbf{c}, & \bar{\mathbf{s}}+\Delta\mathbf{s} > \mathbf{0}\ , \\ & (\bar{\mathbf{x}}+\Delta\mathbf{x})(\bar{\mathbf{s}}+\Delta\mathbf{s}) & = & \mu\,\mathbf{e}\ . & \end{array}$$

Egyszerű számítások után kapjuk, hogy

$$\begin{array}{rrrcll} A\Delta\mathbf{x} & & & = & \mathbf{0}, & \bar{\mathbf{x}}+\Delta\mathbf{x} > \mathbf{0}\ , \\ A^T\Delta\mathbf{y}+ & & \Delta\mathbf{s} & = & \mathbf{0}, & \bar{\mathbf{s}}+\Delta\mathbf{s} > \mathbf{0}\ , \\ \bar{\mathbf{s}}\,\Delta\mathbf{x}+ & \bar{\mathbf{x}}\,\Delta\mathbf{s}+ & \Delta\mathbf{x}\,\Delta\mathbf{s} & = & \mu\,\mathbf{e}-\bar{\mathbf{x}}\,\bar{\mathbf{s}}\ , & \end{array}$$

ahol az utolsó egyenlet nemlineáris (kvadratikus). Ezt a nemlineáris egyenlőtlenségrendszert egyszerűsítjük. Két dolgot hagyunk el: a pozitivitási megkötéseket és a másodrendű tagot. Ekkor a harmadik egyenlet a következő alakúvá válik

$$\bar{\mathbf{s}}\,\Delta\mathbf{x} + \bar{\mathbf{x}}\,\Delta\mathbf{s} = \mu\,\mathbf{e} - \bar{\mathbf{x}}\,\bar{\mathbf{s}}\ .$$

Newton-rendszer: keresendő a $(\Delta\mathbf{x}, \Delta\mathbf{y}, \Delta\mathbf{s}) \in \mathbb{R}^{n+m+n}$ vektor úgy, hogy

$$\begin{array}{rrrcl} A\Delta\mathbf{x} & & & = & \mathbf{0}\ , \\ A^T\Delta\mathbf{y}+ & & \Delta\mathbf{s} & = & \mathbf{0}\ , \\ \bar{\mathbf{s}}\,\Delta\mathbf{x}+ & \bar{\mathbf{x}}\,\Delta\mathbf{s} & & = & \mu\,\mathbf{e}-\bar{\mathbf{x}}\,\bar{\mathbf{s}}\ . \end{array}$$

25.40. állítás. *Legyen* $\operatorname{rang}(A) = m$. *A Newton rendszer megoldása egyértelmű.*

A bizonyítást az Olvasóra bízzuk (lásd 25.3-4. gyakorlat).

Legyenek $\bar{S} = \operatorname{diag}(\bar{\mathbf{s}})$ és $\bar{X} = \operatorname{diag}(\bar{\mathbf{x}})$ pozitív mátrixok. Ekkor az utolsó egyenletet a következő formára hozzuk

$$\bar{S}\,\Delta\mathbf{x} + \bar{X}\,\Delta\mathbf{s} = \mu\,\mathbf{e} - \bar{S}\,\bar{\mathbf{x}}\ , \quad \text{azaz} \quad \Delta\mathbf{x} + \bar{S}^{-1}\bar{X}\,\Delta\mathbf{s} = \mu\,\bar{\mathbf{s}}^{-1} - \bar{\mathbf{x}}\ .$$

Alkalmazzuk az A mátrixot a kapott n-dimenziós vektorokra és használjuk fel az $\bar{\mathbf{x}} \in \mathcal{P}$ összefüggést, illetve a Newton-rendszer első egyenletét. Ekkor

$$A\bar{X}\bar{S}^{-1}\Delta\mathbf{s} = A\,\Delta\mathbf{x} + A\bar{X}\bar{S}^{-1}\Delta\mathbf{s} = \mu\,A\,\bar{\mathbf{s}}^{-1} - A\,\bar{\mathbf{x}} = \mu\,A\,\bar{\mathbf{s}}^{-1} - \mathbf{b}\ .$$

A Newton-rendszer második egyenletére alkalmazzuk az $A\bar{X}\bar{S}^{-1}$ mátrixot, majd pedig rendezzük át az egyenletet. Ekkor

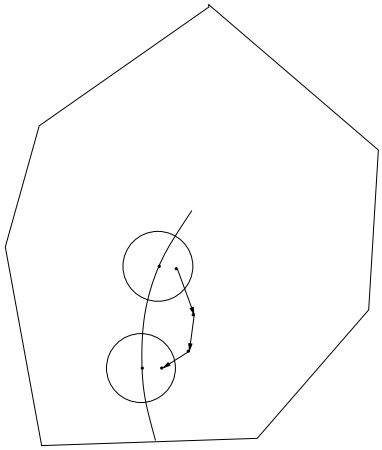
$$-A\,\bar{X}\,\bar{S}^{-1}\,A^T\Delta\mathbf{y} = A\bar{X}\bar{S}^{-1}\Delta\mathbf{s} = \mu\,A\bar{\mathbf{s}}^{-1} - \mathbf{b}\ .$$

Mivel $\operatorname{rang}(A) = m$, ezért $A\bar{X}\bar{S}^{-1}A^T \in \mathbb{R}^{m\times m}$ és nem szinguláris mátrix, hiszen az A teljes rangú, így $\Delta\mathbf{y}$ egyértelműen kiszámítható:

$$\Delta\mathbf{y} = (A\bar{X}\bar{S}^{-1}A^T)^{-1}(\mathbf{b} - \mu\,A\bar{\mathbf{s}}^{-1})\ . \tag{25.16}$$

Ezek után egyszerűen és egyértelműen kiszámíthatók a $\Delta\mathbf{s}$ és a $\Delta\mathbf{x}$ vektorok, azaz

$$\Delta\mathbf{s} = -A^T\Delta\mathbf{y} \qquad \text{és} \qquad \Delta\mathbf{x} = \mu\,\bar{\mathbf{s}}^{-1} - \bar{\mathbf{x}} - \bar{X}\bar{S}^{-1}\Delta\mathbf{s}\ . \tag{25.17}$$



25.3. ábra. Primál-duál algoritmus iterációi a primál változók terében.

Tekintettel arra, hogy kiszámoltuk az adott megoldás módosítására szolgáló vektorokat, amelyek segítségével a μ-centrumot megközelíthetjük, minden lényeges információval rendelkezünk ahhoz, hogy megfogalmazzuk az algoritmusunkat. A μ-centrumtól vett távolságot a $\partial_1(\mathbf{x}, \mathbf{s}, \mu) = \delta_1(\mathbf{v})$ centralitási mértékkel mérjük, ahol $\mathbf{v} = \sqrt{\mathbf{xs}/\mu}$.

Az eljárás bemenő adataként megadjuk az $\varepsilon > 0$ megkívánt számítási pontosságot, a $\tau \geq 0$ eltérés paramétert, a θ $(0 < \theta < 1)$ előre rögzített csökkenési paramétert, valamint egy $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0)$ belső pontot, amely a τ által meghatározott környezetben van, azaz $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0) \in \mathcal{P}_+ \times \mathcal{D}_+$, $(\mathbf{x}^0)^T \mathbf{s}^0 = \mu^0 n$ és $\partial_1(\mathbf{x}^0, \mathbf{s}^0, \mu^0) \leq \tau$.

PRIMÁL-DUÁL-NEWTON-LÉPÉSES-BELSŐPONTOS

```
1  x ← x^0
2  s ← s^0
3  μ ← μ^0
4  while nμ > ε
5        do μ ← (1 − θ)μ
6           while ∂_1(x, s; μ) > τ
7                 do határozzuk meg Δx-et és Δs-t (25.17) alapján
8                    számítsuk ki az α > 0 megengedett lépéshosszt
9                    x ← x + αΔx
10                   s ← s + αΔs
```

A fenti algoritmus egy belső iterációjának lépéseit a 25.3. ábra szemlélteti.

Az algoritmus esetén a gyakorlati és elméleti változatokat az különbözteti meg, hogy a τ és a θ paraméterek, valamint az α lépéshossz megválasztása hogyan történik. Világos az algoritmus szerkezetéből, hogy a θ paraméter megválasztása befolyásolni fogja azt, hogy a belső ciklust hányszor kell majd végrehajtani. A belső ciklus elvégzésének a számát befolyásolhatjuk azzal, hogy a τ értékét a θ értékétől függően választjuk meg. A θ rögzítése

természetesen kihat az α lehetséges értékére is. Az α lépéshossz megválasztásakor az a célunk, hogy a centralitási mértéket

$$\vartheta_1(\mathbf{x} + \alpha\,\Delta\mathbf{x}, \mathbf{s} + \alpha\,\Delta\mathbf{s}; \mu)$$

minimalizáljuk az α paraméterben.

Mielőtt rátérnénk az algoritmus egy lehetséges gyakorlati, illetve elméleti változatának az elemzésére, néhány olyan állítással foglalkozunk, amelyek ismeretére mindenfeleképpen szükségünk lesz. Emellett az iterációnként előforduló feladatot, ún. ***átskálázással*** egységes formára hozzuk. Ez az egységes forma a számításainkat és az elemzéseinket is leegyszerűsíti.

25.41. lemma. *$\Delta\mathbf{x}$ és $\Delta\mathbf{s}$ ortogonális vektorok.*

A $\Delta\mathbf{x}$ és a $\Delta\mathbf{s}$ Newton-lépések merőlegességét, amelynek az bizonyítását az Olvasóra bízzuk (lásd 25.3-5. gyakorlat), felhasználva szükséges és elégséges feltételt adhatunk a teljes Newton-lépés megengedettségére. (Ennek az bizonyítása is egy újabb feladat az Olvasó számára, lásd 25.3-6. gyakorlat.)

25.42. lemma. *A (P) és (D) feladatok esetén $\bar{\mathbf{x}} + \Delta\mathbf{x} \geq \mathbf{0}$ és $\bar{\mathbf{s}} + \Delta\mathbf{s} \geq \mathbf{0}$ pontosan akkor teljesül, ha $\mu\,\mathbf{e} + \Delta\mathbf{x}\,\Delta\mathbf{s} \geq \mathbf{0}$, ahol $\bar{\mathbf{x}} \in \mathcal{P}_+$, $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$ és $\mu > 0$ a rögzített cél centrum paramétere.*

A továbbiakban jelölje

$$\mathbf{x}^+ = \bar{\mathbf{x}} + \Delta\mathbf{x}\,, \quad \mathbf{y}^+ = \bar{\mathbf{y}} + \Delta\mathbf{y} \quad \text{és} \quad \mathbf{s}^+ = \bar{\mathbf{s}} + \Delta\mathbf{x}$$

a Newton-lépéssel kapott új pontokat. Az új pontok által meghatározott dualitásrés kiszámítása is az Olvasó feladata lesz (lásd 25.3-8. gyakorlat), amelyet az alábbi lemmában fogalmazunk meg.

25.43. lemma. *A (P) és (D) feladatok esetén, ha a $(\Delta\mathbf{x}, \Delta\mathbf{s})$ olyan, hogy $\mathbf{x}^+ \in P_+$, $(\mathbf{y}^+, \mathbf{s}^+) \in D_+$, akkor $(\mathbf{x}^+)^T\mathbf{s}^+ = \mu\,n$, ahol $\mu > 0$ a rögzített cél centrum paramétere.*

Átskálázás

Célunk olyan jelölések bevezetése, melyekkel a Newton-rendszer egyszerűbb alakra hozható, ezzel kiemelve a feladat struktúráját. Az alábbi vektorok segítségével átírt Newton-rendszer harmadik feltételének jobb oldalán megjelenik a $\mathbf{v}^{-1} - \mathbf{v}$ vektor, melynek normája fogja mérni a pontunk távolságát a centrális úttól, ugyanis ha a pontunk rajta van a centrális úton, akkor ez a kifejezés nullával egyenlő. Legyen $\bar{\mathbf{x}} \in \mathcal{P}_+$ és $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$. Definiáljuk a következő vektorokat:

$$\mathbf{d} = \sqrt{\frac{\bar{\mathbf{x}}}{\bar{\mathbf{s}}}}\,, \quad \mathbf{v} = \sqrt{\frac{\bar{\mathbf{x}}\,\bar{\mathbf{s}}}{\mu}}\,, \qquad \text{ekkor} \qquad \frac{\mathbf{d}^{-1}\,\bar{\mathbf{x}}}{\sqrt{\mu}} = \frac{\mathbf{d}\,\bar{\mathbf{s}}}{\sqrt{\mu}} = \mathbf{v}\,.$$

Továbbá

$$\mathbf{d}_x = \frac{\mathbf{d}^{-1}\,\Delta\mathbf{x}}{\sqrt{\mu}} \qquad \text{és} \qquad \mathbf{d}_s = \frac{\mathbf{d}\,\Delta\mathbf{s}}{\sqrt{\mu}}\,.$$

Ekkor

$$\mathbf{x}^+ = \bar{\mathbf{x}} + \Delta\mathbf{x} = \sqrt{\mu}\,\mathbf{d}\,(\mathbf{v} + \mathbf{d}_x)\,, \quad \text{és} \quad \mathbf{s}^+ = \bar{\mathbf{s}} + \Delta\mathbf{s} = \sqrt{\mu}\,\mathbf{d}^{-1}\,(\mathbf{v} + \mathbf{d}_s)\,, \quad \text{valamint}$$

$$\mathbf{x}^+\,\mathbf{s}^+ = \mu\,\mathbf{e} + \Delta\mathbf{x}\,\Delta\mathbf{s} = \mu\,\mathbf{e} + \sqrt{\mu}\,\mathbf{d}\,\mathbf{d}_x\;\sqrt{\mu}\,\mathbf{d}^{-1}\,\mathbf{d}_s = \mu\,(\mathbf{e} + \mathbf{d}_x\,\mathbf{d}_s)\,.$$

Az új jelölésekkel a 25.42. lemma eredménye a következő alakban írható, amelynek a bizonyítása az Olvasó feladata lesz (lásd 25.3-9. gyakorlat).

25.44. következmény. *A* (P) *és* (D) *feladatok esetén* $\bar{\mathbf{x}} + \Delta\mathbf{x} \geq \mathbf{0}$ *és* $\bar{\mathbf{s}} + \Delta\mathbf{s} \geq \mathbf{0}$ *pontosan akkor teljesül, ha* $\mathbf{e} + \mathbf{d}_x\,\mathbf{d}_s \geq \mathbf{0}$, *ahol* $\bar{\mathbf{x}} \in \mathcal{P}_+$ *és* $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$.

Mivel $\bar{\mathbf{s}}\,\mathbf{d} = \bar{\mathbf{x}}\,\mathbf{d}^{-1} = \sqrt{\mu}\,\mathbf{v}$, ezért

$$\bar{\mathbf{x}}\,\Delta\mathbf{s} + \bar{\mathbf{s}}\,\Delta\mathbf{x} = \bar{\mathbf{x}}\,\sqrt{\mu}\,\mathbf{d}^{-1}\,\mathbf{d}_s + \bar{\mathbf{s}}\,\sqrt{\mu}\,\mathbf{d}\,\mathbf{d}_x = \sqrt{\mu}\,(\bar{\mathbf{s}}\,\mathbf{d}\,\mathbf{d}_x + \bar{\mathbf{x}}\,\mathbf{d}^{-1}\,\mathbf{d}_s) = \mu\,\mathbf{v}\,(\mathbf{d}_x + \mathbf{d}_s)\,.$$

Másfelől $\mu\,\mathbf{e} - \bar{\mathbf{x}}\,\bar{\mathbf{s}} = \bar{\mathbf{x}}\,\Delta\mathbf{s} + \bar{\mathbf{s}}\Delta\mathbf{x} = \mu\,\mathbf{v}\,(\mathbf{d}_x + \mathbf{d}_s)$, és $\mu\,\mathbf{e} - \bar{\mathbf{x}}\,\bar{\mathbf{s}} = \mu\,\mathbf{e} - \mu\,\mathbf{v}^2 = \mu\,\mathbf{v}\,(\mathbf{v}^{-1} - \mathbf{v})$, tehát $\mathbf{d}_x + \mathbf{d}_s = \mathbf{v}^{-1} - \mathbf{v}$. Bevezetve a $\mathbf{d}_y = \Delta\mathbf{y}/\sqrt{\mu}$ vektort, az átskálázott Newton-rendszer a következő lesz:

$$\begin{aligned} AD\,\mathbf{d}_x \qquad\quad &= \mathbf{0}\,, \\ (AD)^T\,\mathbf{d}_y + \quad \mathbf{d}_s \;&= \mathbf{0}\,, \\ \mathbf{d}_x + \quad \mathbf{d}_s \;&= \mathbf{v}^{-1} - \mathbf{v}\,. \end{aligned}$$

Tehát a $\mathbf{d}_x \in \mathrm{Null}(AD)$, és a $\mathbf{d}_s \in \mathrm{rowsp}(AD) = \mathrm{Null}(HD^{-1})$, ahol H tetszőleges olyan mátrix, amelynek nulltere megegyezik az A mátrix sorterével.[14] Ekkor

$$\mathbf{d}_x = P_{AD}(\mathbf{v}^{-1} - \mathbf{v})\,, \quad \text{és} \quad \mathbf{d}_s = P_{HD^{-1}}(\mathbf{v}^{-1} - \mathbf{v})\,, \tag{25.18}$$

és mivel $(\Delta\mathbf{x})^T\,\Delta\mathbf{s} = \mathbf{d}_x^T\,\mathbf{d}_s = 0$, ezért

$$\|\mathbf{d}_x\|^2 + \|\mathbf{d}_s\|^2 = \|\mathbf{v}^{-1} - \mathbf{v}\|^2\,.$$

A $\mathbf{d}_x + \mathbf{d}_s = \mathbf{0}$ egyenlőség pontosan akkor teljesül, ha $\mathbf{v}^{-1} - \mathbf{v} = \mathbf{0}$, ami azzal ekvivalens, hogy $\mathbf{v}^{-1} = \mathbf{v}$, azaz $\mathbf{v} = \mathbf{e}$. Átskálázás után centralitási mértékünk a következő alakú lesz

$$\partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \mu) = \frac{1}{2}\left\|\sqrt{\frac{\mu\,\mathbf{e}}{\bar{\mathbf{x}}\,\bar{\mathbf{s}}}} - \sqrt{\frac{\bar{\mathbf{x}}\,\bar{\mathbf{s}}}{\mu}}\right\| = \frac{1}{2}\,\|\mathbf{v}^{-1} - \mathbf{v}\|\,.$$

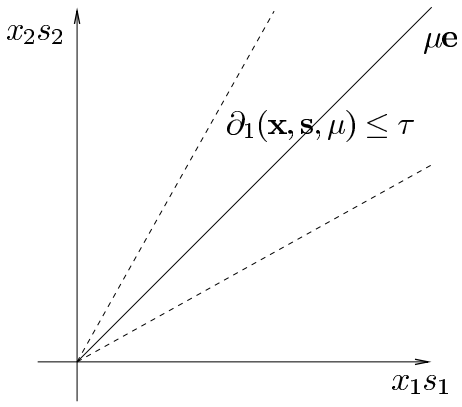
A 25.4. ábra a centrális út környezetét szemlélteti, azaz azon pontok halmazát az átskálázott terében, melyek centralitási mértéke egy megadott korlát alatt marad.

25.3.2. Primál-duál Newton-lépéses belsőpontos algoritmus: gyakorlati változat

Az algoritmus gyakorlati változatának az elemzésekor a következő feltételezésekkel élünk: a τ és a θ paraméterek értéke nem függ a feladat változóinak a számától, legyen például $\tau = 100$ és $\theta = 0.9$.

Tegyük fel, hogy adottak az $\bar{\mathbf{x}} \in \mathcal{P}^+$ és az $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}^+$ megoldás vektorok, amelyekhez

[14] $\mathrm{Null}(M)$ az M mátrix nullterét, míg rowsp(M) a mátrix sorterét jelöli. P_M pedig az a projekció, mely az M mátrix nullterére képez.



25.4. ábra. Centrális út környezete az átskálázott térben, ahol $\mu = \mathbf{x}^T\mathbf{s}/n$ és $x_1s_1 = \mu v_1^2$, $x_2s_2 = \mu v_2^2$.

$\bar{\mu} = \bar{\mathbf{x}}^T\bar{\mathbf{s}}/n$ dualitásrés tartozik, és teljesítik a $\partial_1(\bar{\mathbf{x}},\bar{\mathbf{s}};\bar{\mu}) \le \tau = 100$ egyenlőtlenséget.

A Primál-duál-Newton-lépéses-belsőpontos algoritmust numerikus szempontból elemezve észrevehetjük, hogy az algoritmus leginkább munkaigényes része a $\Delta\mathbf{y}$ meghatározása, (25.16), hiszen az $R = A\bar{X}\bar{S}^{-1}A^T$ mátrix inverzének a kiszámítását igényeli. Ezzel szemben $\Delta\mathbf{s}$ és $\Delta\mathbf{x}$ (25.17) szerinti meghatározása már csak mátrix-vektor szorzást, illetve mátrix-vektor szorzást és vektor-vektor összeadást igényel.

Tegyük fel, hogy a MATLAB programrendszer segítségével szeretnénk a primál-duál Newton-lépéses belsőpontos algoritmus egy implementációját elkészíteni, ekkor már a $\Delta\mathbf{y}$ kiszámítására is több lehetőségünk nyílik:

1. `Δ y = inv(R)*r;`
2. `Δ y = R \ r;`

MATLAB utasítással számoljuk ki a $\Delta\mathbf{y}$ vektort, ahol $\mathbf{r} = \mathbf{b} - \mu A\bar{\mathbf{s}}^{-1}$.

Köztudott, hogy a mátrix invertálás numerikus szempontból az egyik leginkább instabil számítási eljárás, ezért más módszerhez célszerű folyamodnunk. A megoldandó feladatunk numerikus szempontból ugyanis az

$$R\ \Delta\mathbf{y} = \mathbf{r} \tag{25.19}$$

lineáris egyenletrendszer megoldása, ahol az $R \in \mathbb{R}^{m\times m}$ reguláris mátrix. Figyelembe véve, hogy az $\bar{X}, \bar{S}^{-1}$ pozitív diagonális mátrixok és $\operatorname{rang}(A) = m$, ezért az R mátrix szimmetrikus és pozitív definit lesz. A pozitív definit mátrixok Cholesky-faktorizációja numerikusan jóval stabilabb számítási eljárás, mint az invertálás, és az eredménye $R = LL^T$ alakú, ahol $L \in \mathbb{R}^{m\times m}$ alsó háromszög mátrix. Ekkor az egyenletünk

$$R\Delta\mathbf{y} = L(L^T\Delta\mathbf{y}) = \mathbf{r}$$

alakú lesz, és az

$$L\mathbf{z} = \mathbf{r} \quad \text{és} \quad L^T\Delta\mathbf{y} = \mathbf{z}$$

helyettesítésekkel a (25.19) egyenletrendszer megoldását két háromszög mátrixszal adott

lineáris egyenletrendszer megoldására vezettük vissza. Tehát a harmadik lehetőségünk a Cholesky-faktorizáció

3. `L = chol(R) ;  z = L \ r;  Δ y = L' \ z;`

Ebben az esetben célszerű elkészíteni a visszahelyettesítésen alapuló lineáris egyenletrendszert megoldó eljárást. Megjegyeznénk, hogy a MATLAB speciális esetben a 2. pontban tárgyalt LU felbontás esetén Cholesky-faktorizációt hajt végre.

Numerikus bonyodalmak

Tekintettel arra, hogy az $\mathbf{x}\,\mathbf{s} = \mu\,\mathbf{e}$ egyenlet megoldásait közelítjük az $\bar{\mathbf{x}}$ és $\bar{\mathbf{s}}$ vektorokkal, és a μ paraméterrel nullához tartunk (külső ciklus), nem meglepő, hogy minél kisebb lesz a μ, annál inkább rosszul kondicionált lesz a (25.19) egyenletrendszer mátrixa. Ennek a természetesen fellépő numerikus jelenségnek a kiküszöbölésére az ún. ***regularizációt***[15] ajánljuk, azaz az R mátrix helyett dolgozzunk az

$$R_\rho = \rho\, I + A\,(\rho\, I + \bar{X}\,\bar{S}^{-1})\,A^T$$

mátrixszal, ahol $\rho = 10^{-8}$. (Feltételezzük, hogy 32-bites gépen dolgozunk, dupla pontosságú lebegőpontos aritmetikával.) Az R_ρ mátrix, az R mátrixhoz hasonlóan pozitív definit lesz, de a sajátértékei soha sem válhatnak 10^{-8}-nál kisebbé.[16] A regularizáció a következő megfontolásokon alapul: mivel a numerikus gondok forrása az, hogy az $\bar{X}\,\bar{S}^{-1}$ mátrix diagonális elemei kicsikké válnak, a $\rho\, I$ mátrix hozzáadásával elértük, hogy a diagonális elemek értéke 10^{-8}-nál nem lehet kisebb.[17] Másfelől, az R mátrix rosszul kondicionáltságának az oka lehet maga az A mátrix is. Célszerű tehát az $A\,(\rho\, I + \bar{X}\,\bar{S}^{-1})\,A^T$ mátrixhoz is hozzáadni a $\rho\, I$ mátrixot. Ezt a módosítást az algoritmus legelején vezessük be, mert amíg az R sajátértékei 10^{-8}-nál nagyobbak, addig ennek úgy sincsen jelentős hatása, amikor pedig kisebbé válnak, akkor a regularizáció a számítások elvégzéséhez nélkülözhetetlen lesz.

Természetesen más módszer is ismert a szakirodalomból a mátrixok kondíciószámának a karbantartására.

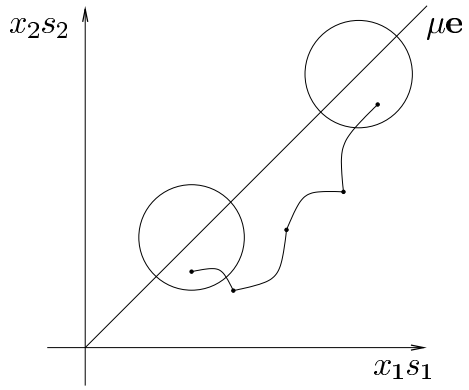
Az α lépéshossz meghatározása

Miután kiszámítottuk a $\Delta\mathbf{x}$ és $\Delta\mathbf{s}$ irányokat, az $\alpha > 0$ lépéshosszt szeretnénk úgy meghatározni, hogy az megengedett legyen, azaz az

$$\bar{\mathbf{x}} + \alpha\,\Delta\mathbf{x} > \mathbf{0} \qquad \text{és} \qquad \bar{\mathbf{s}} + \alpha\,\Delta\mathbf{s} > \mathbf{0}$$

egyenlőtlenségek teljesüljenek. Ezekből az egyenlőtlenségekből kiszámítható az $\bar{\alpha} > 0$ felső korlát, (25.10). Világos, hogy létezik olyan index, amelyre vagy $\bar{x}_i + \bar{\alpha}\,(\Delta\mathbf{x})_i = 0$ vagy $\bar{s}_i + \bar{\alpha}\,(\Delta\mathbf{s})_i = 0$, teljesül. Ezért az $\bar{\alpha}$ érték esetén a ∂_1 centralitási mérték nem értelmezett. Elvileg a $\partial_1(\bar{\mathbf{x}} + \alpha\,\Delta\mathbf{x}, \bar{\mathbf{s}} + \alpha\,\Delta\mathbf{s}; \mu)$ minimumát kell meghatározni a $(0, \bar{\alpha})$ nyílt intervallumon. Figyelembe véve a ∂_1 függvény domináns büntető tulajdonságát, amikor a feltételek felé közelítünk, intuitíven, numerikus stabilitást is figyelembe véve belátható, hogy a ∂_1 függvényt minimalizáló α értéket a $[0.005, 0.995]$ intervallumban kell keresnünk. Alkalmazhatnánk iránymenti minimalizálást is, de a ∂_1 függvény tulajdonságai alapján nyilvánvaló, hogy a

[15] A regularizáció kérdésére a *Megjegyzések a fejezethez* című részben még kitérünk.
[16] Az R_ρ mátrix kondíciószáma korlátos marad.
[17] Ez azt jelenti, hogy a 10^{-8}-nál kisebb számokat 10^{-8}-nak tekintjük.



25.5. ábra. Primál-duál algoritmus iterációi az átskálázott térben, ahol $x_1s_1 = \mu v_1^2$ és $x_2s_2 = \mu v_2^2$.

keresett α érték a felső korlát közelében lesz. Egy egyszerű visszaléptetéses algoritmussal kereshetjük meg közelítőleg a ∂_1 minimumát adó α értéket.

Ezekből az elemekből összeállítható MATLAB-ban olyan megvalósítás, amely esetén néhány száz feltételes, és több száz (akár ezer) változós gyakorlati feladatot is rövid idő alatt megoldhatunk, akár $\varepsilon = 10^{-8}$-os pontossággal. A tapasztalat szerint, ha a θ értéket nagyobbra választjuk, például $\theta = 0.99$, akkor a megadott méretű gyakorlati feladatoknál, ahol az A mátrix ritka, általában 30-nál nem lesz több iterációra szükségünk.

Nagyméretű lineáris programozási feladatok megoldásához szükséges jobban kihasználni a gyakorlati feladatok mátrixának ritkaságát a Cholesky-faktorizációnál, ezért az ilyen feladatok esetén általában szükséges a MATLAB Cholesky-faktorizációjánál jobb faktorizációs eljárást alkalmazni.

25.3.3. Primál-duál Newton-lépéses belsőpontos algoritmus: elméleti változat

Az elméleti algoritmus változat elemzésének a szokásos menete a következő: tetszőlegesen rögzítsük le a θ és a τ paraméterek értékét, a változók számától függetlenül. Legyen a $\mu^+ = (1-\theta)\mu$ és tegyük fel, hogy az adott $\bar{\mathbf{x}} \in \mathcal{P}_+$, $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$ megoldás esetén $\partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \mu^+) > \tau$ teljesül.[18] A belső ciklusban az a célunk, hogy néhány iterációval a μ^+-centrum τ-környezetébe kerüljünk (lásd a 25.5. ábrát, mely egy belső ciklus lépéseit szemlélteti). A lineáris programozás belsőpontos szakirodalmából ismert a következő meglepő eredmény: az $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{P}_+ \times \mathcal{D}_+$ megoldásból indulva az algoritmusban az előírt utasításokat végrehajtva, legfeljebb $O(n)$ számú megoldás kiszámításával, a μ^+-centrum τ-környezetébe jutunk. Tehát a belső ciklus meghívására legfeljebb $O(n)$ alkalommal kerülhet sor, mielőtt a külső ciklusban a μ centralitási paraméter értékét csökkentenénk.

25.45. tétel. *A* Primál-duál-Newton-lépéses-belsőpontos *algoritmus esetén egy adott* $\bar{\mathbf{x}} \in$

[18] Ha nem így volna, akkor a μ paraméter értékét ismét $(1-\theta)$-szorosára csökkentjük és újra kiszámítjuk a centralitási mértékét az aktuális cél centrum paraméterével és a megoldás segítségével.

$\mathcal{P}_+$ és $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$ megoldásból indítva az algoritmust legfeljebb

$$\left\lceil \frac{1}{\theta} \lg \frac{\bar{\mathbf{x}}^T \bar{\mathbf{s}}}{\varepsilon} \right\rceil$$

alkalommal módosítjuk a μ centralitási paramétert, mielőtt az algoritmus előállít egy $\hat{\mathbf{x}} \in \mathcal{P}_+$ és $(\hat{\mathbf{y}}, \hat{\mathbf{s}}) \in \mathcal{D}_+$ megoldást, amely esetén az $\hat{\mathbf{x}}^T \hat{\mathbf{s}} \leq \varepsilon$ teljesül.

Bizonyítás. Legyen $(\mathbf{x}^0, \mathbf{s}^0) = (\bar{\mathbf{x}}, \bar{\mathbf{s}})$. Tekintsük azt a megoldás sorozatot, amelyre a $\partial_1(\mathbf{x}^i, \mathbf{s}^i; \mu^i) \leq \tau$ teljesül, ezek azok a megoldások, amelyek előállítása után a μ centralitási paramétert csökkenti az algoritmus. Legyen ez az

$$(\mathbf{x}^0, \mathbf{s}^0),\ (\mathbf{x}^1, \mathbf{s}^1),\ \ldots,\ (\mathbf{x}^k, \mathbf{s}^k),\ \ldots$$

pontsorozat, amelynek az elemei rendre a $\mu^0, \mu^1, \ldots, \mu^k, \ldots$ paraméterű centrumokhoz tartoznak, és az azoktól mért távolságuk nem nagyobb, mint τ. Sőt a pontpárokhoz tartozó dualitásrés egyre kisebb, hiszen $(\mathbf{x}^k)^T \mathbf{s}^k = \mu^k n = (1-\theta)^k \mu^0 n$, ahol $1 - \theta < 1$, tehát ez a sorozat tart a nullához. Azaz bármely $\varepsilon > 0$ adott pontossághoz létezik k index úgy, hogy

$$(\mathbf{x}^k)^T \mathbf{s}^k = \mu^k n = (1-\theta)^k \mu^0 n \leq \varepsilon, \qquad \text{ekkor} \qquad k \lg(1-\theta) + \lg(\mu_0 n) \leq \lg \varepsilon\ ,$$

a logaritmus monoton növekedése miatt. Figyelembe véve a $-\lg(1-\theta) \geq \theta$ összefüggést, az előző egyenlőtlenségnél egy erősebbet követelünk meg, amely valamely, az előzőleg megadott k indexnél nagyobbra teljesül

$$k \lg(1-\theta) + \lg(\mu^0 n) \leq -k\theta + \lg(\mu^0 n) \leq \lg \varepsilon\ ,$$

tehát

$$\lg(\mu^0 n) - \lg \varepsilon \leq k\theta\ ,$$

ezért

$$\frac{1}{\theta} \lg \frac{\mu^0 n}{\varepsilon} \leq k\ ,$$

valamely $k \in \mathbb{N}$ esetén. Tehát az $(\hat{\mathbf{x}}, \hat{\mathbf{s}}) = (\mathbf{x}^k, \mathbf{s}^k)$ jelölést használva, teljesül $\hat{\mathbf{x}}^T \hat{\mathbf{s}} \leq \varepsilon$. ■

Tekintettel arra, hogy a belső ciklus iteráció számának a meghatározása meghaladná a fejezet terjedelmét és a felhasznált szakmai eszköztárát, inkább egy érdekes, speciális paraméter választás mellett bizonyítjuk be a Primál-duál-Newton-lépéses-belsőpontos algoritmus polinomiális lépésszámának legfőbb részleteit.

Az egyszerűség kedvéért válasszuk a $\theta = 1/(2\sqrt{n})$ és $\tau = 1/\sqrt{2}$ paraméter értékeket. Ebben az esetben a külső ciklus kiértékelésére

$$2\sqrt{n} \lg \frac{\mu^0 n}{\varepsilon} \leq k, \quad \text{azaz} \quad \left\lceil 2\sqrt{n} \lg \frac{\mu^0 n}{\varepsilon} \right\rceil = k$$

alkalommal kerül sor.[19] Elemzésünk fő tétele az lesz, hogy az adott paraméterek mellett a belső ciklus kiértékelésére minden alkalommal pontosan egyszer kerül sor és ekkor $\alpha = 1$,

[19] Az általánosság korlátozása nélkül feltehetjük, hogy $\mu^0 = 1$, például a beágyazás miatt (lásd 25.1. alfejezet).

tehát teljes Newton-lépést teszünk a célul kitűzött μ-centrum irányába, és az egy lépéssel előállított megoldás a centrum τ-környezetében lesz.

Nyilvánvaló, hogy az algoritmus iterációinak számára felső becslést kapunk, ha a belső ciklus iterációinak felső korlátját – speciális esetünkben az 1-et – megszorozzuk a külső ciklus kiértékelésének a számával. Ebből egyszerűen következik, hogy a primál-duál teljes Newton-lépéses belsőpontos algoritmus $O(\sqrt{n}\,\lg\frac{n}{\varepsilon})$ iterációban előállít egy olyan $\hat{\mathbf{x}} \in \mathcal{P}_+$ és $(\hat{\mathbf{y}}, \hat{\mathbf{s}}) \in \mathcal{D}_+$ megoldást, amely esetén az $\hat{\mathbf{x}}^T\hat{\mathbf{s}} \le \varepsilon$ teljesül.

Rátérünk a primál-duál teljes Newton-lépéses belsőpontos algoritmus részletes elemzésére.

25.46. állítás. *Legyen* $\bar{\mathbf{x}} \in \mathcal{P}_+$, $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$, *és* $\mu > 0$. *Ha* $\partial = \partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}, \mu)$, *akkor*

$$\|\mathbf{d}_x\,\mathbf{d}_s\|_\infty \le \partial^2 \quad \text{és} \quad \|\mathbf{d}_x\,\mathbf{d}_s\| \le \partial^2\sqrt{2}\,.$$

Az előző állítás a 25.3-10. gyakorlatot felhasználva, $\mathbf{a} = \mathbf{d}_x$ és $\mathbf{b} = \mathbf{d}_s$ szereposztással, valamint figyelembe véve a $\mathbf{d}_x + \mathbf{d}_s = \mathbf{v}^{-1} - \mathbf{v}$ összefüggést, egyszerűen bizonyítható (lásd 25.3-11. gyakorlat).

A következőkben megmutatjuk, hogy ha a centrális út $\partial_1(\mathbf{x}, \mathbf{s}, \mu) < 1$ környezetében vagyunk, akkor a teljes Newton-lépés megengedett, illetve az így kapott pont is ebben a környezetben lesz.

25.47. tétel. *Ha* $\partial = \partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \mu) \le 1$, *akkor* $\mathbf{x}^+ \in \mathcal{P}$, $(\mathbf{y}^+, \mathbf{s}^+) \in \mathcal{D}$. *Továbbá, ha* $\partial < 1$, *akkor* $\mathbf{x}^+ > \mathbf{0}$, $\mathbf{s}^+ > \mathbf{0}$, *és*

$$\partial_1(\mathbf{x}^+, \mathbf{s}^+; \mu) \le \frac{\partial^2}{\sqrt{2\,(1-\partial^2)}}\,.$$

Bizonyítás. Az $\mathbf{x}^+$, és $\mathbf{s}^+$ pontosan akkor megengedett, ha $\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s \ge \mathbf{0}$, aminek elégséges feltétele $\|\mathbf{d}_x\,\mathbf{d}_s\|_\infty \le 1$. Legyen $\mathbf{v}^+ = \sqrt{\mathbf{x}^+\,\mathbf{s}^+/\mu}$, ekkor $(\mathbf{v}^+)^2 = \mathbf{x}^+\,\mathbf{s}^+/\mu = (\mathbf{e} + \mathbf{d}_x\,\mathbf{d}_s)$. Másfelől

$$\begin{aligned} 2\,\partial^+ &= \left\|(\mathbf{v}^+)^{-1} - \mathbf{v}^+\right\| = \left\|(\mathbf{v}^+)^{-1}(\mathbf{e} - (\mathbf{v}^+)^2)\right\| = \left\|(\mathbf{v}^+)^{-1}\,(\mathbf{e} - \mathbf{e} - \mathbf{d}_x\,\mathbf{d}_s)\right\| \\ &= \left\|\frac{\mathbf{d}_x\,\mathbf{d}_s}{\sqrt{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s}}\right\| \le \frac{\|\mathbf{d}_x\,\mathbf{d}_s\|}{\sqrt{1-\|\mathbf{d}_x\,\mathbf{d}_s\|_\infty}} \le \frac{\sqrt{2}\,\partial^2}{\sqrt{1-\partial^2}}\,, \end{aligned}$$

és így teljesül a

$$\partial^+ \le \frac{\partial^2}{\sqrt{2\,(1-\partial^2)}}$$

egyenlőtlenség. ∎

Világos, hogy ha $\partial \le 1/\sqrt{2}$, akkor a tétel szerint $\partial^+ = \partial_1(\mathbf{x}^+, \mathbf{s}^+; \mu) \le \partial^2$, azaz ez esetben az algoritmus lokálisan kvadratikusan konvergens. Ez ad részben magyarázatot arra is, hogy miért választottuk a $\tau = \frac{1}{\sqrt{2}}$ értéket a teljes Newton-lépéses algoritmus esetén.

A 25.3-14. gyakorlat eredményét felhasználva pontosabb felső korlátot tudunk adni új pontunk centralitási mértékére.

25.48. tétel. *Ha* $\partial = \partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}, \mu) < 1$ *, akkor*

$$\partial_1(\mathbf{x}^+, \mathbf{s}^+, \mu) \leq \frac{\partial^2}{\sqrt{2\,(1-\partial^4)}}\,.$$

Bizonyítás. Legyen $\partial^+ = \partial_1(\mathbf{x}^+, \mathbf{s}^+, \mu)$, $\mathbf{x}^+ = \bar{\mathbf{x}} + \Delta\mathbf{x}$, $\mathbf{s}^+ = \bar{\mathbf{s}} + \Delta\mathbf{s}$ és $\mathbf{v}^+ = \sqrt{\frac{\mathbf{x}^+\,\mathbf{s}^+}{\mu}} = \sqrt{\mathbf{e} + \mathbf{d}_x\,\mathbf{d}_s}$. Ekkor a $\mathbf{d}_x^T\mathbf{d}_s = 0$ összefüggést is figyelembe véve

$$\begin{aligned}\left\|\mathbf{v}^+\right\|^2 &= \mathbf{e}^T(\mathbf{v}^+\,\mathbf{v}^+) = \mathbf{e}^T(\mathbf{v}^+)^2 = \mathbf{e}^T\left(\sqrt{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s}\right)^2 = \mathbf{e}^T(\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s)\\ &= \mathbf{e}^T\mathbf{e} + \mathbf{d}_x^T\mathbf{d}_s = n\end{aligned}$$

adódik. Itt Ling 2. lemmáját (25.3-14. gyakorlat) alkalmazva, azt kapjuk, hogy

$$\begin{aligned}4\,(\partial^+)^2 &= \left\|(\mathbf{v}^+)^{-1} - \mathbf{v}^+\right\|^2 = \left((\mathbf{v}^+)^{-1} - \mathbf{v}^+\right)^T\left((\mathbf{v}^+)^{-1} - \mathbf{v}^+\right) =\\ &= \left\|(\mathbf{v}^+)^{-1}\right\|^2 + \left\|\mathbf{v}^+\right\|^2 - 2\,n = \left\|(\mathbf{v}^+)^{-1}\right\|^2 - n = \left\|\frac{\mathbf{e}}{\sqrt{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s}}\right\|^2 - n =\\ &= \mathbf{e}^T\left(\frac{\mathbf{e}}{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s} - \mathbf{e}\right)\,.\end{aligned}$$

A 25.3-14. gyakorlatot a $\mathbf{u} = \mathbf{d}_x$ és $\mathbf{v} = \mathbf{d}_s$ vektorokkal alkalmazva

$$4\,(\partial^+)^2 \leq \frac{2\partial^4}{1-\partial^4}$$

adódik, ugyanis $\|\mathbf{d}_x + \mathbf{d}_s\| = 2\partial$ a centralitási mérték definíciója szerint, ahol $\partial < 1$. Átrendezés után pedig megkapjuk a tétel állítását. ■

A 25.47., valamint a 25.48. tételekben beláttuk, hogy a centrális út megfelelő környezetéből indulva a teljes Newton-lépés sem vezet ki a környezetből, tehát a teljes Newton-lépéses algoritmus megfelelő τ (például $\tau < 1$) esetén jól definiált lesz.

Az algoritmus lépésszámának meghatározása érdekében először vizsgáljuk meg, hogy hogyan változik a centralitási mérték a μ paraméter változtatása következtében. A következő lemma bizonyítását az Olvasóra bízzuk (lásd 25.3-16. gyakorlat).

25.49. lemma. *Legyen* $\mathbf{x} \in \mathcal{P}_+$, $(\mathbf{y}, \mathbf{s}) \in \mathcal{D}_+$ *és* $\mu > 0$, *amelyre* $\mathbf{x}^T\mathbf{s} = \mu\,n$. *Továbbá legyen* $\partial = \partial_1(\mathbf{x}, \mathbf{s}, \mu)$ *és* $\mu^+ = (1-\theta)\,\mu$. *Ekkor*

$$\partial_1(\mathbf{x}, \mathbf{s}, \mu^+)^2 = (1-\theta)\,\partial^2 + \frac{\theta^2 n}{4\,(1-\theta)}\,.$$

Az előző eredmény ismeretében már könnyen bizonyítható, hogy a Primál-duál-Newton-lépéses-belsőpontos algoritmus esetén a belső ciklusban pontosan egy iterációra kerül sor, ami egy teljes Newton-lépésnek felel meg.

25.50. tétel. *Legyen* $\tau = 1/\sqrt{2}$ *és* $\theta = 1/(2\sqrt{n})$, *akkor a* Primál-duál-Newton-lépéses-belsőpontos *algoritmus pontosan egy teljes Newton-lépést tesz a belső ciklusban.*

Bizonyítás. Legyen $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{P}_+ \times \mathcal{D}_+$ olyan megoldás, amelynek a távolsága a μ-centrumától $\partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \bar{\mu}) \leq \tau = 1/\sqrt{2}$. Ekkor a primál-duál Newton-lépéses belsőpontos algoritmus alapján $\mu^+ = (1-\theta)\,\bar{\mu}$ lesz az új μ-centrum paramétere (külső ciklus). Az algoritmust alkalmazva tegyünk egy teljes Newton-lépést a célul kitűzött, új centrum felé (belső ciklus). Ekkor az $(\mathbf{x}^+, \mathbf{y}^+, \mathbf{s}^+) \in \mathcal{P}_+ \times \mathcal{D}_+$ pontot kapjuk, amely megengedettségét a 25.47. tétel biztosítja. A 25.48. tétel alapján az új pont és a régi centrum távolsága a ∂_1 centralitási mérték szerint

$$\partial_1(\mathbf{x}^+, \mathbf{s}^+, \mu) \leq \frac{\partial^2}{\sqrt{2\,(1-\partial^4)}} \leq \frac{\tau^2}{\sqrt{2\,(1-\tau^4)}} = \frac{\frac{1}{2}}{\sqrt{\frac{3}{2}}} < \frac{1}{2}$$

lesz. Továbbá $(\mathbf{x}^+)^T \mathbf{s}^+ = \mu^+ n$ a 25.43. lemma alapján. A 25.49. lemmát felhasználva

$$\begin{aligned}
\partial_1(\mathbf{x}^+, \mathbf{s}^+, \mu^+)^2 &= (1-\theta)\,\partial_1(\mathbf{x}^+, \mathbf{s}^+, \mu)^2 + \frac{n\,\theta^2}{4(1-\theta)} < \frac{1-\theta}{4} + \frac{n\,\theta^2}{4(1-\theta)} \\
&= \frac{1-\theta}{4} + \frac{n\left(\frac{1}{4\,n}\right)}{4\,(1-\theta)} = \frac{1-\theta}{4} + \frac{1}{16\,(1-\theta)} \\
&\leq \frac{1}{4} + \frac{1}{8} = \frac{3}{8} < \frac{1}{2} = \tau^2
\end{aligned}$$

adódik, figyelembe véve a $0 < \theta = 1/(2\sqrt{n}) \leq 1/2$ összefüggést, amely bármely n esetén teljesül. Az iteráció után az új pont ismét a centrális út megfelelő környezetében van, tehát nincsen szükség további iterációra a belső ciklusban. ■

25.3.4. Primál-duál prediktor-korrektor belsőpontos algoritmus

Az előző részben ismertetett primál-duál algoritmust továbbfejlesztve jutunk el a ***prediktor-korrektor algoritmushoz.*** A bevezetett jelöléseket használjuk a továbbiakban is. Mint már láttuk, a $\mathbf{d}_x$ és $\mathbf{d}_s$ vektorok merőlegesek egymásra, valamint $\mathbf{d}_x$ az AD, míg $\mathbf{d}_s$ a HD^{-1} mátrixok nullterére vett vetülete a $\mathbf{v}^{-1} - \mathbf{v}$ vektornak (lásd (25.18)).

Bontsuk szét a $\mathbf{d}_x$ és $\mathbf{d}_s$ vektorokat egy ún. affin skálázási és egy centralizáló rész összegére a következőképpen:

$$\begin{array}{lllllll}
\textbf{\textit{centralizáló lépés:}} & \mathbf{d}_x^c & = P_{AD}(\mathbf{v}^{-1}) & \text{és} & \mathbf{d}_s^c & = P_{HD^{-1}}(\mathbf{v}^{-1}), \\
\textbf{\textit{affin skálázási lépés:}} & \mathbf{d}_x^a & = P_{AD}(-\mathbf{v}) & \text{és} & \mathbf{d}_s^a & = P_{HD^{-1}}(-\mathbf{v}).
\end{array}$$

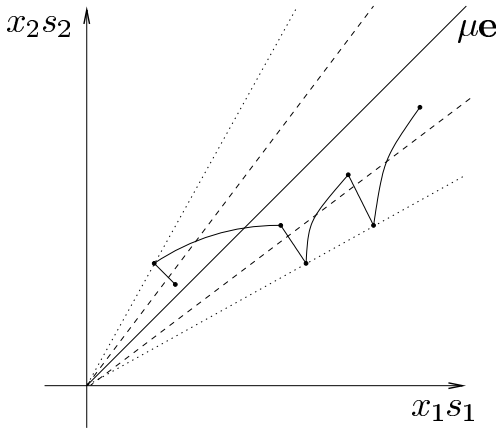
Azaz

$$\mathbf{d}_x = \mathbf{d}_x^c + \mathbf{d}_x^a \qquad \text{és} \qquad \mathbf{d}_s = \mathbf{d}_s^c + \mathbf{d}_s^a,$$

továbbá a $\mathbf{v}^{-1}$ és $\mathbf{v}$ vektorok felbontását kapjuk a következő értelemben

$$\mathbf{d}_x^c + \mathbf{d}_s^c = \mathbf{v}^{-1}, \tag{25.20}$$

$$\mathbf{d}_x^a + \mathbf{d}_s^a = -\mathbf{v}. \tag{25.21}$$



25.6. ábra. Prediktor-korrektor algoritmus iterációi az átskálázott térben, ahol $x_1s_1 = \mu v_1^2$ és $x_2s_2 = \mu v_2^2$.

Az eredeti skálázással analóg módon definiáljuk az affin skálázási, illetve a centralizáló lépésben a Newton-irányokat:

$$\Delta^a\mathbf{x} = \sqrt{\mu}\,\mathbf{d}\,\mathbf{d}_x^a, \qquad \Delta^a\mathbf{s} = \sqrt{\mu}\,\mathbf{d}^{-1}\mathbf{d}_s^a\ ,$$

$$\Delta^c\mathbf{x} = \sqrt{\mu}\,\mathbf{d}\,\mathbf{d}_x^c, \qquad \Delta^c\mathbf{s} = \sqrt{\mu}\,\mathbf{d}^{-1}\mathbf{d}_s^c\ .$$

Az affin skálázási lépésben α hosszúságú Newton-lépést teszünk meg, míg a centralizáló lépésben meglépjük a teljes Newton-lépést, így az $(\mathbf{x},\mathbf{s})$ pontból a következő pontokba lépünk:

$$\mathbf{x}^a(\alpha) = \mathbf{x} + \alpha\Delta^a\mathbf{x}, \qquad \mathbf{s}^a(\alpha) = \mathbf{s} + \alpha\Delta^a\mathbf{s}\ ,$$

$$\mathbf{x}^c = \mathbf{x} + \Delta^c\mathbf{x}, \qquad \mathbf{s}^c = \mathbf{s} + \Delta^c\mathbf{s}\ .$$

A fenti azonosságok alapján könnyen beláthatóak az alábbi összefüggések az affin skálázási, illetve a centralizáló lépésekre:

$$\mathbf{x}\,\Delta^a\mathbf{s} + \mathbf{s}\,\Delta^a\mathbf{x} = -\mathbf{x}\,\mathbf{s}\ , \tag{25.22}$$

$$\mathbf{x}\,\Delta^c\mathbf{s} + \mathbf{s}\,\Delta^c\mathbf{x} = \mu\,\mathbf{e}\ , \tag{25.23}$$

ugyanis $\mathbf{x}\Delta^a\mathbf{s} = \sqrt{\mu}\,\mathbf{x}\,\mathbf{d}^{-1}\mathbf{d}_s^a = \mu\,\mathbf{v}\,\mathbf{d}_s^a$, valamint $\mathbf{s}\Delta^a\mathbf{x} = \sqrt{\mu}\,\mathbf{s}\,\mathbf{d}\,\mathbf{d}_x^a = \mu\,\mathbf{v}\,\mathbf{d}_x^a$. Az utóbbi két egyenlőséget összeadva, és figyelembe véve (25.21) összefüggést, a (25.22) egyenlőséget kapjuk

$$\mathbf{x}\,\Delta^a\mathbf{s} + \mathbf{s}\,\Delta^a\mathbf{x} = \mu\,\mathbf{v}\,\mathbf{d}_s^a + \mu\,\mathbf{v}\,\mathbf{d}_x^a = \mu\,\mathbf{v}(-\mathbf{v}) = -\mathbf{x}\,\mathbf{s}\ .$$

A centralizáló lépésre vonatkozó (25.23) összefüggést hasonlóan bizonyíthatjuk felhasználva a definíciókat és a (25.20) egyenlőséget (lásd 25.3-17. gyakorlat).

25.51. lemma. *Legyen $\mathbf{x}^T\mathbf{s} = n\mu$. Tegyük fel, hogy megengedett lépéseket teszünk, ekkor a α hosszúságú affin skálázási lépésben $(1-\alpha)$-szorosára csökken a dualitásrés, míg a centralizáló lépés során amennyiben megengedett, megduplázódik.*

Bizonyítás. Először vizsgáljuk meg az affin skálázási lépés esetén a dualitásrést:

$$\mathbf{x}^a(\alpha)\,\mathbf{s}^a(\alpha) = (\mathbf{x}+\alpha\Delta^a\mathbf{x})(\mathbf{s}+\alpha\Delta^a\mathbf{s}) = \mathbf{x}\,\mathbf{s}+\alpha(\mathbf{x}\,\Delta^a\mathbf{s}+\mathbf{s}\,\Delta^a\mathbf{x})+\alpha^2\Delta^a\mathbf{x}\,\Delta^a\mathbf{s}\ .$$

A (25.22) egyenlőséget felhasználva

$$\mathbf{x}^a(\alpha)\,\mathbf{s}^a(\alpha) = (1-\alpha)\,\mathbf{x}\,\mathbf{s}+\alpha^2\Delta^a\mathbf{x}\,\Delta^a\mathbf{s} \tag{25.24}$$

adódik. Mivel $\Delta^a\mathbf{x}$ és $\Delta^a\mathbf{s}$ merőlegesek egymásra (ugyanis $\mathbf{d}_x^a$ és $\mathbf{d}_s^a$ vektorok merőlegesek a 25.41. lemma bizonyításához hasonló megfontolások alapján), a következőt kapjuk:

$$\mathbf{x}^a(\alpha)^T\mathbf{s}^a(\alpha) = \mathbf{e}^T((1-\alpha)\mathbf{x}\,\mathbf{s}+\alpha^2\Delta^a\mathbf{x}\,\Delta^a\mathbf{s}) = (1-\alpha)\mathbf{x}^T\mathbf{s}\ ,$$

amivel beláttuk az első állításunkat. A centralizáló lépésnél hasonlóan járunk el:

$$\mathbf{x}^c\,\mathbf{s}^c = (\mathbf{x}+\Delta^c\mathbf{x})(\mathbf{s}+\Delta^c\mathbf{s}) = \mathbf{x}\,\mathbf{s}+(\mathbf{x}\,\Delta^c\mathbf{s}+\mathbf{s}\,\Delta^c\mathbf{x})+\Delta^c\mathbf{x}\,\Delta^c\mathbf{s}\ .$$

A (25.23) összefüggést figyelembe véve az

$$\mathbf{x}^c\,\mathbf{s}^c = \mathbf{x}\,\mathbf{s}+\mu\,\mathbf{e}+\Delta^c\mathbf{x}\,\Delta^c\mathbf{s}$$

egyenlőséget kapjuk. Az előzőekhez hasonlóan az $\Delta^c\mathbf{x}$ és $\Delta^c\mathbf{s}$ vektorok is merőlegesek egymásra, így

$$(\mathbf{x}^c)^T\mathbf{s}^c = \mathbf{e}^T(\mathbf{x}\,\mathbf{s}+\mu\,\mathbf{e}+\Delta^c\mathbf{x}\,\Delta^c\mathbf{s}) = \mathbf{x}^T\mathbf{s}+\mu\,\mathbf{e}^T\mathbf{e} = 2n\mu\ ,$$

ezzel beláttuk a második állítást is. ■

Az előző lemma világossá teszi, hogy a dualitásrés csökkenését az affin skálázási lépéssel biztosítjuk, míg a fenti módon definiált centralizáló lépés következtében a dualitásrés megduplázódna, ezzel elrontva az algoritmus konvergenciáját, ezért ehelyett egy teljes Newton-lépést teszünk meg az aktuális μ érték felé. Ennek következtében, mint már az előző részben is láttuk (25.43. lemma), a pontunk dualitásrése változatlan marad, de közelebb kerülünk a centrális úthoz. Ezen tulajdonsága miatt nevezzük ezt a lépést centralizáló lépésnek. Ennek megfelelően az affin lépést prediktor, míg a centralizáló lépést korrektor lépésnek is nevezik.

Célunk olyan megoldáspár előállítása, melynek a dualitásrése közel nulla, azaz a dualitásrést szeretnénk minél gyorsabban csökkenteni, vagyis az affin skálázási lépés hatékonysága fontos kérdése az algoritmusnak. Ezen a gondolaton alapszik a prediktor-korrektor algoritmus. Megjegyezzük, hogy ha az affin skálázási lépés esetén a teljes Newton-lépés megengedett, akkor e lépéssel egy nulla dualitásrésű ponthoz jutunk, azaz egy optimális megoldáshoz, ám ez az esetek többségében nem következik be.

A prediktor-korrektor algoritmus esetén ahelyett, hogy az affin skálázási és a centralizáló lépésből egy Newton-lépést készítenénk, az előző primál-duál algoritmusunk Newton-lépését két részre szedjük szét. Először egy centralizáló lépést, majd egy affin skálázási lépést teszünk. Mivel a teljes affin lépés nem megengedett, egy α paraméterrel tompítjuk. Az α megfelelő megválasztásával egyrészt biztosítjuk a lépésünk megengedettségét, másrészt korlátozzuk az új pontunk túlzott eltávolodását a centrális úttól. A centralizáló lépésünk célja visszajutni a centrális út megfelelő környezetébe, ezzel biztosítva az algoritmus konvergenciáját.

Az algoritmus működését a 25.6. ábra szemlélteti, ahol a szaggatott vonal a szűkebb, míg a pontozott a tágabb környezete a centrális útnak. Mint ahogy az ábrán is látható, egy affin lépés során eltávolodunk a centrális úttól, de nem hagyjuk el a tágabb környezetet, míg a centralizáló lépés során visszajutunk a szűkebb környezetbe.

A prediktor-korrektor algoritmus bemenő adataként — hasonlóan a primál-duál Newton-lépéses belsőpontos algoritmushoz — meg kell adnunk az $\varepsilon > 0$ megkívánt számítási pontosságot, a $1 > \tau \geq 0$ eltérés paramétert, valamint egy $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0)$ belső pontot, amely a τ által meghatározott környezetben helyezkedik el, azaz $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0) \in \mathcal{P}_+ \times \mathcal{D}_+$, $(\mathbf{x}^0)^T\mathbf{s}^0 = \mu^0 n$ és $\partial_1(\mathbf{x}^0, \mathbf{s}^0, \mu^0) \leq \tau$.

PREDIKTOR-KORREKTOR-BELSŐPONTOS

```
1  x ← x^0
2  s ← s^0
3  μ ← μ^0
4  while nμ ≥ (1 − α)ε
5        do μ ← x^T s/n
6           számítsuk ki a Δx és Δs értékét a Newton rendszer (1265. oldal)
            megoldásával                          ▷ Korrektor lépés (Newton-lépés).
7           x ← x + Δx
8           s ← s + Δs
9           számítsuk ki a Δ^a x és Δ^a s értékét   ▷ Prediktor lépés (affin skálázási lépés).
10          határozzuk meg a maximális α lépéshosszt úgy, hogy
              ∂_1(x(α), s(α), x(α)^T s(α)/n) ≤ τ     (x(α) = x + αΔ^a x, s(α) = s + αΔ^a s)
11          x ← x + αΔ^a x
12          s ← s + αΔ^a s
13          μ ← (1 − α)μ
```

Affin skálázási lépés

A továbbiakban megvizsgáljuk, hogy egy affin skálázási lépés során hogyan változik a centralitási mérték a lépéshossz függvényében. Legyen $(\mathbf{x}, \mathbf{s})$ pozitív megengedett primál-duál pár. Legyen továbbá $\mu > 0$, melyre $\mathbf{x}^T\mathbf{s} = n\mu$. Elemzésünk során az előző részben definiált

$$\partial = \partial(\mathbf{x}, \mathbf{s}, \mu) = \frac{1}{2}\|\mathbf{v}^{-1} - \mathbf{v}\|, \text{ ahol } \quad \mathbf{v} = \sqrt{\frac{\mathbf{x}\,\mathbf{s}}{\mu}}\,,$$

centralitási mértéket használjuk. A számolások során szükségünk lesz a $\mathbf{v}$ vektor becslésére. A következő lemmában alsó és felső korlátot adunk a koordinátákra, a bizonyítást az Olvasóra bízzuk (lásd 25.3-18. gyakorlat).

25.52. lemma. *Legyen* $\rho(\partial) = \partial + \sqrt{1 + \partial^2}$. *Ekkor*

$$\frac{1}{\rho(\partial)} \leq v_i \leq \rho(\partial), \qquad 1 \leq i \leq n\,.$$

Az $\mathbf{v}$ vektorra adott korlátok ismeretében az α lépéshossz megengedettségére a következő elégséges feltételt kapjuk.

25.53. lemma. *Legyen* $(\mathbf{x},\mathbf{s})$ *megengedett pont, melyre* $\mathbf{x}^T\mathbf{s} = n\mu$ *és* $\partial = \partial(\mathbf{x},\mathbf{s},\mu) < \tau$. *Jelölje továbbá* $\mathbf{x}^+ = \mathbf{x} + \alpha\Delta^a\mathbf{x}$ *és* $\mathbf{s}^+ = \mathbf{s} + \alpha\Delta^a\mathbf{s}$ *az affin lépés után kapott pontot. Ekkor* $\partial^+ = \partial(\mathbf{x}^+,\mathbf{s}^+,(1-\alpha)\mu) \le \tau$, *ha az* α *paraméterre teljesül az alábbi egyenlőtlenség*

$$\frac{\alpha^2 n}{1-\alpha} \le 2\sqrt{2}\left(\tau\sqrt{\frac{4}{\rho(\partial)^2} + 4\partial\rho(\partial)\sqrt{2} + 2\tau^2} - 2\partial\rho(\partial) - \tau^2\sqrt{2}\right).$$

Továbbá rögzített τ *mellett a fenti kifejezés jobb oldala* ∂ *monoton csökkenő függvénye.*

Bizonyítás. A (25.24) egyenlőséget felhasználva a következőt kapjuk

$$\mathbf{x}^+\mathbf{s}^+ = (1-\alpha)\mathbf{x}\,\mathbf{s} + \alpha^2\Delta^a\mathbf{x}\,\Delta^a\mathbf{s} = \mu\left((1-\alpha)\mathbf{v}^2 + \alpha^2\mathbf{d}_x^a\,\mathbf{d}_s^a\right).$$

Vezessük be az alábbi jelölést

$$\mathbf{v}^+ = \sqrt{\frac{\mathbf{x}^+\mathbf{s}^+}{(1-\alpha)\mu}}, \text{ ekkor } \quad (\mathbf{v}^+)^2 = \mathbf{v}^2 + \frac{\alpha^2\mathbf{d}_x^a\,\mathbf{d}_s^a}{1-\alpha}.$$

Az új pontunk centralitási mértékére

$$\partial^+ = \frac{1}{2}\left\|(\mathbf{v}^+)^{-1}\left(\mathbf{e}-\mathbf{v}^+\right)\right\| \le \frac{1}{2}\left\|(\mathbf{v}^+)^{-1}\right\|_\infty\left\|\mathbf{e}-(\mathbf{v}^+)^2\right\|$$

áll fenn. A továbbiakban külön-külön felső korlátot adunk a jobb oldali szorzat két tényezőjére, először a második kifejezést vizsgáljuk.

$$\begin{aligned}\left\|\mathbf{e}-(\mathbf{v}^+)^2\right\| &= \left\|\mathbf{e}-\mathbf{v}^2-\frac{\alpha^2\mathbf{d}_x^a\,\mathbf{d}_s^a}{1-\alpha}\right\| \le \left\|\mathbf{e}-\mathbf{v}^2\right\| + \frac{\alpha^2}{1-\alpha}\left\|\mathbf{d}_x^a\,\mathbf{d}_s^a\right\| \\ &\le \left\|\mathbf{e}-\mathbf{v}^2\right\| + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}.\end{aligned}$$

Az utolsó egyenlőtlenség a 25.46. következmény miatt igaz, figyelembe véve az $\left\|\mathbf{d}_x^a + \mathbf{d}_s^a\right\|^2 = \|\mathbf{v}\|^2 = n$ egyenlőséget. A 25.52. lemmabeli eredmények segítségével kapjuk az alábbiakat

$$\left\|\mathbf{e}-\mathbf{v}^2\right\| = \left\|\mathbf{v}\,\frac{\mathbf{e}-\mathbf{v}^2}{\mathbf{v}}\right\| \le \|\mathbf{v}\|_\infty\left\|\frac{\mathbf{e}-\mathbf{v}^2}{\mathbf{v}}\right\| \le 2\partial\rho(\partial).$$

Tehát a második tényezőre a következő egyenlőtlenség teljesül:

$$\left\|\mathbf{e}-(\mathbf{v}^+)^2\right\| \le 2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}.$$

Az első tényező becsléséhez ismét használjuk fel a 25.52. lemmát és a 25.46. következményt.

$$(v_i^+)^2 \ge v_i^2 - \frac{\alpha^2}{1-\alpha}\left\|\mathbf{d}_x^a\,\mathbf{d}_s^a\right\|_\infty \ge \frac{1}{\rho(\partial)^2} - \frac{\alpha^2 n}{4\,(1-\alpha)},$$

azaz

$$\left\|(\mathbf{v}^+)^{-1}\right\|_\infty \leq \frac{1}{\sqrt{\frac{1}{\rho(\partial)^2} - \frac{\alpha^2 n}{4\,(1-\alpha)}}}\,.$$

A kapott felső korlátokat behelyettesítve a következőre jutunk

$$\partial^+ \leq \frac{2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}}{2\,\sqrt{\frac{1}{\rho(\partial)^2} - \frac{\alpha^2 n}{4\,(1-\alpha)}}}\,.$$

A fentiek alapján a $\partial^+ \leq \tau$ fennállásának elégséges feltétele

$$\left(2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}\right)^2 \leq \frac{4\,\tau^2}{\rho(\partial)^2} - \frac{\alpha^2 n \tau^2}{1-\alpha}\,, \tag{25.25}$$

vigyük át a jobb oldal második tagját a bal oldalra, majd adjunk hozzá $4\sqrt{2}\tau^2\partial\rho(\partial)$-t mindkét oldalhoz. Átrendezések után

$$\left(2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}\right)^2 + 2\,\tau^2\sqrt{2}\left(2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}\right) \leq \frac{4\,\tau^2}{\rho(\partial)^2} + 4\,\tau^2\partial\rho(\partial)\sqrt{2}\,,$$

$$\left(2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)} + \tau^2\sqrt{2}\right)^2 \leq \frac{4\,\tau^2}{\rho(\partial)^2} + 4\,\tau^2\partial\rho(\partial)\sqrt{2} + 2\,\tau^4\,.$$

Mindkét oldalból gyököt vonva

$$2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}(1-\alpha)} + \tau^2\sqrt{2} \leq \tau\sqrt{\frac{4}{\rho(\partial)^2} + 4\partial\rho(\partial)\sqrt{2} + 2\,\tau^2}\,,$$

és α paraméterre rendezve

$$\frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)} \leq \tau\sqrt{\frac{4}{\rho(\partial)^2} + 4\partial\rho(\partial)\sqrt{2} + 2\,\tau^2} - 2\partial\rho(\partial) - \tau^2\sqrt{2}\,,$$

ami a lemma első állítása. A második állítás bizonyításához vegyük észre, hogy a lemmabeli egyenlőtlenség ekvivalens a (25.25) egyenlőtlenséggel, így elegendő az utóbbit vizsgálni. Könnyen belátható, hogy a (25.25) egyenlőtlenség bal oldali kifejezése mind α, mind ∂ változóban monoton nő, míg a jobb oldali kifejezés mindkettő szerint monoton fogy. Azaz ha α kielégíti az egyenlőtlenséget valamely ∂ érték mellett, akkor ugyanezen α kielégíti kisebb ∂ értékre is. Mivel az előbb említett két egyenlőtlenség ekvivalens, ugyanez igaz a lemmabeli egyenlőtlenségre is, ezzel a lemma második állítását is beláttuk. ■

Megmutatható, hogy ha $\tau = 1/2$ és $\alpha = 1/(2\sqrt{n})$, akkor minden iteráció után olyan $(\mathbf{x}, \mathbf{s})$ pontba érkezünk, melyre $\partial(\mathbf{x}, \mathbf{s}, \mu) \leq \tau$, azaz az algoritmus jól definiált (lásd 25.3-19. gyakorlat).

Ezek után megfogalmazhatjuk fő tételünket, mely a PREDIKTOR-KORREKTOR-BELSŐPONTOS-ALGORITMUS lépésszámát határozza meg. A bizonyítás a PRIMÁL-DUÁL-NEWTON-LÉPÉSES-BELSŐPONTOS algoritmus lépésszám bizonyításához (25.50. tétel) teljesen hasonlóan megy.

25.54. tétel. *Legyen* $\tau = 1/2$ *és* $\alpha = 1/(2\sqrt{n})$, *ekkor a* PREDIKTOR-KORREKTOR-BELSŐPONTOS *algoritmus legfeljebb*

$$\left\lceil 2\sqrt{n}\lg\frac{n\mu^0}{\varepsilon}\right\rceil$$

lépésben állít elő egy olyan $\mathbf{x}\in\mathcal{P}_+$, $(\mathbf{y},\mathbf{s})\in\mathcal{D}_+$ *primál-duál pontpárt, amelyre* $\mathbf{x}^T\mathbf{s}\le\varepsilon$.

A prediktor-korrektor algoritmus adaptív változata

A PREDIKTOR-KORREKTOR-BELSŐPONTOS algoritmus lépészámát tovább csökkenthetjük, ha a lépéshossz megválasztási stratégiáját megváltoztatjuk. A fenti algoritmusban az α értéke egy fix szám. Ezzel szemben, ha az algoritmus során a pontunktól függően változtatjuk a α értékét, azaz a dualitásrés csökkentése mértékét, akkor egy hatékonyabb, gyorsabb algoritmust kapunk.

Először α megengedett értékére adunk egy megfelelőbb korlátot a 25.53. lemma eredményének a javításával. A bizonyítás is az említett lemmához hasonlóan történik, így az Olvasóra bízzuk (lásd 25.3-20. gyakorlat).

25.55. lemma. *Legyen* $(\mathbf{x},\mathbf{s})$ *megengedett pont, melyre* $\mathbf{x}^T\mathbf{s}=n\mu$ *és* $\partial=\partial(\mathbf{x},\mathbf{s},\mu)<\tau$. *Jelölje továbbá* $\mathbf{x}^+=\mathbf{x}+\alpha\Delta^a\mathbf{x}$ *és* $\mathbf{s}^+=\mathbf{s}+\alpha\Delta^a\mathbf{s}$ *az affin lépés után kapott pontot. Ekkor* $\partial^+=\partial(\mathbf{x}^+,\mathbf{s}^+,(1-\alpha)\mu)\le\tau$, *ha az* α *lépéshosszra teljesül az alábbi egyenlőtlenség*

$$\frac{\alpha^2}{1-\alpha}\left\|\mathbf{d}_x^a\,\mathbf{d}_s^a\right\|\le 2\tau\left(\sqrt{\frac{1}{\rho(\partial)^2}+2\partial\rho(\partial)+\tau^2}-\tau\right)-2\partial\rho(\partial)\ .$$

Az adaptív algoritmus esetén is létezik megfelelő τ, illetve α értékek, melyekre az eljárás jól definiált lesz, mégpedig $\tau=1/3$ és $\alpha=2/(1+\sqrt{1+13\,\|\mathbf{d}_x^a\,\mathbf{d}_s^a\|})$ választás eleget tesz elvárásainknak (lásd 25.3-21. gyakorlat).

Kvadratikus konvergencia

Könnyen látható, hogy a konvergencia rendje α paraméter választásától függ. A kvadratikus konvergencia feltétele

$$(1-\alpha)\mu=O(\mu^2),\qquad\text{azaz}\qquad 1-\alpha=O(\mu)\ .$$

Megmutatjuk, hogy az előző adaptív algoritmus estén megadott α érték teljesíti ezt a feltételt, amikor a μ elegendően kicsi, azaz a fent tárgyalt adaptív prediktor-korrektor algoritmus konvergenciája kvadratikus. Ehhez először az $1-\alpha$ kifejezésre adunk felső becslést, melynek bizonyítását az Olvasóra bízzuk (lásd 25.3-22. gyakorlat).

25.56. lemma. *A lépéshossz* $\alpha=2/(1+\sqrt{1+13\,\|\mathbf{d}_x^a\mathbf{d}_s^a\|})$ *választása esetén*

$$1-\alpha\le\frac{13}{4}\left\|\mathbf{d}_x^a\,\mathbf{d}_s^a\right\|\ .$$

Sorszm	Vektor	$\mathcal{B}$	$\mathcal{N}$
1	$\mathbf{x}$	$\Theta(1)$	$\Theta(\mu)$
2	$\mathbf{s}$	$\Theta(\mu)$	$\Theta(1)$
3	$\mathbf{v}$	$\Theta(1)$	$\Theta(1)$
4	$\mathbf{d}$	$\Theta(1/\sqrt{\mu})$	$\Theta(\sqrt{\mu})$
5	$\mathbf{d}_x^a$	$O(\mu)$	$O(1)$
6	$\mathbf{d}_s^a$	$O(1)$	$O(\mu)$
7	$\Delta^a\mathbf{x}$	$O(\mu)$	$O(\mu)$
8	$\Delta^a\mathbf{s}$	$O(\mu)$	$O(\mu)$

25.1. táblázat. A változók mérete, amikor μ elegendően kicsi.

Azaz az adaptív prediktor-korrektor algoritmus konvergenciája kvadratikus, ha $\left\|\mathbf{d}_x^a\mathbf{d}_s^a\right\| = O(\mu)$ *teljesül.*

A 25.1 táblázatban, amikor μ elegendően kicsi, összefoglaljuk a bevezetett vektorok koordinátáinak nagyságrendjét aszerint, hogy az indexük a $\mathcal{B}$ vagy az $\mathcal{N}$ halmazba esik. (A bizonyítástól technikai jellege miatt eltekintünk.)

Ezek után megfogalmazhatjuk alfejezetünk fő tételét

25.57. tétel. *Az adaptív prediktor-korrektor algoritmus kvadratikusan konvergens.*

Bizonyítás. Az előzőekben belátottak alapján $\mathbf{d}_x^a\,\mathbf{d}_s^a = O(\mu)$, ez pedig a 25.56. lemma szerint pontosan azt jelenti, hogy az algoritmus kvadratikusan konvergens. ■

25.3.5. Önreguláris függvényen alapuló belső pontos algoritmus

A következőkben ismertetett elmélet kiindulópontja az előző rész bevezetőjében említett ellentmondás, miszerint a rövid lépéses algoritmusok $O(\sqrt{n}\lg\frac{n}{\varepsilon})$ bizonyított lépésszáma jobb, mint a hosszú lépéses algoritmusok esetén ismert $O(n\lg\frac{n}{\varepsilon})$, ezzel szemben a gyakorlatban a hosszú lépéses módszerek hatékonyabbnak bizonyultak. Peng, Roos és Terlaky ennek az ellentmondásnak a feloldása céljából a hosszú lépéses eljárásokat új centralitási mérték bevezetése mellett vizsgálta meg. Az új mértékeket az önreguláris függvények[20] segítségével definiálták. A megváltoztatott centralitási mértéknek megfelelően a Newton-rendszert is módosították. Az így kapott új algoritmus tekinthető az előző részben bevezetett ∂_1 centralitási mérték, illetve a hozzá tartozó Newton-rendszer általánosításának. Ebben a részben az önreguláris függvények tulajdonságaira épülő belsőpontos algoritmusok egy speciális esetét ismertetjük Peng, Roos és Terlaky egyik cikke alapján, melyben a módosított belsőpontos algoritmus lépésszáma $O(\sqrt{n}\lg n\lg(\frac{n}{\varepsilon}))$, amely a rövid lépéses algoritmusok lépésszámát igen jól megközelíti.

25.58. definíció. *Egy* $\psi : (0,\infty) \to \mathbb{R}$ *kétszer folytonosan deriválható függvényt* ***önreguláris függvénynek*** *nevezünk, ha teljesíti a következő feltételeket:*

[20] Az önreguláris függvény elnevezés a függvénycsalád azon tulajdonságából származik, hogy – mint látni fogjuk – az első és második deriváltak egymást korlátozzák, azaz a függvény egyféleképpen regularizálja önmagát.

1. *$\psi(t)$ szigorúan konvex függvény és a globális minimumpontjában, $t = 1$ pontban eltűnik, azaz $\psi(1) = \psi'(1) = 0$. Továbbá léteznek $\nu_2 \geq \nu_1 > 0$ és $p \geq 1$, $q \geq 1$ konstansok úgy, hogy*

$$\nu_1(t^{p-1} + t^{-1-q}) \leq \psi''(t) \leq \nu_2(t^{p-1} + t^{-1-q}) , \quad \forall t \in (0, \infty) .$$

2. *Bármely $t_1, t_2 > 0$ esetén*

$$\psi(t_1^r t_2^{1-r}) \leq r\psi(t_1) + (1-r)\psi(t_2) , \quad \forall r \in [0, 1] .$$

Ha a ψ önreguláris függvény, akkor a q paramétert a ***büntetés mértékének*** (a szám nullához tartását büntetjük), míg a p paramétert a ***növekedés mértékének*** nevezzük.

Az önreguláris függvények két speciális családját emelnénk ki. Az első családot a

$$\Upsilon_{p,q}(t) = \frac{t^{p+1} - 1}{p(p+1)} + \frac{t^{1-q} - 1}{q(q-1)} + \frac{p-q}{pq}(t-1) , \quad \text{ahol } p \geq 1 , q > 1 ,$$

függvény definiálja. Ekkor a definícióban szereplő konstansok $\nu_1 = \nu_2 = 1$. A második család a

$$\Gamma_{p,q}(t) = \frac{t^{p+1} - 1}{p+1} + \frac{t^{1-q} - 1}{q-1} , \quad \text{ahol } p \geq 1 , q > 1 ,$$

függvény által definiált, és a konstansok értéke $\nu_1 = 1$, illetve $\nu_2 = q$.
Ezen függvények segítségével definiáljuk az eljárás során használt távolságfüggvényt

$$\Psi(\mathbf{v}) = \sum_{i=1}^{n} \psi(v_i)$$

egyenlőséggel.

Ebben a részben a második családba tartozó

$$\psi(t) = \Gamma_{1,q}(t) = \frac{t^2 - 1}{2} + \frac{t^{1-q} - 1}{q-1} , \quad q > 1$$

önreguláris függvénycsaládot használjuk. Ekkor a távolságfüggvény

$$\Psi_q(\mathbf{x}, \mathbf{s}, \mu) = \Psi_q(\mathbf{v}) = \frac{\mathbf{e}^T\mathbf{v}^2 - n}{2} + \frac{\mathbf{e}^T\mathbf{v}^{1-q} - n}{q-1} .$$

A Ψ függvény első tagja a $\mathbf{v}$ vektor koordinátáinak növekedését, míg a második tag a nullához tartásukat bünteti. Jelölje a második tagot

$$\Psi_0(\mathbf{v}) = \sum_{i=1}^{n} \psi_0(v_i), \text{ ahol} \qquad \psi_0(t) = \frac{t^{1-q} - 1}{q-1} .$$

Ezekkel a jelölésekkel a távolságfüggvényünk ***magfüggvénye*** a következő alakban írható

$$\psi(t) = \frac{t^2 - 1}{2} + \psi_0(t) . \tag{25.26}$$

Az előző részhez hasonlóan definiáljuk a Newton-rendszert, és annak átskálázását. A Newton-irányt megváltoztatjuk az önreguláris távolságfüggvény segítségével a következőképpen:

$$\begin{aligned} AD\,\mathbf{d}_x &= \mathbf{0}\,, \\ (AD)^T\mathbf{d}_y + \mathbf{d}_s &= \mathbf{0}\,, \\ \mathbf{d}_x + \mathbf{d}_s &= -\nabla\Psi(\mathbf{v}) = \mathbf{v}^{-q} - \mathbf{v}\,. \end{aligned}$$

Az előző belsőpontos algoritmusokhoz hasonlóan az Önreguláris-primál-duál-belsőpontos algoritmus bemeneti adataként meg kell adnunk az $\varepsilon > 0$ számítási pontosságot, a τ eltérési paramétert ($0 \le \tau < 1$), a θ csökkentési paramétert ($0 < \theta < 1$), valamint egy $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0) \in \mathcal{P}_+ \times \mathcal{D}_+$ kezdeti megoldást, amelyre $(\mathbf{x}^0)^T\mathbf{s}^0 = \mu^0\, n$.

Önreguláris-primál-duál-belsőpontos

```
1  x ← x^0
2  s ← s^0
3  μ ← μ^0
4  while n μ ≥ ε
5        do μ ← (1 − θ)μ
6           while Ψ(v) ≤ (τ − 1)n/2
7                 do számítsuk ki a Δx és Δs értékét
8                    α minimalizálja Ψ-t
9                    x ← x + αΔx
10                   s ← s + αΔs
```

Az új távolságfüggvényünknek és az új Newton-rendszernek megfelelően definiálja az elemzésnél használt centralitási mértéket

$$\sigma(\mathbf{v}) = \|\nabla\Psi(\mathbf{v})\| = \left\|\mathbf{v}^{-q} - \mathbf{v}\right\| = \|\mathbf{d}_x + \mathbf{d}_s\| = \|(\mathbf{d}_x, \mathbf{d}_s)\| \tag{25.27}$$

kifejezéssel, ahol az utolsó egyenlőség $\mathbf{d}_x$ és $\mathbf{d}_s$ vektorok merőlegessége miatt igaz (ezt az előző részben már beláttuk).

Korlátok a v vektorra és a lépéshosszra

Első lépésként alsó és felső korlátot adunk a $\mathbf{v}$ vektor koordinátáira, melynek bizonyítását az Olvasóra bízzuk (lásd 25.3-23. gyakorlat).

25.59. lemma. *Legyen $\sigma = \sigma(\mathbf{v})$. Ekkor*

$$v_{\min} \ge (1+\sigma)^{-\frac{1}{q}}, \qquad v_{\max} \le 1 + \sigma\,.$$

Vezessük be a következő jelöléseket:

$$\Delta_x = \frac{\Delta\mathbf{x}}{\mathbf{x}} = \frac{\mathbf{d}_x}{\mathbf{v}}, \qquad \Delta_s = \frac{\Delta\mathbf{s}}{\mathbf{s}} = \frac{\mathbf{d}_s}{\mathbf{v}}\,. \tag{25.28}$$

Továbbá jelölje az új pontunkat az α lépéshosszal tompított Newton-lépés után $(\mathbf{x}^+, \mathbf{s}^+)$, azaz $\mathbf{x}^+ = \mathbf{x} + \alpha\,\Delta\mathbf{x}$ és $\mathbf{s}^+ = \mathbf{s} + \alpha\,\Delta\mathbf{s}$. Ekkor a következőt írhatjuk:

$$\mathbf{x}^+ = \mathbf{x}\,(\mathbf{e} + \alpha\,\Delta_x), \qquad \mathbf{s}^+ = \mathbf{s}\,(\mathbf{e} + \alpha\,\Delta_s)\,.$$

Könnyen látható, hogy α pontosan akkor megengedett lépéshossz, ha $\mathbf{e} + \alpha\,\Delta_x \geq 0$ és $\mathbf{e} + \alpha\,\Delta_s \geq 0$ teljesül, így a legnagyobb megengedett lépéshosszra fennáll a következő:

$$1 \leq \alpha_{\max}\,\|(\Delta_x, \Delta_s)\|_\infty \leq \alpha_{\max}\,\|(\Delta_x, \Delta_s)\|\ . \tag{25.29}$$

Ennek segítségével alsó becslést adunk a legnagyobb megengedett lépéshosszra.

25.60. lemma. *Legyen* (Δ_x, Δ_s) *a (25.28) formulák szerint definiált vektor, ekkor*

$$\|(\Delta_x, \Delta_s)\| \leq \sigma(1+\sigma)^{\frac{1}{q}}$$

teljesül. Valamint az $\alpha_{\max}$ *legnagyobb megengedett lépéshosszra fennáll az*

$$\alpha_{\max} \geq \frac{1}{\sigma(1+\sigma)^{\frac{1}{q}}}$$

egyenlőtlenség.

Bizonyítás. A 25.59. lemma és a σ mérték (25.27) definíciója alapján

$$\|(\Delta_x, \Delta_s)\| = \left\|\left(\frac{\mathbf{d}_x}{\mathbf{v}}, \frac{\mathbf{d}_s}{\mathbf{v}}\right)\right\| \leq \frac{\|(\mathbf{d}_x, \mathbf{d}_s)\|}{v_{\min}} = \frac{\sigma}{v_{\min}} \leq \sigma(1+\sigma)^{\frac{1}{q}}\ .$$

A (25.29) egyenlőtlenséget átrendezve

$$\alpha_{\max} \geq \frac{1}{\|(\Delta_x, \Delta_s)\|}$$

alsó becslést kapjuk, amiből az előző egyenlőtlenség figyelembevételével következik a lemma második állítása. ■

Vizsgáljuk meg a távolságfüggvény csökkenési mértékét az α lépéshossz függvényében. Az elemzés során megadunk egy α^* megengedett lépéshosszt, majd alsó korlátot adunk a dualitásrés új értékére α^* függvényében. Egy Newton-lépés megtétele után jelölje $\mathbf{v}$ vektor új értékét $\mathbf{v}^+ = \sqrt{\mathbf{x}^+\mathbf{s}^+/\mu}$. Ekkor

$$\begin{aligned}(\mathbf{v}^+)^2 &= \frac{(\mathbf{x}+\alpha\Delta\mathbf{x})(\mathbf{s}+\alpha\Delta\mathbf{s})}{\mu} = \frac{1}{\mu}\,\frac{\mathbf{x}}{\mathbf{v}}\left(\mathbf{v}+\alpha\,\frac{\mathbf{v}\,\Delta\mathbf{x}}{\mathbf{x}}\right)\frac{\mathbf{s}}{\mathbf{v}}\left(\mathbf{v}+\alpha\,\frac{\mathbf{v}\,\Delta\mathbf{s}}{\mathbf{s}}\right)\\ &= (\mathbf{v}+\alpha\,\mathbf{d}_x)(\mathbf{v}+\alpha\,\mathbf{d}_s)\ .\end{aligned}$$

Figyelembe véve a $\mathbf{d}_x$ és $\mathbf{d}_s$ vektorok ortogonalitását

$$\mathbf{e}^T(\mathbf{v}^+)^2 = \mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{d}_x+\mathbf{d}_s) = \mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{v}^{-q}-\mathbf{v})\ .$$

A távolságfüggvény értéke az új pontban

$$\begin{aligned}\Psi(\mathbf{x}^+,\mathbf{s}^+,\mu) &= \Psi(\mathbf{v}^+) = \frac{\mathbf{e}^T(\mathbf{v}^+)^2 - n}{2} + \Psi_0(\mathbf{v}^+)\\ &= \frac{\mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{v}^{-q}-\mathbf{v}) - n}{2} + \Psi_0\left(\sqrt{(\mathbf{v}+\alpha\,\mathbf{d}_x)(\mathbf{v}+\alpha\,\mathbf{d}_s)}\right)\ .\end{aligned} \tag{25.30}$$

A következő lemma a ψ_0 függvény egy fontos tulajdonságára mutat rá.

25.61. lemma. *Legyen $t_1 > 0$ és $t_2 > 0$. Ekkor*

$$\psi_0\left(\sqrt{t_1 t_2}\right) \leq \frac{1}{2}\left(\psi(t_1) + \psi_0(t_2)\right) \ .$$

Bizonyítás. Könnyen belátható, hogy az állítás pontosan akkor igaz, ha a $\psi_0(e^z)$ mint z függvénye konvex, ami akkor és csak akkor áll fenn, ha $\psi_0'(t) + t\,\psi_0''(t) \geq 0$ teljesül minden $t \geq 0$ esetén. Mivel

$$\psi_0'(t) = -t^{-q} \ , \qquad \psi_0''(t) = q\,t^{-1-q} \ ,$$

így $\psi_0'(t) + t\,\psi_0''(t) = (q-1)t^{-q} > 0$, ezzel a lemmát beláttuk. ■

Folytassuk a távolságfüggvény új pontbeli értékének vizsgálatát a lemma eredményét felhasználva

$$\Psi_0(\mathbf{v}^+) = \Psi_0\left(\sqrt{(\mathbf{v} + \alpha\,\mathbf{d}_x)(\mathbf{v} + \alpha\,\mathbf{d}_s)}\right) \leq \frac{1}{2}\sum_{i=1}^{n}\left(\psi_0(v_i + \alpha\,d_{xi}) + \psi_0(v_i + \alpha\,d_{si})\right) .$$

A fenti becslést a (25.30) egyenlőségbe helyettesítve

$$\Psi(\mathbf{x}^+, \mathbf{s}^+, \mu) \leq \frac{\mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{v}^{-q} - \mathbf{v}) - n}{2} + \frac{1}{2}\sum_{i=1}^{n}\left(\psi_0(v_i + \alpha\,d_{xi}) + \psi_0(v_i + \alpha\,d_{si})\right)$$

adódik. Legyen $f(\alpha)$ a Ψ függvény megváltozása egy lépés során a lépéshossz függvényében, azaz

$$f(\alpha) = \Psi(\mathbf{x}^+, \mathbf{s}^+, \mu) - \Psi(\mathbf{x}, \mathbf{s}, \mu) \leq f_1(\alpha) \ ,$$

ahol

$$\begin{aligned} f_1(\alpha) = \ & -\Psi(\mathbf{x}, \mathbf{s}, \mu) + \tfrac{\mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{v}^{-q} - \mathbf{v}) - n}{2} + \\ & + \tfrac{1}{2}\textstyle\sum_{i=1}^{n}\left(\psi_0(v_i + \alpha d_{xi}) + \psi_0(v_i + \alpha d_{si})\right) . \end{aligned}$$

Vegyük észre, hogy $f(0) = f_1(0) = 0$. Számoljuk ki az f_1 függvény α lépéshossz szerinti első, illetve második deriváltját.

$$f_1'(\alpha) = \frac{1}{2}\mathbf{v}^T(\mathbf{v}^{-q} - \mathbf{v}) + \frac{1}{2}\sum_{i=1}^{n}\left(\psi_0'(v_i + \alpha d_{xi})d_{xi} + \psi_0'(v_i + \alpha d_{si})d_{si}\right) .$$

A Newton-rendszer harmadik egyenletét, valamint σ (25.27) definícióját használva

$$\begin{aligned} f_1'(0) &= \frac{1}{2}\mathbf{v}^T(\mathbf{v}^{-q} - \mathbf{v}) - \frac{1}{2}\sum_{i=1}^{n} v_i^{-q}(d_{xi} + d_{si}) = \frac{1}{2}(\mathbf{d}_x + \mathbf{d}_s)^T(\mathbf{v} - \mathbf{v}^{-q}) \\ &= -\frac{1}{2}\sigma^2 \end{aligned} \tag{25.31}$$

adódik. Továbbá tekintetbe véve, hogy a $\psi''(t) = qt^{-1-q}$ függvény monoton csökken az

$$\begin{aligned} f_1''(\alpha) &= \frac{1}{2}\sum_{i=1}^{n}\left(\psi_0''(v_i + \alpha d_{xi})d_{xi}^2 + \psi_0''(v_i + \alpha d_{si})d_{si}^2\right) \\ &= \frac{1}{2}\sum_{i=1}^{n}\left(\left(q(v_i + \alpha d_{xi})^{-q-1}\right)d_{xi}^2 + \left(q(v_i + \alpha d_{si})^{-1-q}\right)d_{si}^2\right) \\ &\leq \frac{q}{2}\sum_{i=1}^{n}(v_{\min} - \alpha\sigma)^{-q-1}(d_{xi}^2 + d_{si}^2) \end{aligned}$$

egyenlőtlenség adódik, ahol az utolsó egyenlőtlenségnél a következőket használtuk fel

$$v_i + \alpha d_{xi} \geq v_{\min} - \alpha \|\mathbf{d}_x\| \geq v_{\min} - \alpha\sigma,$$

$$v_i + \alpha d_{si} \geq v_{\min} - \alpha \|\mathbf{d}_s\| \geq v_{\min} - \alpha\sigma \ .$$

A (25.27) azonosságot figyelembe véve

$$f_1''(\alpha) \leq h(\alpha) = \frac{1}{2} q\sigma^2 (v_{\min} - \alpha\sigma)^{-1-q} \tag{25.32}$$

egyenlőtlenség adódik. Minden olyan α lépéshosszra, melyre $v_{\min} - \alpha\sigma \geq 0$ teljesül, integrálással az

$$f_1'(\alpha) = f_1'(0) + \int_0^\alpha f_1''(\xi) d\xi \leq f_1'(0) + \int_0^\alpha h(\xi) d\xi$$

egyenlőtlenség adódik. Még egyszer integrálva az egyenlőtlenséget, valamint felhasználva az $f(\alpha) \leq f_1(\alpha)$ egyenlőtlenséget is, és azt, hogy $f_1(0) = 0$, az

$$f(\alpha) \leq f_1(\alpha) = \int_0^\alpha f_1'(\zeta) d\zeta \leq f_2(\alpha) = f_1'(0)\alpha + \int_0^\alpha \int_0^\zeta h(\xi) d\xi d\zeta$$

egyenlőtlenséget kapjuk. Nyilván $f_2''(\alpha) = h(\alpha) > 0$, így f_2 konvex függvény. Mivel $f_2(0) = 0$, $f_2'(0) = f_1'(0) < 0$ és $f_2''(\alpha) = h(\alpha)$ végtelenhez tart, ha α a $v_{\min}/\sigma$ értékhez tart, ezért az f_2 függvény a minimumát azon $\tilde{\alpha} > 0$ helyen veszi fel, mely az

$$f_2'(\alpha) = f_1'(0) + \int_0^\alpha h(\xi) d\xi = 0$$

egyenlet megoldása. A (25.32) összefüggés alapján az előző egyenlet a következővel ekvivalens

$$f_1'(0) + \frac{1}{2} q\sigma^2 \int_0^\alpha (v_{\min} - \xi\sigma)^{-1-q} d\xi = 0 \ .$$

Ezen egyenletet α-ra megoldva az

$$\tilde{\alpha} = \frac{v_{\min}}{\sigma} \left(1 - (1 + \sigma v_{\min}^q)^{\frac{1}{-q}}\right) \tag{25.33}$$

kifejezésre jutunk. A 25.59. lemma alapján

$$v_{\min}^q \geq \frac{1}{\sigma + 1} \ ,$$

amit a (25.33) összefüggésbe helyettesítve, az f_2 függvény minimum helyére a következő becslést kapjuk

$$\tilde{\alpha} \geq \frac{v_{\min}}{\sigma} \left(1 - \left(1 + \frac{\sigma}{\sigma + 1}\right)^{\frac{1}{-q}}\right) \ .$$

A 25.3-24. gyakorlat eredményét alkalmazva

$$\left(1 + \frac{\sigma}{\sigma + 1}\right)^{-\frac{1}{q}} = \left(1 - \frac{\sigma}{2\sigma + 1}\right)^{\frac{1}{q}} \leq 1 - \frac{1}{\sigma} q(2\sigma + 1)$$

adódik. Az előző egyenlőtlenség és a 25.59. lemma segítségével az $\tilde{\alpha}$ lépéshosszra a következő alsó korlátot nyerjük

$$\tilde{\alpha} \geq \frac{v_{\min}}{\sigma} \frac{\sigma}{q(2\sigma+1)} = \frac{v_{\min}}{q(2\sigma+1)} \geq \frac{1}{q(2\sigma+1)(\sigma+1)^{\frac{1}{\sigma}}} .$$

A továbbiakban feltesszük, hogy $\sigma \geq 1$ és az

$$\alpha^* = \frac{1}{3q\sigma(\sigma+1)^{\frac{1}{q}}} \tag{25.34}$$

lépéshosszal számolunk. Vegyük észre, hogy $\alpha^* \leq \tilde{\alpha}$.

Alkalmazzuk a 25.3-25. gyakorlatot az f_2 függvényre. Először ellenőrizzük, hogy az előző lemma feltételei teljesülnek-e: $f_2(0) = 0$ és $f_2'(0) < 0$. Továbbá $f_2''(\alpha) = h(\alpha) > 0$, és

$$f_2'''(\alpha) = h'(\alpha) = \frac{q(q+1)\sigma^3}{2}(v_{\min} - \alpha\sigma)^{-2-q} > 0 .$$

Tehát alkalmazhatjuk a lemmát, valamint a (25.31) azonosságot figyelembe véve az $f(\alpha^*)$ értékére a következő felső becslést kapjuk:

$$f(\alpha^*) \leq f_2(\alpha^*) \leq \frac{\alpha^* f_2'(0)}{2} = \frac{\alpha^* f_1'(0)}{2} = -\frac{\alpha^* \sigma^2}{4} .$$

Végül behelyettesítve α^* (25.34) definícióját és tekintetbe véve $\sigma \geq 1$ feltevésünket

$$f(\alpha^*) \leq \frac{-\sigma}{12q(\sigma+1)^{\frac{1}{q}}} \leq \frac{-\sigma}{12q(2\sigma)^{\frac{1}{q}}} \leq \frac{-\sigma^{\frac{q-1}{q}}}{24q} . \tag{25.35}$$

A μ paraméter iterálásának hatása

A következőkben néhány technikai eredményt ismertetünk, melyekre a konkrét elemzések során lesz szükségünk. Mivel

$$\psi'(t) = t - t^{-q} , \qquad \psi''(t) = 1 + qt^{-q-1} ,$$

ezért $\psi''(t) \geq 1$ minden $t > 0$ esetén. Továbbá ψ függvényt második deriváltja és integrálás segítségével felírva alsó, illetve felső becslést adhatunk meg (a bizonyításhoz lásd 25.3-26. gyakorlatot).

25.62. lemma. *A ψ magfüggvényre a következő egyenlőtlenség teljesül*

$$\frac{1}{2}(t-1)^2 \leq \psi(t) \leq \frac{1}{2}\psi'(t)^2 , \qquad t > 0 .$$

A fenti lemmában a távolságfüggvény magfüggvényére kapott alsó becslés segítségével a $\mathbf{v}$ vektor normájára felső becslést adhatunk.

25.63. következmény. *A következő egyenlőtlenség teljesül*

$$\|\mathbf{v}\| \leq \sqrt{n} + \sqrt{2\Psi(\mathbf{v})} .$$

Bizonyítás. Felhasználva a 25.62. lemmát és a Cauchy–Schwarz-egyenlőtlenséget a

$$\begin{aligned} 2\Psi(\mathbf{v}) = \quad & 2\sum_{i=1}^{n}(v_i-1)^2 = \|\mathbf{v}\|^2 - 2\mathbf{e}^T\mathbf{v} + n \\ \geq \quad & \|\mathbf{v}\|^2 - 2\,\|\mathbf{e}\|\,\|\mathbf{v}\| + \|\mathbf{e}\|^2 = (\|\mathbf{v}\| - \|\mathbf{e}\|)^2 \end{aligned}$$

egyenlőtlenséget kapjuk. Ezt átrendezve

$$\|\mathbf{v}\| \leq \|\mathbf{e}\| + \sqrt{2\Psi(\mathbf{v})} = \sqrt{n} + \sqrt{2\Psi(\mathbf{v})}$$

adódik, amiből már következik az állítás. ■

Az előző lemmában a ψ függvényre adott felső korlátot összevetve a σ definíciójával a távolságfüggvényre felső becslést eredményez.

25.64. következmény. $\Psi(\mathbf{v}) \leq \frac{1}{2}\sigma(\mathbf{v})^2$.

Bizonyítás. Ismét a 25.62. lemmát használva, illetve a (25.27) alapján

$$\Psi(\mathbf{v}) = \sum_{i=1}^{n}\psi(v_i) \leq \frac{1}{2}\sum_{i=1}^{n}\psi'(v_i)^2 = \frac{1}{2}\,\|\nabla\Psi(\mathbf{v})\|^2 = \frac{1}{2}\,\sigma(\mathbf{v})^2\ .$$

Ezzel a bizonyítást befejeztük. ■

A következőkben megvizsgáljuk, hogyan változik a magfüggvény értéke, ha a függvény argumentumát egy $\beta \geq 1$ számmal megszorozzuk.

25.65. lemma. *Legyen $\beta \geq 1$. Ekkor*

$$\psi(\beta t) \leq \psi(t) + \frac{1}{2}(\beta-1)t^2\ .$$

Bizonyítás. A (25.26)-ban a bevezetett jelölést használva

$$\psi(\beta t) = \frac{\beta^2t^2-1}{2} + \psi_0(\beta t) = \psi(t) + \frac{1}{2}(\beta^2t^2 - t^2) + \psi_0(\beta t) - \psi_0(t)\ .$$

Mivel a ψ_0 függvény monoton fogyó, $\psi_0(\beta t) - \psi_0(t) \leq 0$. Ebből pedig már következik a bizonyítandó egyenlőtlenség. ■

A μ paraméter megváltozásának a távolságfüggvényre gyakorolt hatásának vizsgálatakor használjuk fel a fenti lemmában kapott egyenlőtlenséget. A 25.66. lemmában megfogalmazott egyenlőtlenség bizonyítását az Olvasóra bízzuk (lásd 25.3-27. gyakorlat).

25.66. lemma. *Legyen $\theta \in (0,1)$ és $\mu^+ = (1-\theta)\mu$. Ekkor*

$$\Psi(\mathbf{x},\mathbf{s},\mu^+) \leq \Psi(\mathbf{x},\mathbf{s},\mu) + \frac{\theta}{2(1-\theta)}\left(2\Psi(\mathbf{v}) + \sqrt{2n\Psi(\mathbf{v})} + n\right)\ .$$

A szükséges becslések előállítása után most rátérünk az algoritmus lépésszámának meghatározására. Az eljárás külső ciklusa elején (a μ paraméter iterálása előtt) $\Phi(\mathbf{x},\mathbf{s},\mu) = \Psi(\mathbf{v}) \le \tau$ teljesül. A 25.66. lemma alapján a μ frissítése után

$$\Phi(\mathbf{x},\mathbf{s},\mu^+) \le \tau + \frac{\theta}{2(1-\theta)}\left(2\tau + \sqrt{2n\tau} + n\right) \tag{25.36}$$

áll fenn. A (25.34) alatt meghatározott α^* értéket használva (25.35) alapján a távolságfüggvény értékének csökkenése legalább

$$\frac{\sigma^{\frac{q-1}{q}}}{24q}\ . \tag{25.37}$$

Megjegyeznénk, hogy ez az eredmény csak a $\sigma \ge 1$ feltétel teljesülése mellett igaz, ám a 25.64. következménybeli egyenlőtlenség alapján ennek elégséges feltétele a $\Phi(\mathbf{x},\mathbf{s},\mu) \ge 1$ egyenlőtlenség. Ami pedig $\tau \ge 1$ mellett biztosan fennáll minden belső ciklus megkezdésekor. A 25.64. következmény és (25.37) alapján minden belső iteráció során a távolságfüggvény csökkenése legalább

$$\frac{\Psi^{\frac{q-1}{2q}}}{24q}\ , \tag{25.38}$$

ahol $\Psi = \Psi(\mathbf{x},\mathbf{s},\mu)$ a távolságfüggvény értéke a belső iteráció megkezdése előtt.

25.67. lemma. *Minden belső ciklusban legfeljebb*

$$\left\lceil \frac{48q^2(\tau + \frac{\theta}{2(1-\theta)}(2\tau + \sqrt{2n\tau} + n))^{\frac{q+1}{2q}}}{q+1} \right\rceil$$

iteráció történik.

Bizonyítás. Jelölje Ψ a távolságfüggvény értékét a belső ciklus elején. Egy iteráció után az új értéket pedig jelölje Ψ^+. Ekkor (25.38) alapján

$$\Psi^+ \le \Psi - \frac{\Psi^{\frac{q-1}{2q}}}{24q}\ .$$

Emeljük az egyenlőtlenséget $(q+1)/(2q)$ hatványra, majd alkalmazzuk az $(1-t)^\alpha \le 1 - \alpha t,\ \ (\alpha,\ t \in [0,1])$ összefüggést

$$\begin{aligned}(\Psi^+)^{\frac{q+1}{2q}} &\le \left(\Psi - \frac{\Psi^{\frac{q-1}{2q}}}{24q}\right)^{\frac{q+1}{2q}} = \Psi^{\frac{q+1}{2q}}\left(1 - \frac{\Psi^{-\frac{q+1}{2q}}}{24q}\right)^{\frac{q+1}{2q}} \\ &\le \Psi^{\frac{q+1}{2q}}\left(1 - \frac{q+1}{2q}\ \frac{\Psi^{-\frac{q+1}{2q}}}{24q}\right) = \Psi^{\frac{q+1}{2q}} - \frac{q+1}{48q^2}\ .\end{aligned}$$

Ha $\Psi^{(k)}$ jelöli a k-adik belső iteráció után a távolságfüggvény értékét, akkor a következő becslés áll fenn

$$\Psi^{(k)} \le \Psi^{\frac{q+1}{2q}} - k\,\frac{q+1}{48q^2}\ .$$

Ha a $\Psi^{(k)}$ értékre adott felső korlát kisebb, mint τ, akkor a belső iterációk száma legfeljebb k, tehát

$$\Psi^{\frac{q+1}{2q}} - k\,\frac{q+1}{48q^2} \le \tau\,,$$

átrendezés után

$$\frac{48q^2}{q+1}\left(\Psi^{\frac{q+1}{2q}} - \tau\right) \le k$$

adódik. Az alsó korlátot felülről becsülve $\tau > 0$, illetve (25.36) figyelembevételével a lemma állítását kapjuk. ■

Az algoritmusunk hosszú lépéses módszer, mivel a θ paraméter nem függ a feladat dimenziójától. Ezt figyelembe véve a külső iterációk számát a következő lemma adja meg, melynek bizonyítását az Olvasóra bízzuk (lásd 25.3-28. gyakorlat).

25.68. lemma. *A külső iterációk száma legfeljebb* $\left\lceil \frac{1}{\theta} \lg \frac{n}{\varepsilon} \right\rceil$.

A belső iterációk maximális számának és a külső iterációk maximális számának szorzata felső becslést ad az eljárás lépésszámára.

25.69. tétel. *Az algoritmus legfeljebb*

$$\left\lceil \frac{48q^2(\tau + \frac{\theta}{2(1-\theta)}(2\tau + \sqrt{2n\tau} + n))^{\frac{q+1}{2q}}}{q+1} \right\rceil \left\lceil \frac{1}{\theta} \lg \frac{n}{\varepsilon} \right\rceil$$

lépésben egy olyan $(\mathbf{x}, \mathbf{s})$ *megengedett pontpárt ad, melyre* $\mathbf{x}^T\mathbf{s} < \varepsilon$.

Ezek alapján az algoritmus lépésszáma $0 < \theta < 1$ konstans, $\tau = n$ választás esetén

$$O\left(q\, n^{\frac{q+1}{2q}} \lg \frac{n}{\varepsilon}\right)\,.$$

Rögzített $q > 1$ esetén ez a hosszú lépéses algoritmusokra ismert $O(n \lg \frac{n}{\varepsilon})$ lépésszám javítását jelenti. A $q = \frac{1}{2}\lg n$ választás minimalizálja az előző felső korlátot. Ekkor az iterációszám

$$O\left(\sqrt{n} \lg n \lg \frac{n}{\varepsilon}\right),$$

ami már igen közel van a belsőpontos algoritmusok elméletében ismert legjobb $O(\sqrt{n} \lg \frac{n}{\varepsilon})$ lépésszámhoz.

Gyakorlatok

25.3-1. Tekintsük a következő adatokkal adott lineáris programozási feladatot

$$\begin{array}{lrrrl} \min & 5\,x_1 & -5\,x_2 & +7\,x_3 & \\ & 3\,x_1 & -3\,x_2 & -2\,x_3 & = -3 \\ & x_1 & -1\,x_2 & -5\,x_3 & = \;\; 1\,, \end{array}$$

ahol $x_1,\ x_2, x_3 \ge 0$. Vizsgáljuk meg, teljesül-e a feladatra a belső pont feltétel.

25.3-2. Bizonyítsuk be a 25.38. állítást.

25.3-3. Bizonyítsuk be a 25.39. állítást.

25.3-4. Bizonyítsuk be a 25.40. állítást. *Útmutatás.* Bizonyítsuk be, hogy a

$$\begin{pmatrix} 0 & A & 0 \\ A^T & 0 & I \\ 0 & \bar{S} & \bar{X} \end{pmatrix}$$

együtthatómátrix reguláris, ahol $\bar{X}$, és $\bar{S}$ az $\bar{\mathbf{x}}$, illetve az $\bar{\mathbf{s}}$ vektorokból képzett diagonális mátrixok.

25.3-5. Bizonyítsuk be a 25.41. lemmát.

25.3-6. Bizonyítsuk be a 25.42. lemmát.

25.3-7. Legyen adott a (P) és a (D) feladat, valamint az $\alpha > 0$ szám. Tegyük fel, hogy $\bar{\mathbf{x}} \in \mathcal{P}_+$ és $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$. Bizonyítsuk be, hogy $\bar{\mathbf{x}} + \alpha\,\Delta\mathbf{x} \geq \mathbf{0}$ és $\bar{\mathbf{s}} + \alpha\,\Delta\mathbf{s} \geq \mathbf{0}$ pontosan akkor teljesül, ha

$$\bar{\mathbf{x}}\,\bar{\mathbf{s}} + \alpha\,(\mu\,\mathbf{e} - \bar{\mathbf{x}}\,\bar{\mathbf{s}}) + \alpha^2 \Delta\mathbf{x}\,\Delta\mathbf{s} \geq \mathbf{0}\ . \tag{25.39}$$

Adott $\Delta\mathbf{x}$ és $\Delta\mathbf{s}$ esetén határozzuk meg a maximális α értéket, amelyre az (25.39) egyenlőtlenség teljesül.

25.3-8. Bizonyítsuk be a 25.43. lemmát. Bizonyítsuk be azt is, hogy az adott μ-centrum dualitásrése is $n\,\mu$. *Útmutatás.* Egy $\mathbf{x} \in \mathcal{P}$ és $(\mathbf{y}, \mathbf{s}) \in \mathcal{D}$ megoldáspár μ-centrum, ha $\mathbf{x}\,\mathbf{s} = \mu\,\mathbf{e}$ teljesül.

25.3-9. Bizonyítsuk be a 25.44. következményt.

25.3-10. Legyen $\mathbf{a}$ és $\mathbf{b}$ tetszőleges egymásra merőleges vektor. Bizonyítsuk be, hogy

$$\|\mathbf{a}\,\mathbf{b}\|_\infty \leq \frac{1}{4}\,\|\mathbf{a}+\mathbf{b}\|^2 \quad \text{és} \quad \|\mathbf{a}\,\mathbf{b}\| \leq \frac{\sqrt{2}}{4}\,\|\mathbf{a}+\mathbf{b}\|^2\ .$$

25.3-11. Bizonyítsuk be a 25.46. állítást.

25.3-12. Legyen $\partial = \partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \mu)$ és $\partial^+ = \partial_1(\mathbf{x}^+, \mathbf{s}^+; \mu)$. Határozzuk meg azt a legkisebb ∂ értéket, amely garantálja, hogy $\partial^+ \leq \partial$ teljesüljön. *Útmutatás.* Használjuk fel a 25.47. tételt.

25.3-13. (*Ling 1. lemmája* (1993)) Legyen $\mathbf{u} \in \mathbb{R}^p$ olyan vektor, amelyre $\mathbf{u} > -\mathbf{e}$ és legyen $\sigma = \mathbf{e}^T\mathbf{u}$. Bizonyítsuk be, hogy ha $\mathbf{u} \geq \mathbf{0}$ vagy $\mathbf{u} \leq \mathbf{0}$, akkor

$$\sum_{i=1}^{p} \frac{-v_i}{1+v_i} \leq \frac{-\sigma}{1+\sigma}\ ,$$

és egyenlőség pontosan akkor teljesül, ha legfeljebb egy nem nulla koordinátája van az $\mathbf{u}$ vektornak. *Útmutatás.* Használjuk fel az

$$f : (-1,\infty)^p \to \mathbb{R} \quad \text{és} \quad f(\mathbf{u}) = \sum_{i=1}^{p} \frac{-v_i}{1+v_i}$$

függvény konvexitását.

25.3-14. (*Ling 2. lemmája* (1993)) Legyenek az $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ ortogonális vektorok, és tegyük fel, hogy $\|\mathbf{u}+\mathbf{v}\| = 2\rho$, ahol $\rho < 1$. Bizonyítsuk be, hogy

$$\mathbf{e}^T\left(\frac{\mathbf{e}}{\mathbf{e}+\mathbf{u}\,\mathbf{v}} - \mathbf{e}\right) \leq \frac{2\rho^4}{1-\rho^4}\ .$$

Útmutatás. Alkalmazzuk az első Ling-lemmát két egymásra merőleges vektor koordinátánkénti szorzatára.

25.3-15. Ismételjük meg a 25.3-12. gyakorlatot, de most használjuk fel a 25.48. tételt. Egyenlő-e a két szám? Mi ennek a jelentése?

25.3-16. Bizonyítsuk be a 25.49. lemmát.

25.3-17. Bizonyítsuk be a (25.23) összefüggést.

25.3-18. Bizonyítsuk be a 25.52. lemmát. *Útmutatás.* Felhasználva a centralitási mérték definícióját, és hogy $\mathbf{v} > \mathbf{0}$, $-2\partial v_i \leq 1 - v_i^2 \leq 2\partial v_i$ teljesül minden indexre.

25.3-19. Bizonyítsuk be, hogy a prediktor-korrektor algoritmus $\tau = 1/2$ és $\alpha = 1/(2\sqrt{n})$ értékek esetén jól definiált, azaz minden iteráció után a kapott $(\mathbf{x}, \mathbf{s})$ pontra teljesül $\partial(\mathbf{x}, \mathbf{s}, \mu) \leq \tau$.

25.3-20. Bizonyítsuk be a 25.55. lemmát. *Útmutatás.* Kövessük a 25.53. lemma bizonyítását, azzal a különbséggel, hogy a $\left\|\mathbf{d}_x^a\, \mathbf{d}_s^a\right\|_\infty \leq n/4$ becslés helyett a $\left\|\mathbf{d}_x^a\, \mathbf{d}_s^a\right\|_\infty \leq \left\|\mathbf{d}_x^a\, \mathbf{d}_s^a\right\|$ felső korlátot alkalmazzuk.

25.3-21. Bizonyítsuk be, hogy a prediktor-korrektor algoritmus adaptív változata $\tau = 1/3$ és $\alpha = 2/(1 + \sqrt{1 + 13\,\|\mathbf{d}_x^a\, \mathbf{d}_s^a\|})$ értékek esetén jól definiált.

25.3-22. Bizonyítsuk be a 25.56. lemmát. *Útmutatás.* Vizsgáljuk meg az

$$f(x) = 1 - \frac{2}{1 + \sqrt{1 + 13x}} \qquad \text{függvényt .}$$

25.3-23. Bizonyítsuk be a 25.59. lemmát. *Útmutatás.* Induljunk ki a $\sigma = \|\mathbf{v}^{-q} - \mathbf{v}\| \geq \|v_i^{-q} - v_i\|$ becslésből és végezzünk esetszétválasztást az 1 és $v_{\min}$, illetve $v_{\max}$ értékek viszonyára tekintettel.

25.3-24. Bizonyítsuk be, hogy ha $p \in [0, 1]$, akkor $(1-t)^p \leq 1 - p\,t$ teljesül minden $t \in [0, 1]$ esetén.

25.3-25. Legyen $h(t)$ kétszer differenciálható konvex függvény, továbbá $h(0) = 0$, $h'(0) < 0$ és a h függvény a $t^* > 0$ pontban veszi fel a (globális) minimumát. Bizonyítsuk be, hogy ha h'' függvény monoton nő a $[0, t^*]$ intervallumon, akkor

$$h(t) \leq \frac{t h'(0)}{2} \qquad \text{minden } 0 \leq t \leq t^* \text{ esetén .}$$

25.3-26. Bizonyítsuk be a 25.62. lemmát.

25.3-27. Bizonyítsuk be a 25.66. lemmát. *Útmutatás.* Alkalmazzuk a 25.65. lemmát $\beta = 1/\sqrt{1-\theta}$ értékre, majd használjuk fel a 25.63. következmény eredményét a további becslésekhez.

25.3-28. Bizonyítsuk be a 25.68. lemmát. *Útmutatás.* A 25.45. tétel bizonyításához hasonlóan igazolható.

Megjegyzések a fejezethez

Lineáris programozási feladat megoldásának a módszerei és a számítógépek szoros kapcsolatban fejlődtek az elmúlt évtizedeken keresztül. Az 1950-es és 1960-as évek elején, a gyakorlati lineáris programozási feladatok megoldása során, memória és merev lemez kapacitási korlátok határozták meg a lineáris programozási algoritmusok fejlesztésének irányát

és a gyakorlatban megoldható feladatok méretét. Mivel a lineáris programozási modellek mérete, a gyakorlati alkalmazások követelmények megfelelően folyamatosan növekedett, egy ideig ez kihatott a számítógépek fejlesztésére is [80]. Ma nem a lineáris programozási feladatok méretének a növekedése jelenti a számítógépek fejlesztésének a motorját, habár a nagyméretű lineáris és a lineáris egészértékű feladatok megoldása továbbra is komoly számítási kapacitások meglétét igényeli.

A szakirodalomból ismert egyik első modell, amelyet lineáris egyenlőtlenségrendszerrel fogalmaztak meg, a Fourier-féle mechanikai elv volt, amely Farkas Gyula vizsgálódásainak is a kiindulópontját adta. A lineáris egyenlőtlenségrendszerek megoldhatóságának a kérdését Farkas Gyula [101, 102, 103] már a XIX. században tanulmányozta. Az első ismert számítási eljárást, speciális lineáris egyenlőtlenségrendszerek megoldására Fourier fogalmazta meg. Módszerét tetszőleges lineáris egyenlőtlenségrendszer megoldására Motzkin általánosította és ennek a következtében, ma is Fourier-Motzkin eliminációs módszerként [290] ismerjük. Farkas Gyulának a lineáris egyenlőtlenségrendszerek megoldhatóságáról szóló eredménye, amelyet ma Farkas-lemmaként ismer az egész világ, az optimumszámítás egyik legtöbbet idézet eredménye. Farkas Gyula életéről, optimalizáláselméleti munkásságáról és annak hatásairól Prékopa András írt egy méltatást [291].

Évtizedekig a lineáris programozási feladatok megoldására csak a Dantzig-féle szimplex módszer és variánsai [81] álltak rendelkezésre. Magyarországon a modern lineáris programozási szemlélet elterjesztésére alapvető hatással volt Prékopa [289] könyve. A szimplex módszer és variánsainak megvalósítási technikáit mutatja be Maros [246] hiánypótló könyve. Klee és Minty [200] megmutatták, hogy létezik olyan lineáris programozási feladat, amelynek a megoldása során, egy adott megengedett bázis megoldásból indulva a szimplex módszernek $2^n - 1$ báziscserére van szüksége az optimális megoldás előállítására. Klee és Minty eredménye óta izgalmas megoldatlan kérdése volt a lineáris programozás témakörének, hogy létezik-e olyan algoritmus, amely bármely lineáris programozási feladatot *polinomiális* (illetve *erősen polinomiális*) lépésben old meg, azaz az iterációk száma $O(p(n, L))$ (vagy $O(p(n))$) nagyságrendű, ahol a p egy véges fokú polinom, n a feladat változóinak a száma, míg L a feladat tárigényének egy felső korlátja.

Hacsián [196] megmutatta, hogy a Judin és Nemirovski [185] által konvex optimalizálási feladatok megoldására konstruált *ellipszoid módszert* lineáris programozási feladatra adaptálva polinomiális algoritmust eredményez. Hacsián bebizonyította, hogy az ellipszoid módszer legfeljebb $O(n^2 L)$ iterációban, és $O(n^4 L)$ aritmetikai művelettel bármely lineáris programozási feladatot megold. Gács és Lovász [132] dolgozatukban az ellipszoid módszernek egy egyszerű bizonyítását ismertették. Évekig tartó erőfeszítések eredményeként az ellipszoid módszer különböző számítógépes megvalósításait fejlesztették ki, de ezek egyegy nagyon speciális kombinatorikus optimalizálási feladattól eltekintve, gyakorlati feladatok megoldása során meg sem közelítették a szimplex alapú számítógépes programcsomagok 1980-as évekbeli hatékonyságát. Már lankadni kezdett a téma iránti érdeklődés, amikor Karmarkar [188] felfedezte *projektív skálázású primál belsőpontos algoritmusát* lineáris programozási feladatok megoldására. Bebizonyította, hogy algoritmusa nem több, mint $O(n L)$ iterációban oldja meg a lineáris programozási feladatokat, ahol egy iteráció számítási igénye legfeljebb $O(n^{2.5})$ aritmetikai művelet. Ezzel Karmarkar nem csak Hacsián $O(n^4 L)$ legrosszabb esetben adott aritmetikai művelet korlátját javította $O(n^{3.5} L)$-re, hanem azt is állította, hogy algoritmusa a szimplex módszer implementációnál is sokkal gyorsabban old meg nagy méretű lineáris programozási feladatokat. Azóta számos kutató dolgozott azon,

hogy olyan belsőpontos algoritmusokat fogalmazzanak meg, amelyeknek nem csak az elméleti lépésszáma polinomiális, hanem numerikus szempontból is hatékony legyen. Kezdetben Karmarkar algoritmusának egyszerűsített variánsait dolgozták ki. Megmutatták, hogy az ún. *projektív transzformáció* nem nélkülözhetetlen eleme a lineáris programozási feladat belsőpontos módszerrel történő megoldásánál. Több kutató észrevette, hogy a Karmarkar-algoritmus egyik variánsa egy régi nemlineáris optimalizálási feladatok megoldására kifejlesztett algoritmus, az ún. *logaritmikus büntetőfüggvényes módszer,* lineáris programozási megfelelője. Ez indította el néhány régi nemlineáris programozási algoritmus újra felfedezését, lineáris programozási feladatokra való alkalmazását. Meglepő észrevételt tettek a kutatók: a logaritmikus barrier módszer lineáris programozási megfelelőjét Frisch [122] dolgozta ki, a *középpontok módszere* Huard [164] nevéhez köthető és az általunk is tárgyalt *affin skálázási algoritmus* eredeti tisztán primál, illetve duál változatát Dikin fogalmazta meg. Természetesen az ötvenes és hatvanas években ezeknek az algoritmusoknak sem a gyakorlatban hatékony számítógépes implementálására, sem pedig a lépésszám vizsgálatára nem került sor. Mindhárom szerző messze megelőzte a korát és eredményeiket évtizedekre elfelejtette a tudományos közvélemény.

A modern belsőpontos módszerek numerikus és számítógépes variánsainak a kidolgozásában kulcsszerepet játszott Sonnevend [318] és Megiddo [249] egymástól független eredményei, az ún. *centrális úttal* és *analitikus centrummal* kapcsolatos vizsgálataik. Sonnevend fogalmazta meg az első centrális utat követő algoritmust. Ezt követően Renegar [296] és tőle függetlenül Roos és Vial adott lineáris programozási feladatokat legfeljebb $O(\sqrt{n}L)$ iterációban és $O(n^3L)$ aritmetikai művelettel megoldó belsőpontos algoritmust. Ezek az eredmények jelentősen hozzájárultak Kojima, Mizuno és Yoshise [208] *primál-duál Newton-lépéses logaritmikus barrier algoritmusának* a megfogalmazásához, és az algoritmus legrosszabb esetben való lépésszám felső korlátjának a bizonyításához. Ma is ez az algoritmus a legtöbb számítógépes implementáció alapja. Mind elméleti, mind pedig gyakorlati szempontból igen érdekes változata a belsőpontos algoritmusoknak az ún. *prediktor-korrektor belsőpontos módszer.* Ebből az egyik legérdekesebb variánst, a Mizuno, Todd és Ye [251] prediktor-korrektor algoritmust tárgyaltuk ebben a fejezetben. A belsőpontos módszerek egyik sokat kritizált tulajdonsága, hogy az algoritmusoknak ahhoz, hogy elkezdhessék a feladat megoldását, egy belsőpont megoldás szükséges. Ezt a kérdést oldották meg, a fejezeten általunk is bemutatott módon, beágyazással, Ye, Todd és Mizuno [357]. A beágyazásról, belsőpontos algoritmusokról és azok komplexitásáról, pontos megoldás előállításáról és a témakörhöz kapcsolódó numerikus és számítógépes kérdésekről az érdeklődő olvasó további információkat talál Ye [356] és Roos, Terlaky és Vial [299] könyveiben.

Akit a belsőpontos algoritmusok implementációjának részletei érdeklik, azok az Andersen, Gondzio, Mészáros, Xu által írt könyvfejezethez [19], vagy a Zhu, Peng, Terlaky, Zhang cikkhez [365] fordulhatnak. A belsőpontos módszerek segítségével, napjainkban hatalmas méretű lineáris programozási feladatokat képesek megoldani. Ezeknek a feladatoknak sokszor néhány millió változójuk van, és a legjobb belsőpontos algoritmusra épülő lineáris programozási számítógépes programcsomagokkal 30-60 Newton-rendszer megoldásával megoldják a feladatot. A belsőpontos módszerek számítógépes implementációinak gyorsaságát és minőségét jelentősen befolyásolja a ritka mátrixokkal való műveletekre felhasznált számítási eljárások (szimbolikus faktorizáció, ritka Cholesky- és Bunch–Parlett-faktorizáció) hatékony számítógépes megvalósítása. Numerikus szempontból, a belsőpontos módszerek implementálásának az egyik legfontosabb kérdése, hogyan tudjuk kellő pon-

tossággal kiszámítani a rosszul kondícionált $AXS^{-1}A^T$ mátrix inverzét. Erre a kérdésre, Saunders and Tomlin [306] adott egy ötletes választ: regularizáljuk a mátrixot és a $\Delta\mathbf{x}, \Delta\mathbf{s}$ Newton-irányokat csak közelítőleg számítsuk ki. Ez a módosítás nem érinti hátrányosan a belsőpontos módszerek működését.

A belsőpontos módszerek számítógépes megvalósításainak a fejlesztéséhez jelentősen hozzájárultak a munkáikkal E. D. Andersen és K. D. Andersen [18], Gondzio [140] és Mészáros [254, 255]. További érdekes eredmények találhatók a [141, 245] cikkekben. Mészáros Csaba BPMPD programcsomagja [255], és annak a Dash Association által forgalmazott XPRESS-MP lineáris programozási programcsomagba beépített, továbbfejlesztett változata, ma napjainkban az egyik legjobb belsőpontos lineáris és kvadratikus programozási számítógépes program.

A pivot algoritmusok, illetve a belsőpontos módszerek előnyös és hátrányos elméleti és gyakorlati tulajdonságait hasonlította össze Illés és Terlaky [335].

A belsőpontos módszereknél a legérdekesebb, és igen ellentmondásos eredmény az, hogy a belsőpontos módszerek rövid lépéses változatai elméletileg hatékonyabbak, mint a hosszú lépéses variánsok. A gyakorlatban éppen az ellenkezőjét tapasztaljuk és intuíciónk is azt mondja, hogy a hosszú lépéses algoritmus variánsoknak gyakorlati szempontból hatékonyabbaknak kell lenniük. Valószínűleg a klasszikus belsőpontos módszerek keresési irányát kell megváltoztatni a hosszú lépéses módszerekben. Ezt ismerték fel Peng, Roos és Terlaky [274], akik kifejlesztették az önreguláris függvények elméletét és az önreguláris belsőpontos módszerek első változatait.

A belsőpontos módszerek elméletének érdekes fejleménye Deza, Nematollahi, Peyghami és Terlaky eredménye [89], miszerint a szimplex algoritmusnál ismert Klee–Minty példához hasonló jelenség – azaz az algoritmus „végiglátogatja” a Klee–Minty kocka összes csúcsát, még mielőtt az optimális csúcsba eljut – a belsőpontos algoritmusoknál is előfordulhat. Ennek a jelenségnek az oka az exponenciális számú – redundáns – feltételek jelenlétében, és a poliéder struktúrájában keresendő.

A belsőpontos módszerek iránt érdeklődők számára a Roos, Terlaky és Vial [299],, Ye [356] illetve a Peng, Roos és Terlaky [273] könyvek tanulmányozását javasoljuk. A belsőpontos módszerek fejlesztésének elmúlt több, mint húsz éves időszakában számos optimalizálási (elégséges komplementaritási-, sima konvex optimalizálási-, szemidefinit optimalizálási- és másodrendű kúp optimalizálási-) feladatosztályra is kiterjesztették a belsőpontos módszerek variánsait, jelentősen kiszélesítve a gyakorlati problémáknak numerikusan megoldható körét. Lineáris komplementaritási feladatok megoldása iránt érdeklődők Kojimáék [209] könyvéből, illetve Illések [168, 169] cikkeiből és az azokban felsorolt számos irodalmi hivatkozásokból tájékozódhatnak. Sima konvex optimalizálás területén alapvető eredményeket értek el Nyeszterov és Nemirovski [258] illetve Jarre [178]. A szemidefinit optimalizálás elméleti alapjait és első belsőpontos algoritmusait Alizadeh dolgozta ki és közölte 1991-es doktori disszertációjában. Alizadeh művei közül a [16] tanulmányt javasoljuk feldolgozásra. A szemidefinit és a másodrendű kúp optimalizálási feladatok mérnöki és kombinatorikus optimalizálási alkalmazásainak a virágkorát éljük. A valódi alkalmazások alapját természetesen az alaposan kidolgozott elmélet és algoritmusok mellett, a megfelelően felépített modellek és számítógép programok jelentik. De Klerk [203] könyvében a szemidefinit optimalizálási feladatok megoldására kifejlesztett legfontosabb belsőpontos algoritmusok mellett számos kombinatorikus optimalizálási alkalmazást is tárgyal. Aki a szemidefinit és a másodrendű kúp optimalizálás témakörét minél teljesebben szeretné meg-

ismerni, nem kerülheti el a Wolkowicz, Saigal és Vanderberghe által szerkesztett terjedelmes [352] mű olvasását sem.

Akik magyar nyelven szeretnének többet olvasni a belsőpontos algoritmusokról, azoknak Darvay [82], de Klerk, Roos és Terlaky [204] és Terlaky [337] jegyzeteit, illetve könyvfejezetét, Klafszky és Terlaky [199] ellipszoid algoritmusról, Terlaky [336] Karmarkar-algoritmusról szóló összefoglaló cikkét ajánljuk. A Darvay [83], Pólik és Terlaky [283], illetve az Illés és Nagy [170] cikkek a belsőpontos algoritmusok aktuális kutatási irányaiba adnak betekintést.

26. Tömegkiszolgálási algoritmusok

A tömegkiszolgálás elmélete tömegesen előforduló igények és kiszolgálásuk problémájának matematikai modellezésével és megoldásával foglalkozik.

Tömegkiszolgálási rendszerek (TKR-ek) a gyakorlatban az élet szinte minden területén előfordulnak, rendkívül változatos megjelenési formájuk ellenére működésük sok esetben közös matematikai modellekkel írható le. A matematikai modellek kidolgozására és vizsgálatára már nagyon korán, a múlt század elején sor került. Az első eredmények Erlang (1878-1929) dán matematikus nevéhez fűződnek, aki a koppenhágai telefontársaságnál dolgozott. Ezek a vizsgálatok a telefonhálózatok gyors fejlődésének következtében váltak fontossá. Innen ered az akkor kialakult és a mai napig használt terminológia is, amely sok tekintetben kötődik a telefonnal kapcsolatban megszokott fogalmakhoz (csatorna, hívás, foglaltság, hívás visszautasítása, sorhosszúság, stb.).

A TKR-ekben közös az igények megjelenése, igények kiszolgálása (tágabb értelemben is), sorok képződése, várakozás (illetve elutasítás), igények távozása a rendszerből. Egy TKR-be többféle igény érkezhet, de keletkezhet új igény magában a rendszerben is. Az igények kiszolgálása a megfelelő kiszolgáló egységen történik. A kiszolgáló egység előtt lehetnek várakozási (sorbanállási) helyek, ahova a beérkezett, de a kiszolgáló egységnél még kiszolgálásra nem kerülő (vagy a kiszolgáló egységről kikerülő, s arra újból visszatérő) igények kerülnek. Ha egy TKR-ből anélkül távozhatnak igények, hogy teljes kiszolgálást nyertek volna, akkor a rendszt **veszteségesnek** nevezzük.

Általában elvonatkoztathatunk a vizsgált rendszerek, az igények és a kiszolgálás konkrét jellegétől. Ez annak köszönhető, hogy a vizsgálatok gyakran bizonyos időpontok halmazához (igények megjelenése a rendszerben, kiszolgálások kezdete, illetve befejezése stb.) kötődnek, ezért el lehet tekinteni az igények és kiszolgálások valós hátterétől. Az igények beérkezési időpontjai, a kiszolgálásukhoz szükséges idők stb. általában sztochasztikus jellegűek, ezért az ilyen rendszerek elemzése a valószínűségszámítás eszközeit és módszereit igényli.

Tömegkiszolgálási rendszerek matematikai leírásához a következőket kell megadni:

- A *beérkezési folyamat* leírja a rendszerbe érkező igényeket, de absztrakt igények is lehetnek, továbbá igények képződhetnek a TKR-en belül is. A rendszerbe lépő igények száma és eloszlása függhet a rendszer állapotától. A beérkezési folyamatok általában az egymás után érkező igények közötti időintervallumokkal és az igényelt kiszolgálási időkkel írhatók le (utóbbi, pl. telekommunikációs rendszerek esetén, jelentheti az átviendő adatmennyiség nagyságát). A beérkezések között eltelt idők és a kiszolgálási

idők nagysága általában véletlen mennyiség, de lehet determinisztikus is. Az egymás utáni igények között eltelt időktől (illetve a kiszolgálási időktől) egyfajta homogenitást – azonos eloszlást – szokás megkövetelni.

- A vizsgált rendszer *struktúrája* (*topológiája*) megadja azt, hogy az igények a TKR kiszolgálóegységei között hogyan vándorolhatnak, illetve távozhatnak a rendszerből. A vizsgált rendszer struktúrájához hozzátartozik az egyes kiszolgáló egységekhez tartozó várakozási helyek (puffer terület) nagysága is.
- *Kiszolgálási algoritmusok* (*kiszolgálási elvek*) határozzák meg az egyes igények kiszolgálási szabályait. A TKR-ek hatékonysága jelentős mértékben függ a kiszolgálási algoritmusoktól, ha a kiszolgáló eszközhöz való hozzáférés egymással konkuráló felhasználók mellett történik.
- A TKR meghatározandó *jellemzői, mutatói,* melyek összefüggnek a vizsgálat céljaival. Megemlítünk néhány jellemző példát. *Várakozási rendszereknél* az átlagos várakozási idő a kiszolgálás megkezdéséig, a rendszerben eltöltött átlagos idő, átlagos sorhosszúság, a sorhosszúság eloszlása; *elutasításos rendszereknél* az elutasítás valószínűsége, az elutasítások átlagos száma; *hatékonysági mértékek*, mint pl. az átlagos kihasználtság, kiszolgált igények átlagos száma stb.

26.1. Tömegkiszolgálási rendszerek működésének leirása

Egy rendszer működésének, időben változó jellemzőinek leírása azt jelenti, hogy a beérkezési folyamatból és a rendszer működését meghatározó tulajdonságokból kiindulva megadjuk azt az algoritmust, amely az idő függvényében (vagy csak a megadott időpontokban) előállítja a vizsgált jellemzőket. Látni fogjuk, hogy ezeknek a – különböző jellemzők meghatározásánál különböző – számítási eljárásoknak a megadása már a legegyszerűbb rendszerek és egyszerű kiszolgálási algoritmusok esetén is bonyolult feladatot jelenthet. A beérkezési folyamat és ezzel együtt a rendszer sztochasztikus jellegének, statisztikai törvényszerűségeinek vizsgálatával itt külön nem foglalkozunk. A feladat jellegét néhány konkrét példával mutatjuk be.

A 26.1. ábra egy egyszerű egycsatornás, egy kiszolgáló egységből és korlátlan számú várakozási helyből álló kiszolgálási rendszert (a szakirodalomban alkalmazott jelöléssel $G|G|1|\infty$ típusú rendszert) mutat be: egy fajta igény érkezik a rendszerbe; a kiszolgáló egység előtt (végtelen hosszúságú) sor képződhet; a kiszolgálást egy kiszolgáló egység végzi és az igény a kiszolgálás után távozik a rendszerből.(A tömegkiszolgálási rendszerek leírására általánosan használt a Kendall által bevezetett jelölés, amely $A|B|m|n$ alakú. Ebben A a belépő folyamat jellegére utal (M, ha két belépés között eltelő idő exponenciális; G, ha általános eloszlású); B a kiszolgálási idő eloszlása (B lehetséges értékei ugyanazok lehetnek, mint A-é); m a kiszolgáló eszközök száma; n a rendszerben levő tartózkodási helyek száma (ha nem adjuk meg vagy a ∞ jelet használjuk, akkor erre nincs korlátozás).

Ha egy igény a rendszerbe érkezve üresen találja azt, akkor az igény kiszolgálása azonnal megkezdődik, különben az igény a várakozási sorba kerül. Ha egy igény kiszolgálása befejeződik a kiszolgáló egységen és a várakozási sor nem üres, akkor a sorban állók közül a korábban beérkezett igény kerül kiszolgálásra. Itt fontos megjegyezni, hogy a kiszolgálás módja és a várakozó igények közül való választás sokféle elv szerint történhet (ld. lentebb).

26.1. ábra. Egycsatornás kiszolgálási rendszer.

A vizsgálat kezdete utáni első igény a rendszerbe az X_0 időpontban érkezik be. Jelölje X_i $(i = 1, 2, \ldots)$ az egymás után a rendszerbe érkező i-edik és $(i+1)$-edik igény beérkezési időpontjai között eltelt időt, Y_i $(i = 1, 2, \ldots)$ pedig az i-edik igény kiszolgálásához szükséges időt. Ekkor, ha ismert a rendszer kiinduló állapota (pl. a rendszer üres az első igény beérkezéséig), akkor az $X_0, (X_i, Y_i)$ $(i = 1, 2, \ldots)$ sorozat – a rendszer *beérkezési folyamata* – tetszőleges $t > 0$ időpontban meghatározza a rendszer állapotát és a feladat éppen az, hogy – a rendszer beérkezési folyamatából kiindulva – eljárást adjunk a rendszer különböző jellemzőinek meghatározására tetszőleges t időpontban. Az egyszerűség kedvéért a továbbiakban feltesszük, hogy X_i és Y_i pozitív mennyiségek.

Jelölje $t_1, t_2, \ldots$ és $s_1, s_2, \ldots$ az egymás után a rendszerbe érkező igények beérkezési időpontjait, illetve azokat az időpontokat, amikor az igények kiszolgálása befejeződött a rendszerben (az igények távozási időpontjait). Világos, hogy $X_i = t_{i+1} - t_i$, $i = 1, 2, \ldots$, így a beérkezési időpontokra

$$t_{i+1} = X_0 + \cdots + X_i, \qquad i \geq 1 \,.$$

Megjegyezzük, hogy a távozási időpontokra, melyek a beérkezési időpontokon kívül az egyes kiszolgálási időktől és a rendszer összes tulajdonságától függhetnek, általában nem adható hasonlóan egyszerű formula. Az alábbiakban nézzük meg, hogy bizonyos, a rendszer működését időben leíró jellemzőket milyen algoritmus szerint lehet kiszámítani.

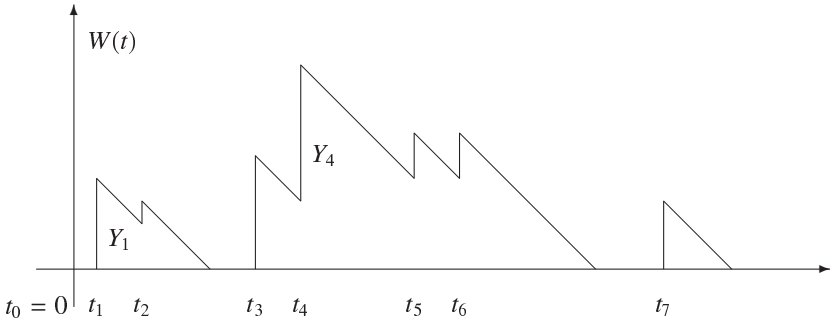
A $W(t)$ virtuális várakozási idő meghatározása. Jelölje $W(t)$ tetszőleges $0 \leq t < \infty$ időpontban a virtuális várakozási időt, vagyis azt az időt, amennyit egy igénynek a kiszolgálás megkezdéséig várakopznia kellene, ha a t időpontban érkezne a rendszerbe. Feltesszük, hogy a $t = 0$ időpontban a rendszer üresen kezdett dolgozni, vagyis $W(0) = 0$ és $W(t)$-t balról folytonosnak definiáljuk, ezáltal az ugrási pontokban egyértelműen meghatározzuk. A virtuális várakozási idő változását szemlélteti a 26.2. ábra.

Látható, hogy a $W(t)$ virtuális várakozási időnek ugrása van a t_i időpontokban, melynek nagysága $W(t_i + 0) - W(t_i) = Y_i (i = 1, 2, \ldots)$.

$W(t)$ értéke a $t_i, i \geq 1$ ugráshelyeken a következő rekurzióval adható meg $(W_i = W(t_i),\ W_1 = 0)$:

$$W(t_{i+1}) = [(W_i + Y_i) - X_i]^+, \quad i \geq 1 \,,$$

ahol $x^+ = \max\{x, 0\}$, $x \in \mathbb{R}$. Könnyen látható az is, hogy két ugráshely között $W(t)$ értékére érvényes a következő összefüggés:



26.2. ábra. Virtuális várakozási idő.

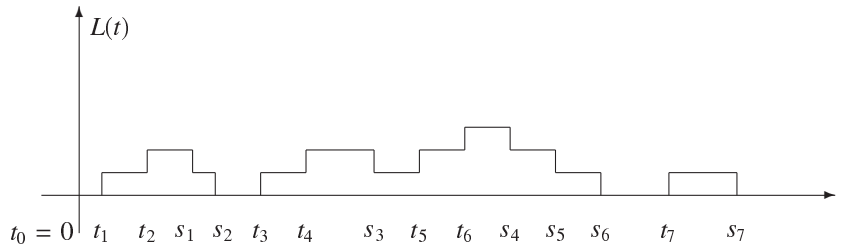
$$W(t) = 0, \qquad 0 \le t \le t_1 ,$$
$$W(t) = [W_i + Y_i - (t - t_i)]^+, \qquad t_i < t \le t_{i+1}, \quad i \ge 1 .$$

VIRTUÁLIS-VÁRAKOZÁSI-IDŐ(t)

```
 1  i ← 1
 2  t(1) ← X(0)
 3  τ ← X(0)
 4  W(1) ← 0
 5  if t ≤ τ
 6     then return W(t) = 0
 7  while t(i) < t
 8        do i ← i + 1
 9           τ ← τ + X(i − 1)
10           t(i) ← τ
11  n ← i − 1
12  for j ← 1 to n
13       do W(j + 1) ← max{0,  W(j) + Y(j) − X(j)}
14  W(t) ← max{0,  W(n) + Y(n) − (t − t(n))}
15  return W(t)
```

Az $L(t)$ sorhosszúság meghatározása. Tekintsük most a t időpontban a rendszerben tartózkodó igények $L(t)$, $0 \le t < \infty$ számát ($L(0) = 0$). Feltesszük, hogy $L(t)$ balról folytonos. $L(t)$ időbeli változását a 26.3. ábra szemlélteti.

Látható, hogy a rendszerben tartózkodó igények $L(t)$ száma a virtuális várakozási idő kiszámításához hasonló egyszerű algoritmussal nem határozható meg, mivel ebben az esetben $L(t)$ ugrásai nem csak a $t_1, t_2, \ldots$ beérkezési időpontokban, hanem az igények $s_1, s_2, \ldots$ távozási időpontjaiban is bekövetkeznek. Egyszerűsíti a helyzetet, hogy elegendő meghatározni $L(t)$-t a foglaltsági periódusokon, mivel rajtuk kívül értéke $L(t) = 0$.



26.3. ábra. Sorhosszúság.

Jelölje $N(t),\ t > 0$ a t időpontig a rendszerbe érkező igények számát, azaz legyen

$$N(t) = \sum_{k=1}^{\infty} I(t_k < t)\,,$$

ahol I az esemény indikátorfüggvénye (az értéke 1, ha az esemény bekövetkezik és 0 egyébként). Legyen

$$k_1 = 1\,,$$
$$k_{i+1} = \min\{k:\ X_{k_i} + \cdots + X_k > Y_{k_i} + \cdots + Y_k,\ \ k > k_i\}, \quad i \geq 1\,.$$

Látható, hogy ekkor a foglaltsági periódusok a következő alakban írhatók fel:

$$\Delta_i = (t_{k_i}, t_{k_i} + Y_{k_i} + \cdots + Y_{k_{i+1}-1}], \quad i \geq 1\,.$$

A Δ_i foglaltsági periódus kezdetén mindig fennáll

$$L(t_{k_i}) = 0, \qquad L(t_{k_i} + 0) = 1\,.$$

Minthogy a Δ_i foglaltsági periódusban pontosan $k_{i+1} - k_i$ igény kerül kiszolgálásra és a beérkezési és a távozási időpontok ismertek, ezért tetszőleges $i \geq 1$ mellett

$$t_k = t_{k_i} + X_{k_i} + \cdots + X_k\,,$$
$$s_k = t_{k_i} + Y_{k_i} + \cdots + Y_k, \quad k_i \leq k \leq k_{i+1} - 1\,,$$

és így az $L(t),\ t \in \Delta_i$ sorhosszúság a következő módon határozható meg:

$$L(t) = \sum_{k=k_i}^{N(t)} I(t_k \leq t < s_k), \quad t \in \Delta_i\,,$$

Megjegyzés. Ha bevezetjük a rendszerből a $t\ \ (t \geq 0)$ időpontig eltávozott igények $M(t)$ számát $(M(0) = 0)$, akkor

$$M(t) = \sum_{k=1}^{\infty} I(s_k < t)\,,$$

akkor $L(t)$ felírható

$$L(t) = N(t) - M(t), \quad t \geq 0$$

alakban.

A következő algoritmus meghatározza a t időpontban a rendszerben tartózkodó igények számát. Az első rész generálja az igények belépési időpontjait, a második rész meghatározza az egyes foglaltsági periódusok befejezésének időpontját és a következő foglaltsági periódus kezdő igényét. Ha a t időpontig elkezdődő utolsó foglaltsági periódus t-ig befejeződik, akkor a rendszer szabad, ellenkező esetben a harmadik rész kiszámítja a t-ig belépett, illetve a t-ig kiszolgált igények számát, és e két érték különbségét képezve visszaadja az aktuális sorhosszt. A rendszer működését a $[0, T]$ intervallumon vizsgáljuk.

SORHOSSZ(T, t)

```
1                                                  ▷ Belépési pontok generálása T-ig.
2  t(1) ← X(0)
3  i ← 1
4  if t < t(1)
5    then return L(t) = 0
6  while t(i) ≤ T
7        do i ← i + 1
8           t(i) ← t(i − 1) + X(i − 1)
9                                  ▷ A foglaltsági periódusok végpontjainak meghatározása.
10 i ← 1
11 j ← 1
12 k(1) ← 1                                        ▷ A foglaltsági periódus kezdő igénye.
13 while t(k(i)) < t
14       do m ← i
15          while X(k(i)) + ··· + X(j) < Y(k(i)) + ··· + Y(j)
16                do j ← j + 1
17   n → j                  ▷ A teljes foglaltsági periódus alatt kiszolgált utolsó igény száma.
18          k(i + 1) ← j + 1                ▷ A következő foglaltsági periódus kezdő igénye.
19          s(k(i + 1) − 1) ← t(k(i)) + Y(k(i)) + ··· + Y(j)
20                                                          ▷ A foglaltsági periódus vége.
21          i ← i + 1
22                                  ▷ Feltételezzük, hogy a ciklusváltozó értéke megőrződik.
23 if s(k(i) − 1) ≤ t
24   then return L(t) = 0
25                                                ▷ Az utolsó foglaltsági periódus teljes volt.
26 n ← k(i − 1)                                   ▷ Visszaállunk a foglaltsági periódus kezdetére.
27  ▷ A belépések számának meghatározása az utolsó nem teljes foglaltsági periódusban.
28 j ← 0
29 while t(n + j) ≤ t
30       do j ← j + 1
31 J ← j
32 ▷                                  A kiszolgált igények számának meghatározása az utolsó
33   nem teljes foglaltsági periódusban.
34 ℓ ← 0
```

```
35  while t(n) + Y(n) + ··· + Y(n + ℓ) ≤ t
36        do ℓ ← ℓ + 1
37  L ← ℓ
38  return L(t) = J − L
```

Bonyolultasbb rendszer (pl. többcsatornás, többféle igényt esetleg más kiszolgálási algoritmussal kiszolgáló rendszer) esetén a rendszert csak több paraméter egyidejű meghatározásával adhatjuk meg. Ennek megfelelően a rendszer jellemzőinek leírása is lényegesen bonyolultabb eljáráshoz vezethet. Sokszor könnyebben levezethető eljáráshoz jutunk, ha a további vizsgálatok számára elegendő, speciális időpontokban tekintjük a rendszert, mint például az igények beérkezési vagy távozási időpontjaiban. Erre vonatkozóan megadunk néhány példát a fejezet végén.

A fenti eljárás bármely X_0, (X_i, Y_i), $i = 1, 2, \ldots$ (akár determinisztikus) beérkezési folyamat esetén megadja a rendszer t időpontbeli $W(t)$ és $L(t)$ jellemzőinek értékét, de megfelelő módon eljárva megadható a rendszer többi jellemzője is az idő függvényében.

Sztochasztikus rendszerek esetén az X_i és Y_i mennyiségek véletlen mennyiségek és leggyakrabban homogén módon viselkednek: az (X_i, Y_i), $i = 1, 2, \ldots$ valószínűségi változók függetlenek és azonos eloszlásúak, melyek függetlenek az X_0 valószínűségi változótól. Ha X_0 és X_i, $i \geq 1$ eloszlása különbözik, késleltett esetről beszélünk. Azt is szokás feltenni, hogy minden egyes i mellett az X_i és Y_i, $i = 1, 2, \ldots$ valószínűségi változók függetlenek. Ez a feltevés nem zárja ki a determinisztikus esetet, vagyis azt, hogy X_i és Y_i 1 valószínűséggel konstans értéket vegyenek fel.

Ilyen rendszerek vizsgálatánál a kérdés az, hogy a rendszer jellemzőinek mikor létezik és hogyan határozható meg a határeloszlása, illetve, amire csak speciális esetekben tudunk válaszolni, hogyan határozhatók meg a jellemzők időtől függő eloszlásai. A jellemzők numerikus meghatározásának általános módszereként tárgyaljuk a fejezet végén a szimulációs módszert, melynél a rendszer jellemzőinek kiszámítását szolgáló (a fenti két példában is bemutatott) algoritmusok alapvető szerepet játszanak. Az analitikus megközelítés általában csak erősen korlátozottan és akkor is csak a konkrét esetre kidolgozva használható.

Tekintsük a korábban vizsgált $G|G|1|\infty$ rendszer esetén a következő jellemzőket (ezek a jellemzők más TKR-ekre is értelmezhetők):

A rendszerben tartózkodó igények átlagos száma:

$$\overline{L}_T = \frac{1}{T} \int_0^T L(t)dt \, ;$$

Egy igényre jutó átlagos várakozási idő:

$$\overline{W}_T = \frac{1}{\overline{L}_T} \frac{1}{T} \int_0^T W(t)dt \, ;$$

A következő algoritmus a rendszerben tartózkodó igények átlagos számát határozza meg. Felhasználja a sorhosszra vonatkozó algoritmus által generált belépési időpontokat $(t(j))$, az egyes foglaltsági periódusok végpontjait $(s(k(i+1)-1))$ és a t időpontig elkezdődött foglaltsági periódusok számát (m).

Az első rész generálja az egyes igények kiszolgálása befejeződéseinek időpontjait. A

második rész az egymás után következő belépési és befejeződési pontokon végigmenve elvégzi a szükséges összegzést és kezeli az utolsó (nem teljes) foglaltsági periódust.

ÁTLAGOS-SORHOSSZ(t)

```
 1  ▷ Az egyes igények kiszolgálása befejeződési időpontjainak előállítása
      (a t időpontig m foglaltsági periódus kezdődik el).
 2  j ← 1
 3  k(1) ← 1
 4  s(1) ← t(1) + Y(1)
 5  for i ← 1 to m
 6      while t(k(i)) + Y(k(i)) + ··· + Y(j) < t(k(i + 1))
 7            do s(j + 1) ← s(j) + Y(j)
 8               j ← j + 1
 9                                                  ▷ Összegzés.
10  S ← 0
11  i ← 1                                  ▷ A foglaltsági periódus sorszáma.
12  k(1) ← 1
13  τ ← t(1)                               ▷ Az aktuális pozició pointere.
14  a ← 1                                  ▷ A belépési pont indexe.
15  b ← 1                                  ▷ A befejeződési pont indexe.
16  while (t(k(i)) < t) ∧ (s(k(i + 1) − 1) < t)
17        do d ← 1          ▷ Jelenlévő igények száma egy foglaltsági perióduson belül.
18           while s(b) ≤ s(k(i + 1) − 1)
19                 do if t(a + 1) < s(b)
20                       then S ← S + d · (t(a + 1) − τ)
21                            d ← d + 1
22                            a ← a + 1
23                            τ ← t(a)
24                    if s(b) < t(a + 1)
25                      then S ← S + d · (s(b) − τ)
26                            d ← d − 1
27                            b ← b + 1
28                            τ ← s(b)
29                    i ← i + 1
30                                  ▷ Az utolsó foglaltsági periódus befejeződik t-ig.
31  if (s(k(m + 1) − 1) ≤ t) ∨ (t(k(m + 1)) = t)
32    then return L̄ = S/t
33                                  ▷ Az utolsó foglaltsági periódus túlnyúlik t-n.
34  τ1 ← t(a)
35  τ2 ← t(a)
36  d ← 1
```

```
37  while τ1 ≤ t
38        do if τ1 ≤ t ≤ τ2
39              then S ← S + d · (t − τ1)
40                   return L̄ = S/t
41           if t(a + 1) < s(b)
42              then τ1 ← τ2
43                   τ2 ← t(a + 1)
44                   S ← S + d · (τ2 − τ1)
45                   d ← d + 1
46                   a ← a + 1
47              else τ1 ← τ2
48                   τ2 ← s(b)
49                   S ← S + d ∗ (τ2 − τ1)
50                   d ← d − 1
51                   b ← b + 1
```

Foglaltsági periódusok eloszlása: azoknak az egymás utáni leghosszabb időszakaszoknak az eloszlása, amikor a kiszolgáló egység foglalt. (A 26.3. ábrán ezek a periódusok: (t_1, s_2), (t_3, s_6), (t_7, s_7).)

Szabad periódusok eloszlása: azoknak az egymás utáni leghosszabb időszakaszoknak az eloszlása, amikor a kiszolgáló egység üres. (Ezek a periódusok a 26.3. ábrán: (t_0, t_1), (s_2, t_3), (s_6, t_7).)

Egyes vizsgálatokban alapvető kérdés, hogy létezik-e a rendszerben tartózkodó igényeknek számának

$$\pi_k = \lim_{t\to\infty} P(L(t) = k), \quad k = 0, 1, \ldots, \qquad \sum_{k=0}^{\infty} \pi_k = 1$$

állandósult (stacionárius) eloszlása és az hogyan határozható meg.

Hasonló kérdés vethető fel a rendszerben tartózkodó igények átlagos számával, illetve az egy igényre jutó átlagos várakozási idővel kapcsolatban (létezik-e $T \to \infty$ mellett az $\overline{L}_T \to \overline{L}$, illetve $\overline{W}_T \to \overline{W}$ határérték), míg a foglaltsági/szabad periódusok esetében a periódushosszak eloszlása érdekes. Ezekkel a kérdésekkel jelen fejezetben részletesen nem foglalkozunk, mindössze egy egyszerű, de a gyakorlat számára fontos TKR esetében adunk formulákat.

26.2. Klasszikus tömegkiszolgálási rendszer

Tekintsük az ún. klasszikus tömegkiszolgálási rendszert, amelyre a határértékek léteznek és az eloszlások meghatározhatók (a tömegkiszolgálás-elméletben ennek a rendszernek a szokásos jelölése $M/M/1$).

Legyen a beérkezési idők közös eloszlása $\lambda > 0$, míg a kiszolgálási időké $\mu > 0$ paraméterű exponenciális eloszlás. Megjegyezzük, hogy e feltétel szerint az időegységre jutó beérkezések átlagos száma λ, míg állandó terhelés mellett a kiszolgáló egységen kiszolgált igények egy időegységre jutó átlagos száma μ. Feltesszük, hogy $\rho = \lambda/\mu < 1$. Eb-

ben az esetben a t ($t \geq 0$) időpontig a rendszerbe érkező igények $N(t)$ száma egy $\lambda > 0$ intenzitású $N(t)$, $t \geq 0$ Poisson-folyamattal azonosítható. Ez utóbbi egy olyan sztochasztikus folyamatot jelent, amelyre tetszőleges $x_0 = 0 < x_1 < x_2 < \cdots$ sorozat mellett az $N(x_{j+1}) - N(x_j)$, $i = 0, 1, \ldots$ növekmények független valószínűségi változók és Poisson-eloszlásúak $\lambda(x_{j+1} - x_j)$ paraméterrel, vagyis

$$P(N(x_{j+1}) - N(x_j) = k) = \frac{[\lambda(x_{j+1} - x_j)]^k}{k!} e^{-\lambda(x_{j+1} - x_j)}, \qquad k = 0, 1, \ldots .$$

Erre a tömegkiszolgálási rendszerre fennállnak a következő összefüggések.

Esetünkben a rendszerben tartózkodó igények határeloszlása létezik, mely megadható a következő alakban:

$$\pi_0 = 1 - \rho, \quad \pi_k = \pi_0 \rho^k, \quad k = 0, 1, \ldots .$$

Látható, hogy a ρ mennyiségnek igen fontos jelentése van, megadja határértékben annak a valószínűségét, hogy a rendszer foglalt.

A rendszerben tartózkodó igények átlagos számának, illetve az egy igényre jutó átlagos várakozási időnek 1 valószínűséggel létező határértékére igaz

$$\overline{L} = \frac{\rho}{1 - \rho}, \quad \text{illetve} \quad \overline{W} = \frac{1}{1 - \rho} \frac{1}{\mu} = \frac{1}{\lambda - \mu}.$$

A szabad periódus eloszlása triviálisan adódik, éppen a beérkezési időközök közös λ paraméterű exponenciális eloszlása. A foglaltsági periódus eloszlásának sűrűségfüggvénye ugyancsak meghatározható, bár az eredmény csak speciális matematikai függvénnyel fejezhető ki:

$$h(x) = \begin{cases} \frac{1}{x\rho^{1/2}} e^{-(\lambda+\mu)x} I_1(2x(\lambda\mu)^{1/2}), & \text{ha } x > 0, \\ 0, & \text{ha } x \leq 0, \end{cases}$$

ahol az $I_1(x)$ függvény az elsőfajú Bessel-függvényt jelöli.

26.3. Kiszolgálási algoritmusok

Egy TKR vizsgálatát alapvetően az határozza meg, hogy a rendszer működését milyen szempontból elemezzük. Általában olyan mutatók vizsgálata a cél, amelyek valamilyen értelemben jellemzik a rendszer működésének hatékonyságát. Ezek a vizsgálatok nem csak egy működő rendszer elemzésére és hatékonyságának növelésére irányulhatnak, hanem már a tervezés fázisában is fontos támpontokat jelenthetnek.

Egy bonyolultabb, több kiszolgáló egységből álló tömegkiszolgálási hálózat működésének hatékonysága nem csak attól függ, hogy az egyes elemei milyen gyorsan és jól működnek, hanem sok esetben szerepe van annak is, hogy a kiszolgálás milyen algoritmus szerint történik. ***Kiszolgáló algoritmuson*** olyan kiszolgálási eljárásokat (szabályokat) értünk, amelyek tetszőleges időpontban megmondják, hogy a rendszerben lévő igények közül melyiket kell éppen kiszolgálni, milyen sorrendben és milyen hosszú ideig tart a kiszolgálás.

A kiszolgálási algoritmusokkal szemben támasztott követelmények különbözőek lehetnek. Adatátvitel esetén lehet a kiszolgálás a késleltetéssel szemben érzékeny (pl. telefon, videokonferencia), de más és más lehet egy esetleges adatvesztés következménye is (pl.

egy számítógépes program esetén az átvitt anyag használhatatlanná válhat, ugyanakkor beszédátvitelnél esetleg csak kismértékű minőségromlással jár). Ennek megfelelően a kiszolgálási algoritmusoknak általában meghatározott kritériumoknak kell eleget tenniük. Megjegyezzük, hogy a kiszolgálási kapacitás növelése a kapacitás szempontjából sok esetben nem hatékony, viszont a megfelelő kiszolgálási algoritmus kialakításával jelentős eredményeket lehet elérni.

26.3.1. A leggyakrabban előforduló kiszolgálási algoritmusok

FIFO (FIRST-IN-FIRST-OUT). Ha a kiszolgáló egység üres az igény beérkezésének időpontjában, akkor a kiszolgálása haladéktalanul megkezdődik és egészen addig tart, amíg a teljes kiszolgálás befejeződik. Ha a beérkező igény a kiszolgáló egységet nem találja üresen, akkor várakozási sorba kerül és a továbbiakban az igények kiszolgálása a beérkezés sorrendjében történik.

LIFO (last-in-first-out). A kiszolgáló egységnél a sorbanálló igények kiszolgálása úgy történik, hogy egy igény kiszolgálása befejeződése esetén a rendszerbe utoljára belépett igény kiszolgálása kezdődik meg és tart a teljes kiszolgálásig.

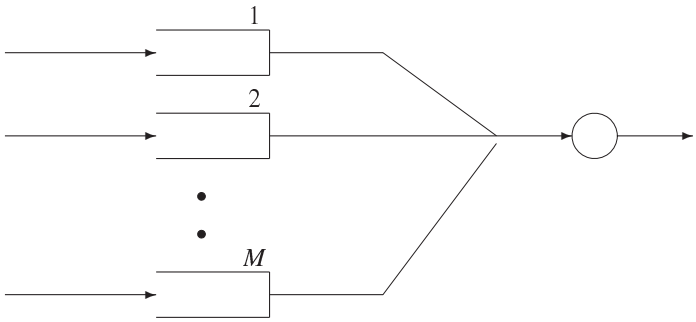
Az FCFS (first-come-first-served, az először érkezett lesz először kiszolgálva) és az LCFS (LAST-COME-FIRST-SERVED, AZ UTOLJÁRA ÉRKEZETT LESZ ELŐSZÖR KISZOLGÁLVA) elvek hasonlítanak a FIFO és LIFO elvekhez: a kiszolgáló egységnél a sorbanálló igények közül az az igény kerül először kiszolgálásra, amelyik előbb (ill. később) érkezett. E két elvnél a rendszerből való távozás sorrendje függhet (az alábbiakban is ismertetett) más kiszolgálási elvektől is. Minthogy sok rendszerre (például egycsatornás rendszerek esetén) az eredmény ugyanaz, ezért néha a FIFO és FCFS (LIFO és LCFS) kiszolgálási eljárásokat egymás szinonimájaként használják, de jó tudni a közöttük levő különbséget.

VÉLETLEN-SORREND (random). A sorbanállók közül a választás véletlenszerűen, pl. egyforma valószínűséggel, azaz egyenletes eloszlás szerinti sorsolással történik és a kiszolgálás az igény teljes kiszolgálásáig tart.

PRIORITÁSOS RENDSZEREK. Ezekben a rendszerekben az igényeket különböző véges M számú osztályba soroljuk (lásd 26.4. ábra). A rendszerbe lépő minden egyes igény egy, a típusának (osztályának) megfelelő indexet (számot), ún. prioritási indexet kap. Az igény prioritása akkor nagyobb, amikor a prioritási indexe nagyobb. Amikor a kiszolgáló egység felszabadul, akkor a sorbanálló igények közül a magasabb prioritású igény kerül kiszolgálásra. Azonos típusú (prioritású) igények közül a választás a fenti elvek valamelyike szerint történhet. Az osztályba sorolás és a prioritások hozzárendelése az egyes osztályokhoz egyfajta eszközt jelent a rendszer hatékonyságának optimalizálására.

MEGSZAKÍTÁSOS PRIORITÁSOS RENDSZER (ABSZOLÚT PRIORITÁSOS RENDSZER, PREEMPTIVE SERVICE DISCIPLINE). Ha egy igény kiszolgálási ideje alatt egy magasabb prioritású igény érkezik a rendszerbe, akkor a kiszolgálás alatt lévő alacsonyabb prioritású igény kiszolgálása azonnal befejeződik és a kiszolgáló egység a magasabb prioritású igény kiszolgálását kezdi meg. A már lezajlott kiszolgálás különböző módokon vehető figyelembe:

- Magasabb prioritású igény belépése esetén az alacsonyabb prioritású kiszolgálása megszakad és az összes magasabb prioritású igény kiszolgálása után folytatódik. Az elvégzett kiszolgálást figyelembe vesszük, a kiszolgálási idő a már elvégzett munkával csökken.



26.4. ábra. Prioritásos rendszer.

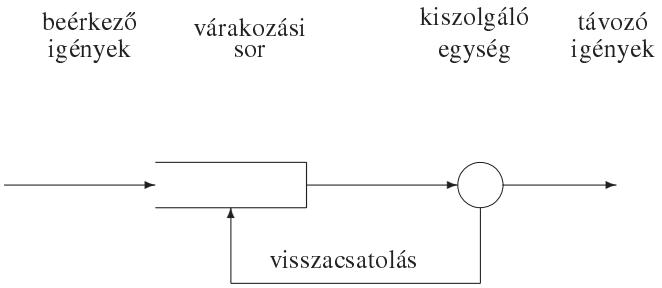
- A kiszolgálás az előbbi pontban leírt módon zajlik, de az alacsonyabb prioritású igényeknél a megszakításig elvégzett munkát nem vesszük figyelembe, a kiszolgálás elölről kezdődik.
- Magasabb prioritású igény belépése esetén az alacsonyabb prioritású kiszolgálása megszakad és az igény elvész.

NEMMEGSZAKÍTÁSOS PRIORITÁSOS RENDSZER (RELATÍV PRIORITÁSOS RENDSZER, NONPREEMPTIVE SERVICE DISCIPLINE). A különbség az előző kiszolgálási elv és eközött abban áll, hogy ha egy igény kiszolgálási ideje alatt egy magasabb prioritású igény érkezik a rendszerbe, akkor az új igény kiszolgálása csak azután kezdődik meg, hogy az éppen kiszolgálás alatt lévő igény kiszolgálása befejeződik. A relatív prioritásos rendszer jellemző példái az LPT (LONGEST-PROCESSING-TIME, LEGHOSSZABB KISZOLGÁLÁSI IDŐ) és az SPT (SHORTEST-PROCESSING-TIME, LEGRÖVIDEBB KISZOLGÁLÁSI IDŐ) kiszolgálási diszciplinák, amelyek esetén a kiszolgálás a kiszolgálási idők csökkenő (növekvő) sorrendjében történik.

A következő kiszolgálási algoritmus széles körben elterjedt a nagy teljesítményű és nagy forgalmú kiszolgáló egységeken történő kiszolgálás szabályozására. Igazi jelentőséget először a nagy teljesítményű multiprogramozású számítógépeknél nyert, de alapvető szerepet játszik a modern telekommunikációs rendszerekben is. Ebben az esetben egy igény egyszerre csak részkiszolgálást nyer és utána visszakerül a várakozási sorba, a kiszolgáló egység és a várakozási sor között egyfajta visszacsatolás van (ld. 26.5. ábra).

KÖRBEFORGÓ KISZOLGÁLÁS (ROUND ROBIN). Minden beérkező igényhez hozzárendelünk egy, akár véletlen nagyságú, kiszolgálási időt (kvantumot). Ha ez alatt az igény kiszolgálása nem fejeződik be, akkor visszakerül a várakozási sorba és ott tartózkodik, amíg a jelenlévő további igények meg nem kapják a nekik előírt kiszolgálási kvantumot. Az újabb kiszolgálásra kerülésnél egy újabb kvantum kerül meghatározásra és a kiszolgálás a leírt módon folytatódik. Ha a kvantum nagysága fix és azonos valamennyi igényre, akkor szokás ezt az elvet IDŐOSZTÁSOS (TIME-SHARING) algoritmusnak nevezni.

PROCESSZOROSZTÁSOS ALGORITMUS (PROCESSOR-SHARING, PS). A processzor egyszerre dolgozik az összes igényen, k igény esetén a kapacitásának $1/k$-adával szolgálja ki az egyes igényeket. Az időosztásos algoritmus esetén meghatározott Q kvantumot egyenlő arányban szétosztjuk a jelenlévő igények között. Tekinthetjük úgy, hogy a rendszerben ta-



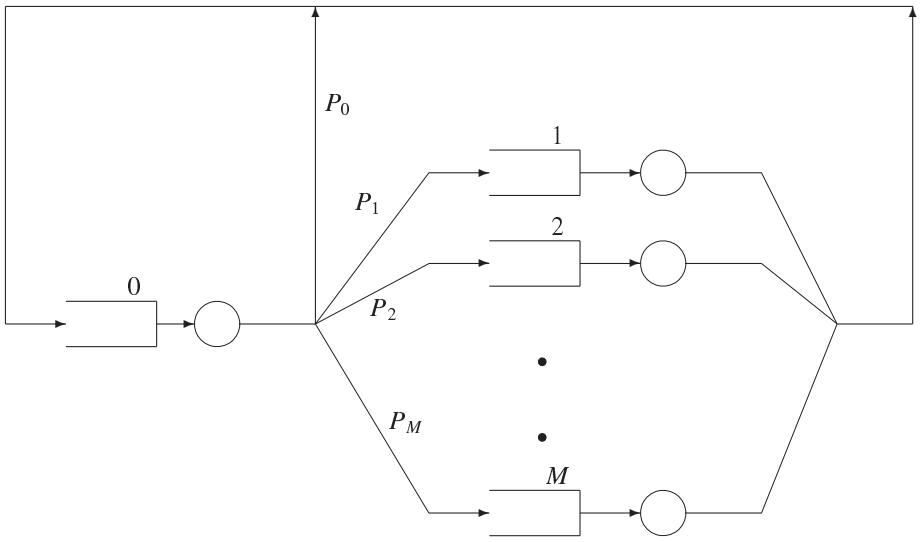
26.5. ábra. Kiszolgálás visszacsatolással.

lálható igények kiszolgálása egyszerre folyik, a kvantum végén valamennyi igény hátralévő kiszolgálási ideje a Q/k értékkel csökken. Ez a kiszolgálási szabály kedvez azoknak az igényeknek, amelyek rövid kiszolgálási időt, de gyors válaszokat igényelnek.

LEGRÖVIDEBB ELTELT IDŐ ALGORITMUS (SHORTEST-ELAPSED-TIME, SET). A különböző kiszolgálási algoritmusok gyakran párosíthatók a prioritásos kiszolgálási szabállyal. Ha a kiszolgálási időt nem ismerjük és a rövidebb kiszolgálási idejű igényeket "előnyben" akarjuk részesíteni, akkor alkalmazhatjuk a SET kiszolgálási diszciplinát a következő módon. Az aktuális igény egységnyi kiszolgálása után áttérünk az addig legkevesebb kiszolgálást kapott igényre a várakozók közül. Ha több ilyen van, akkor például a beérkezési sorrend szerint választhatunk. A rendszer prioritásosként interpretálható oly módon, hogy a k-adik prioritási szinthez rendeljük a már $k-1$ egységnyi kiszolgálást kapott igényeket. A k-adik egységnyi kiszolgálás után az igény vagy elhagyja a rendszert (ha a kiszolgálás befejeződött), vagy csatlakozik a $k+1$-edik sorhoz (ha további kiszolgálás szükséges).

LEGRÖVIDEBB HÁTRALÉVŐ IDŐ ALGORITMUS (SHORTEST-REMAINING-PROCESSING-TIME, SRPT). Amennyiben a szükséges kiszolgálási idő nagysága ismert, akkor a kiszolgálás ütemezhető ennek alapján. Egy lehetséges megoldás a fentiekben ismertetett SPT diszciplina. Az SRPT kiszolgálási elv esetében a kiszolgáló eszközön mindig az az igény van, amelynek a kiszolgálásából a legkevesebb van hátra. Tekinthetjük úgy, hogy a rendszerbe különböző igényfolyamatok lépnek be, ezeket a szükséges kiszolgálási idő határozza meg. Miután az igény a k-adik szinten megkapja a Q nagyságú kiszolgálást, csatlakozik a $k-1$-edik szinthez. Az újonnan belépő igények a kiszolgálási idejük által meghatározott sorhoz csatlakoznak.

ISMÉTLÉSES RENDSZEREK (RETRIAL SYSTEMS). Amennyiben a rendszerbe lépő igény kiszolgálását elutasítjuk, akkor az a későbbiekben újra kezdeményezheti a kiszolgálást. Ilyen például a telefonhívások megismétlése foglaltság esetén, a repülőgépek leszállása (a várakozó repülőgép körözés esetén csak egy pontban kezdheti meg a leszállási műveletet) vagy az ütközések kezelése az ALOHA-ban. Az ismétlés történhet véletlen idő múlva, vagy valamilyen szabályozott módon.



26.6. ábra. Centrális zárt rendszer.

26.4. Centrális zárt rendszerek

A számítógépek egyik egyszerűbb matematikai modelljének is tekinthető a következő, s a 26.6. ábrával szemléltetett rendszer.

A rendszerben rögzített n számú igény van állandóan (emiatt nevezzük *zártnak*) és a megadott ábra szerint vándorolhatnak az igények. Minden egyes kiszolgáló egység előtt kellő számú várakozási hely van a felmerülő (legfeljebb $n-1$) igény számára. A kiszolgáló egységeken a beérkezés sorrendjében (FIFO elv) szerint történik a kiszolgálás, a kiszolgálási idők összességükben függetlenek, s az i-edik egységen azonos $F_i(x)$, $0 \le i \le M$ eloszlásúak. Ha egy kiszolgálás befejeződik a 0-adik egységen, akkor onnan az igény p_i, $0 \le i \le M$ ($p_i \ge 0$, $p_0 + \ldots + p_M = 1$) valószínűséggel átmegy az i-edik egységre, a rendszer állapotától és a kiszolgálási időktől függetlenül. Ha pedig egy igény kiszolgálása valamely i-edik, $1 \le i \le M$, egységen ér véget, akkor onnan a 0-adik kiszolgáló egységhez kerül. A 0-adik kiszolgáló egységet ezen kitüntetett szerepe miatt *központi (centrális) kiszolgáló egységnek* nevezzük. A vizsgált rendszer ezért kapta a *centrális zárt TKR* elnevezést. Megmutatható, hogy ha az összes kiszolgálás várható értéke véges, akkor a rendszer különböző mutatóinak létezik határeloszlása (állandósult eloszlás).

Számítógépek esetén szigorúnak tűnhet az a megszorítás, hogy a rendszer zárt, vagyis állandóan n számú igény tartózkodik a rendszerben. Ha a rendszer kihasználtságát vizsgáljuk, akkor nem mindegy az, hogy milyen külső terhelés mellett tesszük, a rendszer zártsága ebben az esetben azt jelenti, hogy a rendszer külső terhelése állandó. Ekkor ha egy program futása befejeződik, akkor helyette egy másik program lép be a rendszerbe, ami a 0-adik egységről a 0-adik egységre való visszatérést jelenthet az adott esetben. Meg kell jegyezni azt is, hogy a rendszer hatékonysági mutatói (pl. CPU kihasználtság, időegységre jutó kiszolgált programok száma, stb. jelentősen függhet a különböző kiszolgálási algoritmusoktól).

Legyen $t > 0$ egy előre rögzített időpont. Az alábbi algoritmussal meghatározható a t időpontban a rendszerben az egyes egységeknél tartózkodó igények száma és az egyes igények hátralévő kiszolgálási ideje. Az algoritmus leírásához vezessük be a következő jelöléseket:

$L0(i)$, $Lt(i)$, $L(i)$, $0 \le i \le M$ – a vizsgálat kezdetén (a 0 időpontban, a t időpontban, illetve a vizsgálat aktuális időpontjaiban a rendszer egyes egységein tartózkodó igények száma;

$X0(i, j)$, $Xt(i, j)$, $1 \le j \le n$, $0 \le i \le M$ – az előzőekhez hasonló időpontokban a rendszer egyes egységein lévő és a beérkezés sorrendjében tekintett igények hátralévő kiszolgálási idejei;

$T(k)$, $k = 1, 2, \ldots$ – az az időpont, amikor egy (vagy egyszerre több) igény kiszolgálása valahol befejeződik a rendszerben és ez sorrendben a k-adik ilyen időpont a rendszer működésének (0 időpontbeli) elkezdése óta;

generate $X^{(i)}$, illetve *generate* J – az $F_i(x)$ eloszlásfüggvény, illetve a $P(J = i) = p_i$, $1 \le i \le M$ eloszlás szerinti véletlen szám generálása, vagy determinisztikus esetben az előre megadott adatsorokból a soron következő érték kiválasztása.

CENTRÁLIS-ZÁRT-RENDSZER-ÁLLAPOTA(t, M)

```
1  for i ← 0 to M                                   ▷ Kezdeti értékek beállítása.
2      do L(i) ← L0(i)
3         for j ← 1 to n
4             do X(i, j) ← X0(i, j)
5  K ← 0
6  T(K) ← 0
7  x ← min(X(i, 1) : X(i, 1) > 0 ∧ 0 ≤ i ≤ M
8  ▷ Időpont, amikor először fejeződött be egy igény kiszolgálása.
9  while t ≤ T(K) + x                              ▷ A t időpontig tartó vizsgálat ciklusa.
10        do K ← K + 1
11           T(K) ← T(K) + x
12           for i ← 0 to M
13 ▷ A kiszolgáló egység(ek) megjelölése, ahol a T(K) időpont után
     először fejeződik be kiszolgálás.
14              do N(i) ← 0
15                 if X(i, 1) = x
16                   then N(i) ← 1
17       X(i, 1) > xif                              ▷ A hátralévő kiszolgálási idő csökkentése.
18                   then X(i, 1) ← X(i, 1) − x
```

```
19  ▷ A megjelölt kiszolgáló egységekre a T(K) + x utáni állapot beállítása:
20        if N(0) = 1
21          then L(0) ← L(0) − 1
22               m ← generate J
23               y ← generate X^(m)                          ▷ a CPU-n;
24               L(m) ← L(m) + 1
25               X(m, L(m)) ← y
26               if L(0) ≥ 1
27                 then for j ← 1 to L(0)
28                           do X(0, j) ← X(0, j + 1)
29               X(0, L(0) + 1) ← 0
30        for i ← 1 to M
31               do if N(i) = 1
32                     then L(i) ← L(i) − 1
33      z ← generate X^(0)                                   ▷ A többi kiszolgáló egységen.
34                           L(0) ← L(0) + 1
35                           X(0, L(0)) ← z
36                           for j ← 1 to L(i)
37                               do X(i, j) ← X(i, j + 1)
38                           X(i, L(i) + 1) ← 0
39        x ← min(X(i, 1) : 0 ≤ i ≤ M ∧ X(i, 1) > 0)
40  ▷ A következő változás időpontjának meghatározása.
41  for i ← 0 to M
42      do Lt(i) ← L(i)
43         Xt(i, 1) ← max(0, X(i, 1) − (t − T(K)))
44         for j ← 2 to n                                    ▷ A t időpontbeli állapot beállítása.
45             do Xt(i, j) ← X(i, j)
46  return (Lt(i), 0 ≤ i ≤ M), (Xt(i, j), 1 ≤ j ≤ Lt(i), 0 ≤ i ≤ M)
```

Megjegyzés. A 7. (és 39.) sorban levő művelet valójában egy kis önálló algoritmust takar, amely M darab nemnegatív $X(0,1),\ldots,X(m,1)$ szám közül kiválasztja a legkisebb pozitív számot. Ennek megírása igen egyszerű feladat, ezért részletezésére nem térünk ki.

A fenti algoritmust egyszerű kiegészítéssel alkalmazni lehet a rendszer $[0,t]$ időintervallumra vonatkozó átlagos hatékonysági mutatóinak kiszámítására (pl. az egyes egységeken az átlagos sorhosszúság, átlagosan kiszolgált igények száma, átlagos várakozási idő, stb.). Ehhez mindössze annyit kell tenni, hogy a 34. sor után kumuláljuk az egyes $[T(k), T(k+1)]$ szakaszokra, illetve az utolsó, t-t meg nem haladó $T(k)$ és t közé eső időintervallumra a rendszerre vonatkozó integrális mennyiségeket és az algoritmus befejeződése után átlagoljuk t-vel.

26.5. A tömegkiszolgálási rendszerek vizsgálata szimulációval

A tömegkiszolgálási rendszerek jellemzőinek vizsgálatánál gyakran még egyszerűnek látszó modellek esetén sem tudunk egzakt formulákat adni. Ilyen esetekben hasznos és hatékony módszer a szimuláció. Ez a megközelítés azon alapszik, hogy a rendszer működé-

sére nagyszámú kísérletet végzünk, mintegy a „rendszer működését szimuláljuk” és a nyert eredmények alapján vonunk le következtetéseket. A módszer alkalmazása egyébként nem szorítkozik csak a tömegkiszolgálási rendszerek vizsgálatára, sikerrel alkalmazható determinisztikus problémák megoldására is, pl. bonyolult integrálok kiszámítására, differenciál- és integrálegyenletek megoldására, stb., ennek mikéntjére itt nem térünk ki.

A szimulációs módszert a már korábban vizsgált $G|G|1|\infty$ rendszeren keresztül mutatjuk be a korábbi feltételek teljesülése mellett, azonban a módszer hasonló módon alkalmazható más TKR-ek esetén is. Jelölje X_0, (X_n, Y_n), $n = 1, 2, \ldots$ a rendszer beérkezési folyamatát és vizsgáljuk a rendszer egyik jellemzőjét, mondjuk az $\overline{L}_T$ átlagos sorhosszúságot a $[0, T]$ intervallumon. A vizsgálat a következő lépésekből áll:

1. Pszeudovéletlenszám-generátor felhasználásával létrehozzuk a rendszert jellemző eloszlással a véletlen x_0, (x_n, y_n), $1 \le n \le n_0$ számsorozatot (véletlen beérkezési folyamatot, ahol n_0 jelenti azoknak az igényeknek a számát, melyek a $[0, T]$ intervallumon érkeznek a rendszerbe, azaz

$$n_0 = \max\{k : t_k = x_0 + \cdots + x_k < T\} .$$

2. Meghatározzuk az s_k, $1 \le k \le n_0$ távozási időpontokat.

3. Kiszámítjuk az

$$\overline{L}_T = \frac{1}{T} \int_0^T L(t)dt$$

átlagos sorhosszúságot. Ebben az esetben felhasználhatjuk az $L(t) = N(t) - M(t)$, $t \ge 0$ összefüggést, ahonnan

$$\overline{L}_T = \frac{1}{T} \int_0^T L(t)dt = \frac{1}{T} \int_0^T [N(t) - M(t)]dt = \frac{1}{T} \sum_{k=1}^{n_0} [\min(s_k, T) - t_k] .$$

Megjegyezzük, hogy az összegzésben a $\min(s_k, T) - t_k$ éppen azt az időtartamot jelenti a $[0, T]$ intervallumon, amennyit a k-adik igény eltölt a rendszerben.

Megjegyzés. Nagy T (nagy n_0) esetén helytakarékosabb eljárást is követhetünk: egymás utáni foglaltsági periódusokra összegezzük az integrális mennyiségeket egészen a T időpontig és a végén normálunk T-vel. Ekkor elegendő csak az új foglaltsági periódusra vonatkozó t_k és s_k időpontokat ismerni, a megelőző időpontokra a további számításoknál már nincs szükség.

Megjegyzés. Az ún. *regeneratív szimulációs módszerrel* is vizsgálhatjuk az átlagos sorhosszúságot, amely bár kicsit bonyolultabb ebben az esetben, de további következtetésekre ad módot.

Ez a megközelítés azon az észrevételen alapszik, hogy a rendszer a t_{k_i}, $i = 1, 2, \ldots$ időpontokban, az üres periódus után az éppen t_{k_i} időpontban beérkező igényt kezdi kiszolgálni és a rendszer további állapotváltozásai teljes mértékben függetlenek a $0 \le t \le t_{k_i}$ időintervallumon felvett állapotától, vagyis a rendszer a t_{k_i}, $i = 1, 2, \ldots$ időpontokban mintegy *regenerálódik*. Ez azt jelenti, hogy az egymást követő foglaltsági és üres periódusokból álló szakaszok függetlenek, és függetlenek lesznek azok a folyamatszakaszok is,

amelyek leírják a rendszer állapotát. Innen következik, hogy a rendszernek az egyes regeneratív szakaszokon számított jellemzői független azonos eloszlású valószínűségi változókat képeznek, melyekre alkalmazható a nagy számok törvénye, a centrális határeloszlás tétel és ennek alapján pl. konfidencia intervallum is szerkeszthető a rendszer mutatóira. E tulajdonság felhasználásával bizonyítható pl. az is, hogy a rendszerben tartózkodó igények számának létezik-e határeloszlása ($T \to \infty$), vagy létezik-e (1 valószínűséggel) határértéke az $\overline{L}_T$ és $\overline{W}_T$ átlagos értékeknek. Számos TKR rendelkezik a regeneratív tulajdonsággal és hatékonyan vizsgálhatók ezzel a módszerrel. Az $M|G|1$ rendszer esetén ezen a módon az ergodikus eloszlást meghatározhatjuk a kiszolgálási idő eloszlásának ismerete nélkül, csak a közvetlenül ismert adatokra (beérkezési ráta, egy igény kiszolgálási idejének várható értéke és az egy kiszolgálás alatt belépő igények száma) támaszkodva.

Visszatérve konkrét rendszerünk vizsgálatához, meghatározzuk a korábban megadott eljárással a $[0, T]$ intervallumot lefedő minimális n_* számú Δ_j, $1 \le j \le n_*$ foglaltsági periódust a hozzátartozó k_j, $1 \le j \le n_*$ kezdő indexekkel együtt, ahol

$$n_* = \min\{k_j - 1 : \ k_j > n_0, \quad j \ge 1\} .$$

Legyen

$$I_j = \int\limits_{\Delta_j} L(t)dt = \sum_{k=k_j}^{k_{j+1}-1} I(s_k < T)[s_k - t_k], \quad j \ge 1$$

és

$$J_{n_*} = \int\limits_{\Delta_{k_{n*}}} L(t)dt = \sum_{k=k_{n*}}^{k_{n*+1}-1} I(s_k \ge T)[\min(s_k, T) - t_k] .$$

Ekkor a rendszerben tartózkodó igények átlagos száma

$$\overline{L}_T = \frac{1}{T}\int\limits_0^T L(t)dt = \frac{1}{T}\sum_{j=1}^{n_*-1} I_{k_j} + \frac{1}{T}J_{k_{n*}} = \frac{n_*}{T}\frac{1}{n_*}\sum_{j=1}^{n_*} I_{k_j} - \frac{1}{T}(I_{k_{n*}} - J_{k_{n*}}) .$$

Az egyenletben szereplő véletlen mennyiségekről a következőket tudjuk mondani:

- A valószínűségszámításból ismert felújításelmélet alapján a regenerációs periódusok n_* számára fennáll $T/n_* \to \eta$, $T \to \infty$ 1 valószínűséggel, ahol η jelenti egy regenerációs hossz várható értékét, amely minden regeneratív periódusra ugyanaz.
- Minthogy $I_{k_{n*}}$ eloszlása megegyezik I_{k_1} eloszlásával, ezért $J_{k_{n*}} \le I_{k_{n*}}$ miatt nyilvánvalóan
$$\frac{1}{T}(I_{k_{n*}} - J_{k_{n*}}) \le \frac{1}{T}I_{k_{n*}} \to 0, \qquad T \to \infty$$
1 valószínűséggel.
- Az I_j, $1 \le j$ valószínűségi változók függetlenek és azonos eloszlásúak, $n_*/T \to 1/\eta$, $T \to \infty$ 1 valószínűséggel, ezért a véletlen tagszámú
$$\frac{1}{n_*}\sum_{j=1}^{n_*} I_{k_j}$$
összegre igaz a centrális határeloszlás tétel: legyen $\mu = E(I_{k_1})$, $\sigma = D(I_{k_j})$, akkor

tetszőleges valós x szám mellett

$$P\left(\frac{1}{\sqrt{n_*}\sigma}\sum_{j=1}^{n_*}(I_{k_j}-\mu)<x\right)\to\Phi(x),\quad T\to\infty\,,$$

ahol $\Phi(x)$ jelöli a standard normális eloszlásfüggvényt. Megjegyezzük, hogy mindhárom esetben külön vizsgálható a konvergencia sebessége. Ezek az összefüggések teszik lehetővé, hogy a szimulációs eredmény alapján konfidencia intervallumot is szerkesszünk az $\overline{L}$ átlagos sorhosszúságra.

26.5.1. Szimulációs eszközök

A távközlő rendszerek elemzésének későbbiekben vázolt analitikus eszközei mellett egy sokkal szélesebb körben elterjedt elemzési módszer a számítógépes szimuláció. A számítógépes szimuláció során a vizsgálandó rendszer vizsgált jellemzője szempontjából lényeges tulajdonságait kell egy szimulációs program számára leírni, és a szimulációs program ez alapján „lejátssza” a vizsgált rendszer működésének egy időszakát.

Egy távközlő rendszer viselkedését a rendszer véletlen elemeit is jellemző sztochasztikus folyamat segítségével vizsgáljuk. A szimuláció során a véletlen eseményekre, azok ismert sztochasztikus jellemzői (pl. független vagy együttes eloszlás) alapján sorsolunk. A sorsolást a programnyelvek túlnyomó többségében megvalósított *véletlen szám generátorral* végezzük. A tipikus véletlen szám generátorok valójában ún. pszeudovéletlen (álvéletlen) számokat adnak, azaz egy előre meghatározott véges hosszúságú sorozat elemeit adják vissza ciklikusan. Azt, hogy a program indításakor melyik elemtől kezdve kapjuk a sorozat elemeit, egy kezdeti érték (seed) határozza meg. A véletlen szám generátorok általában 0 és 1 között folytonos egyenletes eloszlású véletlen számot szolgáltatnak, amelyből egy transzformációs függvény segítségével képzünk tetszőleges eloszlású számokat. A véletlen szám generátor hossza, illetve a sorozat elemeinek tulajdonságai befolyásolhatják a szimuláció jóságát.

A számítógépes szimuláció a véletlen folyamat egy megvalósulását utánozza. A programot a véletlen szám generátor ugyanazon kezdőértékével újra indítva ugyanazt a megvalósulást fogja adni. Ez nagy segítséget nyújt a szimulációs programok hibáinak megtalálásához, mivel a mindig másfajta hibákat produkáló program kijavítása nagyon nehéz feladat. A programot a véletlen szám generátor más kezdőértékével indítva egy új megvalósulást fog létrehozni.

A vizsgált rendszernek a szimulációs programban kialakított modellje egy matematikai értelemben jól definiált sztochasztikus modell, így elvileg ennek a sztochasztikus modellnek analízis eszközökkel történő elemzésével is meghatározhatók a keresett teljesítmény jellemzők. Azonban az esetek többségében a szimulációs modellek olyan bonyolultak, hogy analitikus vizsgálatuk nem lehetséges. Javasolt azonban a szimulációs és a pontos, vagy ha nem lehetséges, akkor a közelítő analitikus vizsgálatok együttes végrehajtása, mivel bonyolult viselkedések esetén apró hibák is téves következtetésekre vezethetnek.

Szimulációs programot készíthetünk általános célú programozási nyelven (pl. C) vagy valamilyen szimulációs keretrendszer felhasználásával. Az első esetben a szimulációs modell mellett a szimuláció futtatását végző programrészt (szimulációs motor), valamint a vizsgált jellemzők kiszámításához szükséges adatok gyűjtését és feldolgozását végző program-

részt is el kell készíteni.

A második esetben a szimulációs keretrendszer már tartalmazza a szimulációs motort és általában támogatást nyújt a modell leírásához és a szükséges adatok megadásához. Azokat a számítógépes nyelvi vagy grafikus elemeket és a rájuk vonatkozó szabályrendszert, amelyekkel a vizsgálandó modellt leírjuk, modell leíró nyelvnek nevezzük. Alacsony szintű modell leírásnak nevezzük, ha a modell viselkedését egy általános célú programnyelv segítségével adjuk meg. Magas szintű modell leírás esetén előre definiált, esetenként grafikusan megjelenített modellrészekből építhetünk szimulációs modellt.

A magas szintű modell leírás nagyon megkönnyíti a bonyolult szimulációs modellek leírását, azonban azt a veszélyt hordozza magában, hogy a modell készítője előtt rejtve marad a modell alacsony síkjainak viselkedése, és így alapvetően egy ismeretlen rendszer elemzése történik.

A távközlő rendszerek véletlen forgalmi folyamatainak szimulációjára használható eszközök közül mutatunk be néhány példát a teljesség igénye nélkül.

Általános célú integrált szimulációs környezet

- MATLAB program SIMULINK csomagja

  (http://www.mathworks.com/products/simulink): a MATLAB az egyik legelterjedtebb programcsomag technikai számítások elvégzésére. A SIMULINK szimulációs csomag a MATLAB beépített függvényeire építve egy grafikus környezetet biztosít szimulációs modellek tervezéséhez és futtatásához.

Távközlő rendszerek szimulációja

- OMNeT++ (http://www.omnetpp.org):

  Az OMNet++ egy objektum orientált moduláris felépítésű diszkrét esemény szimulátor. Alkalmazható kommunikációs protokollok, számítógép hálózatok, forgalmi modellek, több processzoros és elosztott rendszerek szimulációjára. Támogat animációt és interaktív futtatást. Ingyenesen letölthető.

- ns2 (http://www.isi.edu/nsnam/ns/):

  Az ns2 programot kommunikációs hálózatok kutatási célú szimulációjára fejlesztették ki. Beépített modulok támogatják a TCP ptotokoll, az útvonalválasztási eljárások, a vezetékes és vezeték nélküli hálózatok vizsgálatát. Szintén ingyenesen letölthető.

- OPNET (http://www.opnet.com/):

  Az OPNET egy ipari célokra kifejlesztett szimulációs csomag, amely a magas szintű hálózat modellezés mellett támogatja a hálózati technológiák megértését, tervezését és üzemeltetését.

- Traffic (http://www.erlang-software.com/):

  A Traffic szimulációs csomagot kifejezetten a távközlő hálózatok forgalmi viszonyainak elemzésére fejlesztették ki. Lehetőséget nyújt olyan bonyolult rendszerek elemzésére, amelyeket a hagyományos Erlang-formulák segítségével nem lehet vizsgálni.

26.6. Távközlési algoritmusok

A távközlésben tipikusan sok felhasználó, véletlenszerűen, többe-kevésbé egymástól függetlenül szeretne üzeneteket továbbítani egy közös erőforráson, a távközlő hálózaton. Ennek a feladatnak különböző további követelményeket kielégítő megoldására a távközlő algoritmusoknak egy nagyon szerteágazó széles körét dolgozták ki. Az újabb és újabb megoldások kialakításában szerepet játszottak a folyamatosan változó távközlési igények és távközlési technologiák. Az előbbit a következő alfejezet tárgyalja. A távközlési technológiák fejlődését a nagy sávszélességű digitális (például optikai) adatátvitel elterjedése, és a csomópontokban sávszélesség növekedés ellenére is növekvő komplexitású kiszolgálási funkciók jellemzik.

A fejlődés során néhány megoldási módszer elhalt (például distributed queue dual bus - DQDB), míg mások folyamatosan továbbfejlődnek (pl. internet protocol - IP), amit sok esetben nem a műszaki megoldás minősége, hanem a megoldás mögött felsorakozó távközlési cégek ereje befolyásolt.

Az alábbiakban egy rövid bevezető után a teljesség igénye nélkül ismertetünk néhány fontosabb kiszolgálási elvet, és azok néhány tipikus megvalósítását. A bemutatásra kerülő megoldásokat minden esetben csak a sorbanállási tulajdonságok szempontjából vizsgáljuk. A távközlési protokollok hierarchiájának további elemeit és megoldásait itt nem tárgyaljuk.

26.7. Távközlési igények változása

A kezdetben szinte kizárólag emberek közti beszédátviteli igények a vezetékes és vezeték nélküli számítógépes kommunikáció fejlődésével eltolódtak az esetenként nagy adatmennyiségek gyors átvitelét igénylő számítógépek közti adatcsere irányába, ami nagyon megváltoztatta az igények tulajdonságait és az elvárások jellegét.

A beszédátvitel főbb jellemzői a következők:

1. jól meghatározott kapcsolati időszak (beszélgetés eleje és vége között),
2. a kapcsolat idején jól meghatározott, a rendelkezésre álló összes sávszélességhez képest aránylag alacsony sávszélesség igény (beszéd ideje alatt az alkalmazott beszéd tömörítéstől függően pl. < 12 kbs, beszédszünet alatt 0),
3. alacsony átviteli idő elvárás (pl. < 100 ms),
4. néhány beszédcsomag elvesztése csak minőség romlást okoz, nem igényli az egész kommunikáció megismétlését.

Ezzel szemben a számítógépes kommunikációban

1. állandóan hálózathoz kötött számítógépek esetén nem azonosítható egyértelmű kapcsolati időszak,
2. az igényelt sávszélesség néhány csomag érdemi késleltetés korlát nélküli átvitelétől a hálózat szűk keresztmetszeteinek kapacitásával összemérhető sávszélességig terjedhet,
3. az adatátvitel lehet a késleltetés idejére érzékeny, a késleltetés ingadozásra érzékeny, illetve a késleltetésre érzéketlen,
4. megkülönböztetünk adatvesztésre érzékeny és kevésbé érzéketlen igényeket.

Az első szempont alapján a hagyományos ún. vonalkapcsolt megoldások helyett tért nyertek a csomagkapcsolt megoldások. A tisztán vonalkapcsolt esetben a távközlési erőforrások igényelt részét lefoglajuk az adott kapcsolat idejére. Míg a tisztán csomagkapcsolt esetben az átviendő adatokat apró darabokban, ún. csomagokban továbbítjuk előzetes erőforrás foglalás nélkül.

Az igények relatív nagysága az erőforrásokhoz képest abból a szempontból meghatározó, hogy mekkora relatív ingadozású az igényfolyamat. Amennyiben a felhasználók, az erőforrásoknak csak egy kis részét igénylik, akkor sok felhasználó együttes viselkedése határozza meg az igényfolyamat relatív ingadozását, ellenben ha kevés igényforrás is túlterhelheti a rendszert, akkor az igényfolyamat relatív ingadozása lényegesen nagyobb.

A késleltetére érzékeny igények az ún. valós idejű, vagy párbeszédes igények, mint a telefon, vagy a video konferencia. A késleltetés ingadozásra érzékeny szolgáltatás az audio vagy video folyam átvitele, mert a vevő oldali egyidejű lejátszáskor a lejátszás késletetése nem érzékelhető, viszont a csomagok egymáshoz képesti késése azt eredményezheti, hogy egy csomag még nem érkezik meg, amikor a lejárszására kerülne a sor.

Az adatvesztést elvileg minden átvitel során kerülni kell, de amíg az audio vagy video átvitelben a csomagvesztés csak átmeneti minőségromlást okoz, addig pl. egy fájl átvitel során minimum az elveszett csomag sikeres átvitelig tartó ismétlését igényli.

A felsorolt jellemzőket (késleltetés, késleltetés ingadozás, csomagvesztés) gyűjtőnéven minőségi jellemzőknek nevezik (quality of service, QoS).

26.8. Igények változásának következményei

Az erőforrásokhoz mérten kis sávszélességű vonalkapcsolt igényeket kiszolgáló hálózatokban egy lehetséges kiszolgálási megoldás az erőforrások felosztása, és az aktív kapcsolatokhoz rendelése a kapcsolat idejére. Ez az eljárás a kapcsolat felépítésének pillanatában megvizsgálja, hogy rendelkezésre áll-e az igényelt erőforrás, és ha igen, akkor lefoglalja, és az adott kapcsolathoz rendeli, ha nem, akkor elutasítja az igényt.

Számítógép hálózatokban is alkalmaznak ehhez hasonló elven működő erőforrás megosztási eljárást. Abban az esetben, ha az átviendő adatok adott csomópontpárok között, hosszabb ideig, adott statisztikus jellemzőkkel rendelkeznek, akkor egy ún. hívás engedélyezési eljárás (Call admission control CAC) keretében eldöntjük, hogy az új igény kiszolgálására rendelkezésre áll-e a szükséges erőforrás, és ez alapján a hívást vagy elfogadjuk, vagy elutasítjuk.

Sok, hálózathoz kötött, időnként rövid időre aktívvá váló számítógép esetében azonban nem alkalmazható hatékonyan az erőforrás foglalás, mivel ebben az esetben a kapcsolat felépítéshez szükséges többlet tevékenység már összemérhető az alaptevékenységgel, az adatátvitellel. Ilyen forgalom források esetén kapcsolat felépítési fázis nélkül küldhetnek csomagokat a számítógépek, és a forgalom hálózatba lépési pontjában csomagfolyamot korlátozó eljárásokkal befolyásolható az erőforrások megfelelő megosztása. Ilyen forgalom korlátozó eljárások a leaky bucket és a GCRA.

A forgalmi igények különböző késleltetés és adatvesztés érzékenysége motiválta a forgalmi osztályok bevezetését, ugyanis más típusú kiszolgálás hatékony késleltetésre érzékeny és csomagvesztésre érzékeny forgalmak esetén. Ennek megfelelően adott forgalmi viszonyok esetén késleltetés érzékeny forgalom átvitelére kisebb puffer méret, míg csomagvesztés érzékeny forgalom átvitelére nagyobb pufferméret alkalmas.

Abban az esetben viszont, amikor egy közös erőforráson kell késleltetés- és csomagvesztésérzékeny forgalmi összetevőket tartalmazó forgalmat átvinni, akkor megfelelő puffer méretezéssel nem biztosítható minden minőségi igény kielégítése. Ekkor a legegyszerűbb eljárás az átviteli kapacitás növelése addig, amíg adott puffer méret mellett mind a késleltetés- mind a csomagvesztésérzékeny forgalom minőségi jellemzői elérik a megkövetelt szintet.

A kiszolgálási kapacitás növelése azonban a kapacitás felhasználás szempontjából nem hatékony eljárás. Az így kialakított rendszerek aránylag egyszerűek, de alacsony hatásfokúak.

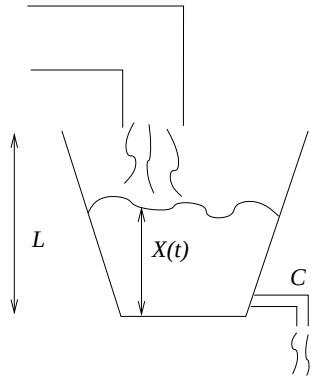
Adott minőségi paraméterek kisebb kapacitás mellett is biztosíthatók, ha a hálózat csúcspontjaiban forgalom osztály specifikus megkülönböztetést és kiszolgálást végzünk. Ez az eljárás bonyulultabb funkciók megvalósítását igényli a csomópontokban, viszont javítja a rendszer kihasználtságát.

A fent említett forgalom szabályozó és a szolgáltatás megkülönböztetést végző eljárásokkal foglalkozunk a következő fejezetben.

26.9. Forgalom szabályozó eljárások

A számítógépes kommunikáció esetén a forgalom források erősen ingadozó viselkedése és nagy adatátviteli kapacitása miatt a halózattal szemben támasztott átviteli igények erősen ingadozhatnak. Az ilyen jelentős ingadozások megnehezítik a csomagok adott minőségi elvárásoknak megfelelő átvitelét. Az ingadozások csökkentésének egyik lehetséges módja a forgalom források korlátozása, hogy csak az előre megállapított adatsebességgel küldhessenek csomagokat a hálózatba. Ezt a korlátozást valósítják meg a hálózat bemenő pontjain alkalmazott forgalom szabályozó eljárások.

Az alábbiakban az eljárások tárgyalása során két modellezési szintet különböztetünk meg. Az első az elvi modell szintje, ahol csak az átviendő adatmennyiséget tekintjük, és nem vesszük figyelembe, hogy csomagokban történik az adatok továbbítása. A második szinten az elvi modell egy valós, csomagtovábbítást végző hálózatban is alkalmazható változatát ismertetjük.



26.7. ábra. Lyukas vödör eljárás.

26.9.1. Lyukas vödör eljárás

Az elvi modell szintjén az átviendő adatmennyiséget folytonosnak tekintjük, fizikai analógia alapján például folyadéknak. Ebben a modellben a forgalom szabályozás feladata a hálózatba folyó folyadék ingadozását adott szabályok szerint csökkenteni (lásd 26.7. ábra).

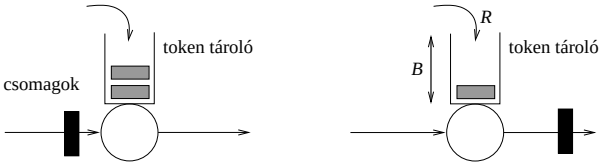
Az eljárás lényegét jól szemléleteti az eljárás neve. A bejövő folyadékot egy L űrtartalmú vödörbe vezetjük, amelyikből maximálisan C sebességgel tud kifolyni a folyadék. Ha a folyadék beáramlási sebessége (R_{IN}) egy intervallumon meghaladja I értékét, akkor a vödörben a folyadékszint (X) emelkedik. Amikor a folyadékszint eléri L-t, akkor a többlet folyadék elvész. A folyadék kiáramlási sebessége mindig C, amíg a vödör nem üres, és amikor a vödör üres, akkor a kiáramlás (R_{OUT}) és a beáramlás sebessége megegyezik.

$$\begin{aligned}
&\frac{dX(t)}{dt} = R_{IN} - C, R_{OUT} = C, && \text{ha } 0 < X(t) < L\,,\\
&\frac{dX(t)}{dt} = 0, R_{OUT} = C, && \text{ha } X(t) = L, R_{IN} > C\,,\\
&\frac{dX(t)}{dt} = 0, R_{OUT} = R_{IN}, && \text{ha } X(t) = 0, R_{IN} < C\,.
\end{aligned} \tag{26.1}$$

26.9.2. Lyukas vödör eljárás csomagtovábbítás esetén

A lyukas vödör eljárás folytonos folyadék beáramlást feltételez. Csomagtovábbító távközlési rendszerek elemzése során azonban az adatok érkezését sok esetben pillanatszerűnek tekintik, és a csomag első vagy utolsó adategységének érkezéséhez rendelik. Ezt a modellt például az a tulajdonság indokolja, hogy a csomagok tetszőleges részének elvesztése esetén a csomag egészét eldobják a távközlő rendszerek, így csomagokra vonatkozóan kell meghatározni a továbbítás módját. Az ismertetett lyukas vödör modell ezt a tulajdonságot nem tükrözi. A fix méretű csomagok átvitelének leírására ezért a modellnek egy módosított változatát használják.

A csomagforgalom lyukas vödör modelljében a csomagok érkezési pillanataiban (t) vizsgáljuk a vödörben a folyadék feltételezett szintjét ($X_a = X - (t - s)$) az előző sikeres



26.8. ábra. Tokentárolós eljárás.

csomagtovábbítás folyadékszintje (X), az azóta eltelt idő ($t - s$), és a megengedett adattovábbítási sebesség (C) alapján. Amennyiben a beérkező csomag még nem tölti tele a vödröt, akkor az a csomag sikeresen továbbítható, viszont ha az érkezés hatására kicsordulna a vödörből a folyadék, akkor eldobjuk a csomagot. Az egyszerűség kedvéért az eljárásban az adatsebesség (C) és a csomagméret (d) helyett a csomagok engedélyezett érkezési időközével (I) számolunk, ahol $I = d/C$.

LYUKAS-VÖDÖR(X, t, s, I, L)

```
1  X_a ← X − (t − s)
2  if X_a < 0
3     then X ← I
4          s ← t
5          return IGAZ
6     else  if X_a > L
7              then return HAMIS
8              else  X ← X_a + I
9                    s ← t
10                   return IGAZ
```

26.9.3. Tokentárolós csomagtovábbítási eljárás (token bucket)

Az előző eljárásokban a vödör mérete jellemezte a megengedettnél gyorsabban átvihető adatmennyiséget. Ha a vödör folyadékszintje helyett a vödörbe még betölthető folyadék ($L - X$) egész csomagméret kerekített értékével jellemezzük a rendszert, akkor egy más szemléletű modellel valósíthatunk meg a fentiekhez hasonló forgalom szabályozó eljárást.

Ebben a modellben a csomagok csak tokenekkel (vagy zsetonokkal, ami egy-egy csomag átvitelét engedélyezi) együtt továbbíthatók. A 26.8. ábrán a feketével jelzett csomag továbbításakor egy szürke token is távozik a rendszerből. A tokenek állandó sebességgel keletkeznek. I időnként keletkezik egy token, amit általában az $R = 1/I$ token keletkezési rátával jellemeznek. A tokentároló méretét a tokenek maximális számával jellemzik $B = L/d$. Ha a tokentároló megtelik, a keletkező újabb tokenek elvesznek. Ha egy csomag érkezésekor a tokentároló üres, akkor a csomag elvész.

26.9.4. GCRA eljárás

Az ATM hálózatok tipikus forgalom szabályozó eljárásaként elterjedt GCRA algoritmus a lyukas vödör eljárás csomagszintű érkezéseket modellező változatának egy hatékonyan implementálható módosítása, amelyben az elméleti érkezési idő (TAT - **T**heoretical **A**rrival **T**ime) helyettesíti a folyadék szintet.

GCRA(TAT, t, I, L)

```
1  if TAT < t
2     then TAT ← t + I
3          return(IGAZ)
4     else if TAT > t + L
5             then return(HAMIS)
6             else X ← X_a + I
7                  return(IGAZ)
```

A felsorolt forgalomszabályozó eljárások mindegyikét közvetve, vagy közvetlenül két paraméter jellemzi a gyorsan továbbítható adatmennyiség (vödör méret) L és a megengedett átlagos adattovábbítási sebesség C. ATM terminológiával az mondják, hogy az ilyen forgalomszabályzón átvezetett forgalom átlagos sebessége nem nagyobb, mint C és maximális börszt mérete L.

Ezeknek a forgalomszabályozó eljárásoknak közös jellemzője, hogy késleltetést nem okoznak, viszont a csomagok egy részét eldobják. E rendszerek tipikus analízis kérdése, hogy adott tulajdonságú forgalomforás esetén mekkora a csomageldobási valószínűség.

26.10. Forgalom megkülönböztetést végző kiszolgálási eljárások

A távközlő hálózatok csomópontjaiban különböző felhasználóktól, különböző irányokból érkező és különböző minőségi elvárásokkal rendelkező csomagokat kell továbbítani adott kimenő irányokba. Nem szinkronizált érkezési folyamatok esetén ez a feladat még akkor is igényelhet konfliktus feloldást, ha a kimenő kapacitás meghaladja a beérkező csomagfolyamok maximális intenzitásainak összegét, ugyanis ha két csomag érkezése közt az idő kisebb, mint az előbb érkezett csomag továbbításának ideje és mindkét csomag továbbítására azonos erőforrást használunk, akkor a második csomag továbbításával várni kell az első továbbításának befejezéséig.

A konfliktusok feloldására több módszert is alkalmaznak távközlő hálózatokban:

1. erőforrás megosztás (fix vagy dinamikus),
2. közös erőforrás konfliktus feloldó algoritmussal,
3. közös erőforrás pufferrel.

Az erőforrás megosztás célja a versenyhelyzet megszüntetése. Az aktív forgalomforrások mindegyikének biztosítunk egy kizárólag számára elérhető erőforrás részt. Attól függően, hogy ezt az erőforrás felosztást milyen gyakran módosítjuk, beszélhetünk statikus

vagy dinamikus erőforrás megosztásról. Dinamikus erőforrás megosztás esetén gyakori, hogy az erőforrás egy részét fenttartják új igények bejelentésére, ahol az igénybejelentő csomagok versenyeznek egymással, és egy központi vezérlő a beérkezett igények alapján végzi az erőforrás megosztást.

Közös erőforrás véletlenszerű használata esetén az igényforrások a véletlen időpillanatokban keletkező csomagjaikat a keletkezés pillanatában próbálják továbbítani (feltételezve, hogy a fennálló elvárásoknak megfelel a csomagkeletkezési folyamat). Ebben az esetben a hálózat véges kapacitásai miatt konfliktusok keletkezhetnek (pl. két forrás egyszerre szeretne csomagot küldeni). A konfliktusfeloldás két lehetséges módszere a konfliktusba került csomagok eldobása, vagy a hálózat csomópontjaiban lévő tárolókban várakoztatása. A csomagok eldobása esetén a forgalomforrás értesül a csomagtovábbítás sikertelenségéről, és egy egyszerű algortimus alapján idővel megpróbálja ismételten elküldeni csomagját. Ennek az algoritmusnak biztosítania kell, hogy a konfliktusok idővel feloldódjanak, és idővel minden csomag továbbításra kerüljön.

A felsorolt eljárások közül sok esetben itt nem tárgyalt mérnöki szempontok alapján kell választani, de amennyiben forgalmi szempontok dominálnak, akkor a következő tényezőket mérlegelhetjük:

- csomagtovábbítási folyamat függése/függetlensége más források viselkedésétől,
- műszaki és algoritmikus megvalósítás bonyolultsága,
- késleltetés, és ha létezik, akkor a csomagvesztés valószínűsége.

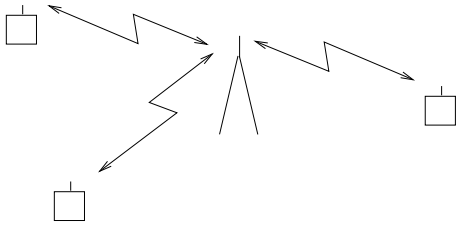
A felsorolt tényezők szerint az konfliktusfeloldási eljárások jellemzői a következők:

1. erőforrás megosztás (fix vagy dinamikus):
   előny: nem zavarják egymás kiszolgálási jellemzőit, általában egyszerű megvalósítani (idő-, frekvencia-, kódosztás),
   hátrány: nem állandó intenzitású forgalomforrások esetén nem hatékony, lassan és nem tetszőlegesen változtatható a megosztási arány, külön protokol szükséges a megosztás karbantartására (igény bejelentés, döntés, megosztás, igény megszünése),
   példa: GSM telefon és bázis állomás közti kapcsolat (idő- és frekvenciaosztás);

2. közös erőforrás konfliktus feloldó algoritmussal;
   előny: igény szerinti erőforrás felhasználás, a hálózat viselkedése egyszerű (ütközés → csomag eldobás),
   hátrány: az elosztott konfliktus feloldás nem hatékony, az igényfolyamatok befolyásolják egymás kiszolgálását,
   példa: ALOHA, CSMA (Ethernet), IEEE 802.11 (wireless LAN)

3. közös erőforrás pufferrel:
   előny: igény szerinti erőforrás felhasználás, hatékony központi konfliktus feloldás
   hátrány: központi puffer és kiszolgálás vezérlést igényel, igényfolyamatok befolyásolják egymás kiszolgálását.
   példa: FIFO, Prioritás, WFQ.

Az erőforrás megosztás algoritmusai általában a kapcsolat érkezési és végződési folyamat követését végzik, ezekre nem térünk ki. A következő fejezetben tárgyaljuk a konfliktus feloldó algoritmusokat, és az azt követő fejezetet szenteljük a pufferes konfliktus feloldás vagy más néven sorbanállási és kiszolgálási rendszerek vizsgálatának.



26.9. ábra. Rádió terminál rendszer.

26.11. Véletlen erőforrás hozzáférés konfliktus feloldó algoritmusai

Véletlen erőforrás hozzáférés esetén több, egymás viselkedését közvetlenül nem ismerő felhasználó próbál adatcsomagokat továbbítani egy közösen használt átviteli közegen. A felhasználók adattovábbítását olyan algoritmus szerint végzik, amelyik a többi felhasználó viselkedéséről valamilyen módon rendelkezére álló információk alapján biztosítja, hogy

- egy rendszerfüggő terhelési szint alatt ne akadjon el az adattovábbítás,
- minden átküldésre szánt csomag idővel átjusson a rendszeren,
- amennyiben nem törekszünk a felhasználók megkülönböztetésére, akkor a rendszer igazságos minden felhasználóhoz, azaz egyforma esélyeket biztosít számukra adatcsomagjaik átvitelére.

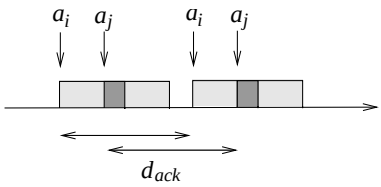
Az alábbiakban ismertetett eljárások egy algoritmus család különböző változatai, amelyek abban különböznek egymástól, hogy

- a felhasználók milyen módon értesülnek a többi felhasználó viselkedéséről,
- milyen tervezési szempontokat érvényesítenek a forgalom ingadozása esetén.

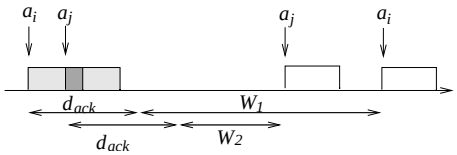
26.11.1. ALOHA eljárás

Az algoritmus család legegyszerűbb tagját rádió terminálok és egy központ közti kommunikáció céljára fejlesztették ki (lásd a 26.9. ábrát). A rendszer két frekvenciasávot használ. Az egyiken a terminálok küldhetnek adatot a központ felé, a másikon a központ az összes terminálnak. (Az ALOHA eljárást Hawai-on fejlesztették ki, maga a név egy helyi üdvözlésből ered.)

Ha az első frekvenciasávban egyszerre több terminál szeretne adatot továbbítani, akkor az azonos frekvenciájú rádiójelek összegződése miatt a központ nem tudja az egyik terminál adatát sem fogni. Ekkor azt mondjuk, hogy a terminálok csomagjai ütköztek. Mivel a második frekvencia sávban csak a központ sugározhat jeleket, így ott nem alakulhat ki ütközés. A sikeresen vett jeleket a központ egy következő üzenetében nyugtázza. A terminálok az ütközés bekövetkezéséről abból értesülnek, hogy az általuk elküldött csomagot a központ egy adott határidőn belül nem nyugtázza. Az ALOHA eljárás ebben a rendszerben biztosítja a felsorolt szempontoknak megfelelő adat kommunikációt.



26.10. ábra. Csomagismétlés várakozás nélkül.



26.11. ábra. Csomagismétlés véletlen várakozással.

Az eljárás működése nagyon egyszerű. Amint egy terminálnak keletkezik egy átviendő adatcsomagja, azonnal elküldi. Ha a rendszerre jellemző határidőig nem érkezik nyugta a csomag sikeres átviteléről, akkor a terminál azt feltételezi, hogy a csomag ütközött. Azt mondjuk, hogy ekkor a terminál blokkolt állapotba kerül. Ebben az esetben addig ismétli a csomag továbbítását, amíg a sikeres átviteléről nyugtát nem kap (lásd a 26.10. ábrát).

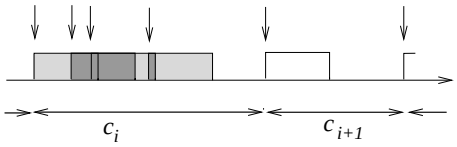
A blokkolt felhasználók a csomagok ismétlését nyilván nem kezdhetik azonnal a határidő letelte után, hiszen ekkor már két ütköző felhasználó is teljesen megakaszhatná a sikeres adat továbbítást, mivel minden csomagismétlési kisérlet alkalmával újra ütköznének a csomagok.

Az ALOHA eljárásban ezért a felhasználók egymástól függetlenül egy véletlen késleltetési időt sorsolnak, és ennek a véletlen időnek a leteltével kisérlik meg a csomagjuk ismételt elküldését (lásd a 26.11. ábrát). Az ábrákon megkülönböztetjük a valódi ütközési időszakot (sötét szürke), és az ütközés miatt feleslegesen használt időszakot (világos szürke).

A rendszer minőségi viselkedése eléggé szemléletes. Amennyiben a felhasználók „nagy” késleltetési időket sorsolnak, akkor kicsi lesz annak az esélye, hogy a kisorsolt késleltetési idők megegyeznek, és a késleltetés elteltével újra ütköznek a csomagok (akár egy, akár több csomag ütközött eredetileg). Ebben az esetben tehát „kevesebb” újraküldési kísérlet szükséges a csomagok átviteléhez, viszont az újraküldési kísérletek közti késleltetési idő „nagy”. A másik szélsőséges viselkedés, amikor a felhasználók „kis” késleltetési időket sorsolnak. Ekkor az ismétlési kísérletek közti idő alacsony, de nagyobb valószínűséggel ütköznek a csomagok, és ezért általában több ismétlési időszakra van szükség. A rendszer hatékony munkapontja e két szélsőség között keresendő.

Az ALOHA rendszerek modellezése és teljesítmény analízise az 1950-es évektől egy erősen kutatott terület, aminek mindig újabb aktualitást adott az egyre újabb ALOHA eljárás változatok bevezetése (lásd a következő alpontokat).

A kiterjedt elemzések különböző felhasználói viselkedést és modellezési megfontolásokat feltételeznek. Általában elmondhatjuk, hogy a valóságos rendszerek tényleges működésének sajátosságait figyelembe vevő analitikus leírás szinte reménytelen. A kidolgozott ana-



26.12. ábra. Üres időszakokkal határolt működési ciklusok.

litikus modellek egyszerűsítő modellezési megfontolásokat tartalmaznak, amelyek azonban sok esetben nagyon pontosan közelítik a valós rendszer működését.

Az eredeti ALOHA rendszerrel kapcsolatban néhány egyszerűen elemezhető teljesítmény jellemzőt tárgyalunk az alábbiakban.

Folytonos idejű ALOHA rendszer

Modellezési megfontolások:

- Az új és az ismételt csomagok együttesen λ paraméterű Poisson-folyamat szerint érkeznek.

  Ez a modellezési megfontolás természetesen (mesterséges esetektől eltekintve) nem teljesül a leírt viselkedés mellett, azonban bizonyos feltételek mellett (pl. a blokkolt felhasználók nem generálnak lényegesen nagyobb forgalmat, mint a nem blokkoltak, a csomagtovábbítási idő lényegesen kisebb, mint az ismétlési idő) jól közelíti a rendszer valódi viselkedését. Ez a modell annyira elterjedt az ALOHA rendszerek elemzésében, hogy „0-adrendű modell”-ként említik.

- A csomagok fix méretűek, az adott átviteli sebesség mellett átviteli idejük T.

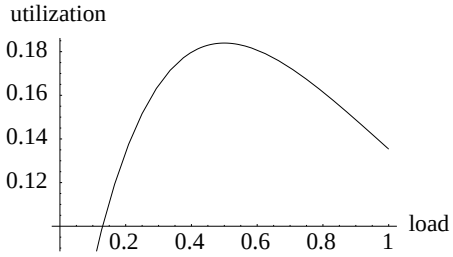
A 0-adrendű modellben azt vizsgáljuk tehát, hogy a λ paraméterű Poisson-folyamat szerint érkező csomagok közül mennyit továbbít sikeresen a rendszer, és hogy ebből mekkora csomag ismétlési valószínűség és késleltetési idő adódik.

A csatorna működését a 26.12. ábrának megfelelően felosztjuk üres időszakok közti működési ciklusokra. Annak valószínűsége, hogy egy működési ciklusban egy csomag átvitele sikeres, megegyezik azzal, hogy a ciklus megkezdése utáni első csomagküldés T-nél később kezdődik, ami $e^{-\lambda T}$.

Annak érdekében, hogy hosszú idő átlagában meghatározzuk a rendszer sikeres, sikertelen és üres időszakainak arányát, meghatározzuk ezen időszakok várható hosszát egy működési ciklus alatt. A üres időszak hossza λ paraméterű exponenciális eloszlású, $1/\lambda$ várható értékkel. A sikeres időszak hossza pontosan T. Egy sikertelen időszak $N-1$ ($N \geq 2$) darab T-nél rövidebb érkezési időközből, és egy lezáró T hosszú időszakból áll. Az $N = 1$ a sikeres csomagtovábbítás esete. A Poisson érkezési folyamat emlékezetmentessége miatt az intervallumok hosszától függetlenül számíthatjuk a sikertelen időszakban ütköző csomagok számát,

$$P(N = n) = (1 - e^{-\lambda T})^{n-1} e^{-\lambda T} .$$

A T-nél rövidebb időszak (U) hosszának eloszlásfüggvénye



26.13. ábra. Kihasználtság ρ a terhelés λT függvényében.

$$F_U(t) = P(U < t) = P(\tau < t | \tau < T) = \begin{cases} \dfrac{1 - e^{\lambda t}}{1 - e^{\lambda T}} & 0 < t < T\,, \\ 1 & T < t\,, \end{cases}$$

amiből $E(U) = \dfrac{1 - e^{-\lambda T} - \lambda T e^{-\lambda T}}{\lambda(1 - e^{-\lambda T})}$. Mindezek alapján egy működési ciklusban

- az üres intervallum várható hossza $E(I) = 1/\lambda$,
- a sikeres csomagtovábbítás várható ideje $E(S) = e^{-\lambda T} T$,
- és a sikertelen csomagtovábbítás várható ideje

$$E(L) = \sum_{n=2}^{\infty} P(N = n)\Big((n-1)E(U) + T\Big) = \frac{1 - e^{-\lambda T} - \lambda T e^{-\lambda T}}{\lambda e^{-\lambda T}}\,.$$

A rendszer kihasználtságát a sikeres csomagtovábbítás időaránya jellemzi

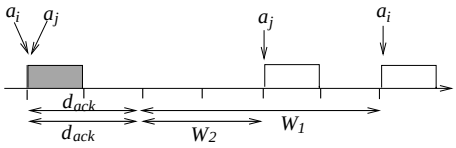
$$\rho = \frac{E(S)}{E(I) + E(S) + E(L)} = \lambda T e^{-2\lambda T}\,.$$

A kihasználtság maximuma $\lambda T = 0.5$ mellett $\rho = 1/2e \sim 0.18394$. Látható, hogy a 0.5-nél nagyobb terhelés esetén a kihasználtság csökken, így ezeknél a rendszereknél ügyelni kell arra, hogy a felhasználók eredő forgalma ne emelkedjen e fölé a szint fölé.

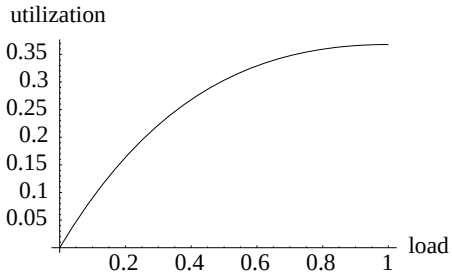
Egy Δ hosszú intervallumban az érkező csomagok várható száma $\lambda\Delta$. Ugyanezen idő alatt a sikeres csomagtovábbítás várható ideje $\rho\Delta$, amennyi idő alatt $\lfloor \rho\Delta/T \rfloor$ csomagot lehet átvinni. Ezek alapján $\Delta \to \infty$ esetén az egy sikeresen átvitt csomagra jutó csomagtovábbítási kísérletek száma, azaz a csomagküldési kísérletek várható száma $E(R) = \lambda T/\rho = e^{2\lambda T}$.

A csomagküldési kísérletek száma (R) alapján a csomagkésleltetési idő a következőképpen adódik

$$\begin{aligned} E(D) &= E\Big(\sum_{r=1}^{\infty} P(R = r)\Big(rT + \sum_{i=1}^{r-1} d_{ack} + W_i\Big)\Big) \\ &= E(R)T + (E(R) - 1)(d_{ack} + E(W))\,. \end{aligned}$$



26.14. ábra. Diszkrét idejű ALOHA rendszer.



26.15. ábra. Diszkrét idejű ALOHA rendszer kihasználtsága.

Diszkrét idejű ALOHA rendszer

A folytonos idejű rendszer egyik legnagyobb hátránya, hogy a 26.10.–26.12. ábrákon sötét szürke színnel jelöl valódi ütközéseken túl a rendszer számára elveszett idő az ábrákon világos szürkével jelölt időszak is, amelyek alatt a felhasználók az egyébként ütközés miatt elvesző csomagjaikkal csökkentik a további csomagok sikeres továbbításának valószínűségét.

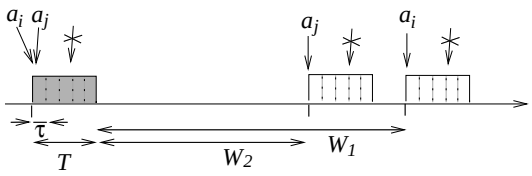
Az ALOHA eljárás egy egyszerű módosításával megszüntethető ez a veszteség. Abban az esetben, ha minden felhasználó csak szinronizált időrésekben küld csomagot, akkor az ütköző csomagok legfeljebb T ideig foglalják a csatornát. Ebben az esetben az ütközés esetén sorsolt véletlen késleltetés T egész számú többszöröse. Ezt a rendszert szokták réselt ALOHA rendszernek is nevezni.

A diszkrét ALOHA rendszer (lásd a 26.14. ábrát) „0-adrendű modellje” azt feltételezi, hogy a felhasználók λT paraméterű Poisson-eloszlású csomagot (új és ismételt együttesen) probálnak elküldeni minden időrésben. Ez a model megfelel a folytonos idejű ALOHA rendszer 0-adrendű modelljének, azzal a kiegészítéssel, hogy a felhasználók egy T idejű időrésben Poisson érkezési folyamat szerint keletkező csomagjaikat a következő időrésben küldik el. E feltételezés mellett egyetlen időrés vizsgálata alapján meghatározhatók a lényeges teljesítmény jellemzők. Jelölje N az időrésben érkező csomagok számát.

$$\rho = P(\text{sikeres csomagátvitel}) = P(N = 1) = \lambda T e^{-\lambda T}$$

A kihasználtság maximuma $\lambda T = 1$ mellett $\rho = 1/e \sim 0.367879$. A folytonos idejű rendszerrel szemben itt a terhelés a rendszer kapacitásának szintjéig ($\lambda T = 1$) növelhető a kihasználtság növekedése mellett.

A csomag ismétlések (R) átlagos számát az időrésben sikeresen átvitt csomagok és az összes csomag aránya adja



26.16. ábra. CSMA rendszer működése.

$$E(R) = \frac{E(N)}{E(\text{sikeres csomagok})} = \frac{\lambda T}{\lambda T e^{-\lambda T}} = e^{\lambda T} .$$

A késleltetési idő pedig a folytonos idejű ALOHA rendszerhez hasonlóan

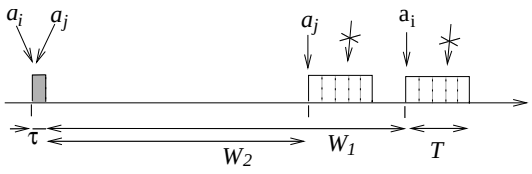
$$\begin{aligned} E(D) &= E\Big(\sum_{r=1}^{\infty} P(R=r)\Big(rT + \sum_{i=1}^{r-1} d_{ack} + W_i\Big)\Big) \\ &= E(R)T + (E(R)-1)(d_{ack} + E(W)) . \end{aligned}$$

Az ALOHA rendszerek pontosabb modelljei megkülönböztetik a felhasználók bizonyos állapotait (csomag előkészítés, új csomag küldés, várakozás, késleltetés, csomag ismétlés) és ez alapján határozzák meg az érkező új és ismételt csomagok számát [79].

26.11.2. CSMA és CSMA/CD

Az ALOHA protokoll család további tagjai az adott fizikai rendszerben a felhasználók számára elérhető információk segítségével javítják a csatorna kihasználtságát.

A folytonos és a diszkrét idejű ALOHA rendszer vizsgálata során már láttuk, hogy javítható a kihasználtság, ha az ütköző csomagok kevesebb ideig foglalják a csatornát. Rádiós rendszerekben nem mindig biztosított, hogy minden felhasználó minden más felhasználó adását venni tudja. Vezetékes rendszerekben viszont a terjedési idő leteltével minden felhasználó értesülhet arról, hogy valaki csomagot továbbít a csatornán. Így egyrészt elkerülhető az, hogy egy felhasználó csomagtovábbításáról értesülve egy másik felhasználó is adni kezdjen (**C**arrier **S**ense **M**ultiple **A**ccess, CSMA), másrészt, ha a terjedési idő kisebb, mint a csomagtovábbítási idő, akkor az esetleges ütközések még a csomagküldés alatt kiderülnek, ha a felhasználók adás alatt is vizsgálják a csatorna állapotát (with **C**ollision **D**etection, CD). A 26.16. és 26.17. ábrák a rendszer diszkrét idejű változatát szemléltetik, ahol a maximális jel terjedési idő τ, a csomagküldések szinkronizált τ hosszú időrések elején kezdődhetnek, és csomagütközés csak az első τ időrésben lehetséges, mert a következő időrésben már minden felhasználó tudja, hogy csomagtovábbítás zajlik a csatornán. A felhasználók csak akkor küldhetnek csomagot, ha úgy tapasztalják, hogy a csatorna üres, így csak azonos τ időrésben kezdődő csomagok ütközhetnek. Az eljárás ütközés detekció nélküli változatában a megkezdett csomagok küldése ütközés esetén is befejeződik (26.16. ábra), míg ütközés detekció esetén az ütköző csomagok továbbítása az ütközés felismerésekor azonnal leáll (26.17. ábra).



26.17. ábra. CSMA/CD rendszer működése.

Diszkrét idejű CSMA rendszer

A rendszer „0-ad rendű modelljének” elemzését egymást követő csomagküldési kísérletek közti intervallumokat vizsgálva végezzük. A 26.16. és 26.17. ábrákon az intervallumok kezdetét az időtengely alatti vonalkák jelzik. Előfordulhat az is, hogy az ábrától eltérően nincs üres intervallum két egymást követő csomagtovábbítási kísérlet között. A „0-adrendű modell” esetünkben azt feltételezi, hogy minden üres vagy befejező intervallum utáni τ hosszú intervallumban $\tau\lambda$ paraméterű Poisson-eloszlású csomagot küldenek a felhasználók. Tehát, az eddigi 0-adrendű modellekkel szemben a csatorna állapota befolyásolja az érkezési folyamatot.

Az átvitel sikerességét a vizsgált intervallum első τ hosszú szeletében érkező csomagok száma (N) határozza meg.

$$P(\text{sikeres csomagátvitel}) = P(N = 1|N \geq 1) = \frac{\lambda\tau e^{-\lambda\tau}}{1 - e^{-\lambda\tau}} .$$

A sikeres, vagy ütköző csomagok küldése után a csatorna addig üres marad, amíg újabb csomagküldés nem kezdődik. Jelölje ezt az üres időt I, ahol I ($I \in \{0, \tau, 2\tau, \ldots\}$) geometriai eloszlású, azaz

$$P(I = i\tau) = e^{-\lambda\tau i}(1 - e^{-\lambda\tau}) .$$

Így a vizsgált intervallumban

- az üres intervallum várható hossza $E(I) = \dfrac{\tau\, e^{-\lambda\tau}}{1 - e^{-\lambda\tau}}$,
- a sikeres csomagtovábbítás várható ideje $E(S) = \dfrac{T\ \lambda\tau e^{-\lambda\tau}}{1 - e^{-\lambda\tau}}$,
- és a sikertelen csomagtovábbítás várható ideje $E(L) = \dfrac{T\ (1 - (1 + \lambda\tau)e^{-\lambda\tau})}{1 - e^{-\lambda\tau}}$,

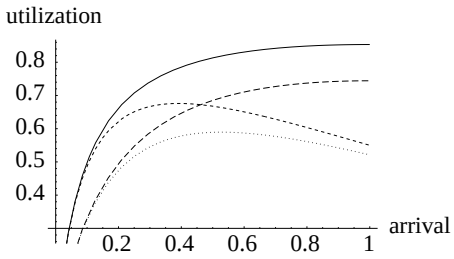
amiből a rendszer kihasználtsága

$$\rho = \frac{E(S)}{E(I) + E(S) + E(L)} = \frac{T\ \lambda\tau e^{-\lambda\tau}}{T(1 - e^{-\lambda\tau}) + \tau e^{-\lambda\tau}} .$$

A 26.18. ábra $\lambda\tau$ függvényében ábrázolja a kihasználtságot $\tau/T = 0.2$ (pontvonal), illetve $\tau/T = 0.1$ (rövid szaggatott vonal) esetén. Látható, hogy rövidebb terjedési idő esetén kisebb az ütközés esélye, és ezért nő a kihasználtság.

Diszkrét idejű CSMA/CD rendszer

Egy „0-ad rendű modellt” vizsgálva a diszkrét idejű CSMA/CD rendszer csak annyiban



26.18. ábra. Diszkrét idejű CSMA és CSMA/CD rendszer kihasználtsága.

különbözik a diszkrét idejű CSMA rendszertől, hogy az ütközés érzélelése miatt a sikertelen csomagtovábbítás várható ideje lecsökken

$$E(L) = \frac{\tau\,(1-(1+\lambda\tau)e^{-\lambda\tau})}{1-e^{-\lambda\tau}}\,.$$

Ennek hatására a rendszer kihasználtsága

$$\rho = \frac{E(S)}{E(I)+E(S)+E(L)} = \frac{T\,\lambda\tau e^{-\lambda\tau}}{T\lambda\tau e^{-\lambda\tau}+\tau(1-\lambda\tau e^{-\lambda\tau})}\,.$$

A 26.18. ábra a CSMA rendszer kihasználtságával együtt ábrázolja a CSMA/CD rendszer kihasználtságát $\lambda\tau$ függvényében $\tau/T = 0.2$ (hosszú szaggatott vonal), illetve $\tau/T = 0.1$ (folytonos vonal) esetén. A rövidebb terjedési idő ismét növeli a kihasználtságot, és a CSMA rendszerrel összehasonlítva azt látjuk, hogy ütközési időszak csökkentése is növeli a kihasználtságot nagyobb terhelések esetén.

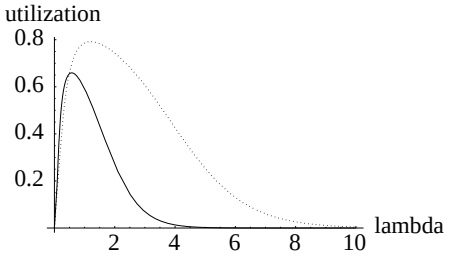
Diszkrét kitartó CSMA/CD rendszer

A CSMA rendszerek eddigi vizsgálata során nem tértünk ki arra a kérdésre, hogy mit csinálnak azok a felhasználók, akiknek küldendő új vagy késleltetett csomag küldési szándéka van akkor, amikor a csatornát más felhasználó foglalja éppen. Az eddigiekben implicit módon azt feltételeztük, hogy ezek a felhasználók úgy járnak el, mintha csomagjuk ütközött volna, és egy véletlen késletetési idő múlva próbálkoznak újra. A gyakorlatban ezt az esetet nem-kitartó (non-persistent) viselkedésnek nevezik.

A felhasználók azonban a csomagméret ismeretében azt is tudják, hogy mikor fejeződik be az éppen zajló csomag továbbítása, mivel minden sikeres csomag T ideig foglalja a csatornát. Ilyenkor a véletlen késletetési időhöz képest csökkenthető a csomagkésleltetés ideje, ha a csatorna szabaddá válása utáni időrésben próbál adni az a felhasználó, akinek a foglalt időszakban keletkezett átviendő csomagja. Ez a viselkedést nevezik kitartó viselkedésnek.

A kitartó CSMA rendszer „0-adrendű modelljében” azt feltételezzük, hogy minden τ hosszú időrésben $\tau\lambda$ paraméterű Poisson-eloszlású csomagot küldenének a felhasználók, és akiknek akkor keletkezik csomagja, amikor a csatorna foglalt, azok a csatorna szabaddá válását követő időrésben probálják továbbítani a csomagot.

Ennek a rendszernek a vizsgálatát felbonthatjuk a sikeres (S), az ütköző (L), és az üres (I) szakaszok vizsgálatára, mivel ezeknek a szakaszoknak a kezdetén memóriamentes a rendszer. Az egyes szakaszok hossza egyszerűen számítható. A sikeres szakasz hossza



26.19. ábra. Diszkrét idejű kitartó CSMA/CD rendszer kihasználtsága.

$E(S) = T$, az ütköző szakasz hossza $E(L) = \tau$, és az üres szakasz várható hossza $E(I) = (1 - e^{-\lambda\tau})^{-1}$. A Π állapot átmeneti mátrix megadja, az egyes szakaszok milyen valószínűséggel követik egymást.

$$\Pi = \begin{array}{|c|c|c|c|} \hline S & L & I & \\ \hline P(\lambda T,1) & P(\lambda T,>1) & P(\lambda T,0) & S \\ P(\lambda\tau,1) & P(\lambda\tau,>1) & P(\lambda\tau,0) & L \\ \dfrac{P(\lambda\tau,1)}{P(\lambda\tau,>0)} & \dfrac{P(\lambda\tau,>1)}{P(\lambda\tau,>0)} & 0 & I \\ \hline \end{array}$$

ahol $P(a,i) = e^{-a}a^i/i!$ és $P(a,>i) = \sum_{j=i+1}^{\infty} P(a,j)$ a Poisson-valószínűségeket jelöli. A $\pi\Pi = \pi$ egyenletet megoldva,

$$\begin{aligned}\rho &= \frac{\pi_S E(S)}{\pi_S E(S) + \pi_L E(L) + \pi_I E(I)} \\ &= \frac{e^{\lambda T}\lambda^2\tau T}{e^{\lambda T} - \lambda T + \lambda\tau\Big(1 - \lambda\tau + \lambda T - e^{\lambda\tau}\lambda T + e^{\lambda T}\left(-1 + e^{\lambda\tau} + \lambda T\right)\Big)} \\ &= \frac{\lambda\tau\,\lambda T}{1 - \lambda T e^{-\lambda T} + \lambda\tau e^{-\lambda T}\Big(1 - \lambda\tau + \lambda T(1 - e^{\lambda\tau})\Big) + \lambda\tau\Big(e^{\lambda\tau} + \lambda T - 1\Big)}\end{aligned}$$

alakban kapjuk a rendszer kihasználtságát.

A 26.19. ábra λ függvényében ábrázolja a kihasználtságot $\tau/T = 0.1$ (pontvonal) illetve $\tau/T = 0.2$ (folytonos vonal) esetén. Az előző esetekhez hasonlóan rövidebb terjedési idő esetén kisebb az ütközés esélye, ezért nő a kihasználtság és az optimális kihasználtsághoz tartózó érkezési intenzitás is.

Összességében a kitartó viselkedés akkor előnyös, ha a csomagtovábbítási intervallumok végén keletkező ütközések miatti késleltetés kisebb, mint a nem kitartó viselkedésből adódó késleltetés. Kis forgalom esetén a kitartó viselkedés csökkenti a késleltetési időt, míg nagy forgalom esetén a nem kitartó eljárás hatékonyabb.

A kitartó és a nem kitartó eljárások között folytonos átmenetet képez a "p kitartó" eljárás, amelyben minden csomagküldési kísérlet alkalmával a felhasználó p valószínűséggel

kitartó, $1-p$ valószínűséggel pedig nem kitartó eljárást követ foglalt csatorna esetén. Látható, hogy $p = 1$ esetén a kitartó, $p = 0$ esetén a nem kitartó eljárást kapjuk, és a köztes tartományban p segítségével beállíthatjuk a sikeres csomagátvitel utáni időresben érkező csomag intenzitást egy optimális értékre.

26.11.3. IEEE 802.11

A napjainkban rohamosan terjedő vezetéknélküli számítógép hálózati hozzáférést biztosító eljárás (wireless fidelity, WF), amit az IEEE 802.11 azonosító számon szabványosított, szintén a diszkrét idejű (réselt) ALOHA eljárás egy módosított változata. Az eljárás tervezésékor többek között a következő szempontokat próbálták érvényesíteni:

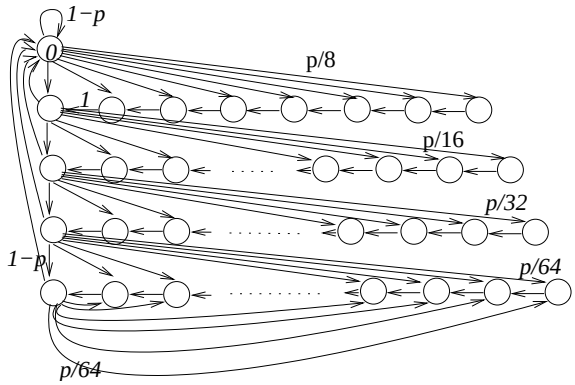
- a sorsolt késleltetési idő korlátos,
- a forgalom változására adaptív,
- prioritás a megkezdett csomagok átvitelére.

Ennek megfelelően az eredeti ALOHA eljárást többek között a következő pontok szerint módosították.

- Ütközéskor a felhasználó egy 1 és M_i közötti számot sorsol egyenletes valószínűséggel és a sorsolás szerinti üres időrés letelte után probálkozik újra adatküldéssel.
- A sorsolás felső határa függ az adott csomag átvitelére eddig tett kísésrletek számától. Az első ütközés után ez az érték $M_1 = 8$, és minden következő ismétlésnél duplázódik a sorsolás felső határa, azaz $M_i = 8 * 2^{i-1}$ mindaddig, amíg el nem éri az M_{max} értékét.
- A nagyobb adatcsomagok átvitele részekre (ún. szegmensekre) bontva történik. Az ALOHA rendszerben a szegmenseket csak egyesével, egymás után lehet továbbítani úgy, hogy mindegyik szegmensnek külön kell versenyeznie a csatornáért. Ilyenkor az egész csomag átvitelének ideje elfogadhatatlanul nagyra nőhet. Az IEEE 802.11 eljárás ezért lehetőséget biztosít a felhasználóknak arra, hogy ha egyszer magukhoz ragadták a csatornát, akkor további verseny nélkül elküldhessék egy csomag összes szegmensét. A protokol ezt úgy valósítja meg, hogy a felhasználók akkor tekintik szabadnak a csatornát, ha azon DIFS (distributed interframe space) ideig nem volt adattovábbítás, míg azonos csomag szegmenseit egy felhasználó SIFS (short interframe space) időközökkel küldheti, és eközben a többi felhasználó végig foglaltnak látja a csatornát, mert SIFS<DIFS.

Az IEEE 802.11 szabvány az ALOHA eljárásra jellemző csatorna elemi hozzáférési módszert (basic access method) többféle erőforrás lefoglalási eljárással ötvözi, aminek az egyik példája az ismertetett szegmentált csomagtovábbítási eljárás. Ebben az anyagban az elemi hozzáférési módszer teljesítményelemzésére javasolt egyik lehetséges modellt ismertetjük. Ez a modell is egy alapvető modellezési közelítésre épül. Azt feltételezi, hogy a rendszerben olyan sok felhasználó van jelen, és ezek viselkedése annyira független egymástól, hogy egy megfigyelt felhasználó elküldött csomagja minden időrésben az előzményektől függetlenül p valószínűséggel ütközik.

A 26.20. ábrán látható diszkrét idejű Markov-lánc azt írja le, hogy a 0 állapotból indulva $M_1 = 8$ és $M_{max} = 64$ esetén, p ütközési valószínűség mellett a rendszer hogyan továbbíthatja a vizsgált felhasználó csomagját. A Markov-lánc 0 állapotához tartozó visszatérési idő, T_0, a csomagkésleltetés ideje időrésekben van kifejezve. A 0 állapot egyensúlyi



26.20. ábra. 802.11 elemi hozzáférésének Markov-lánc modellje.

valószínűsége p_0 azt adja meg, hogy időrésenként maximálisan hány csomagot tud átküldeni a felhasználó. A $p_0 = 1/E(T_0)$ összefüggés meghatározza a késleltetési idő várható értékét, és a Markov-lánc alapján a késleltetési idő magasabb momentumai és eloszlása is számítható.

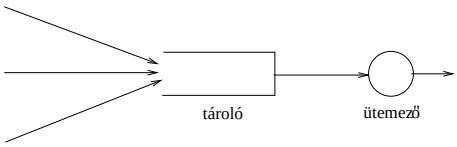
26.12. Sorbanállásos csomagtovábbítási rendszerek

Ebben a fejezetben a sorbanállás elmélet egy speciális részterületét vizsgáljuk, amelyben az "igényeknek" csomagok, a "kiszolgálónak" pedig adott kapacitású átviteli rendszer felel meg, amely kapacitását teljes egészében a kiszolgálás alatt lévő csomag továbbítására használjuk. Az alapvető sorbanállási modellekhez képest azonban az a leglényegesebb eltérés, hogy megkülönböztetjük az egyes forrásokból érkező csomagokat, és vizsgálataink célja nem az egész rendszerre, hanem az egyes források csomagjaira vonatkozó teljesítményjellemzők elemzése.

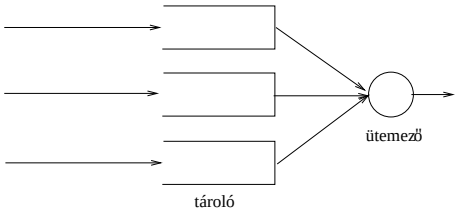
A fejezet elején bevezetett kiszolgálási elveket az alábbi csoportosításnak megfelelően tárgyaljuk, ahol az egymást követő eljárások mind rugalmasabb lehetőséget biztosítanak a különböző források csomagjainak eltérő kiszolgálására.

1. Sorrendi kiszolgálás

   A sorrendi kiszolgálás a csomagokat az érkezési sorrend szerint továbbítja. A gyakorlati megvalósítás részleteitől függetlenül a viselkedést tekinthetjük úgy, mintha egy közös tárolóban egymás után várakoznának a különböző forrásokból érkezett csomagok (26.21. ábra). Mivel ebben a rendszerben a kölünböző források csomagjainak kiszolgálását csak az érkezési pillanatuk határozza meg, ezért semmilyen forgalom megkülönböztetés nem történik, mindegyik forrás számára ugyanazok lesznek a kiszolgálás minőségi jellemzői (pl. várakozási idő). Ennek a kiszolgálási elvnek a vizsgálatára közvetlenül felhasználhatók az elemi sorbanállási eredmények, melyekre itt nem térünk ki.



26.21. ábra. Sorrendi kiszolgálás.



26.22. ábra. Kiszolgálás forgalom megkülönböztetéssel.

2. PRIORITÁSOS KISZOLGÁLÁS
   A prioritásos kiszolgálás során a forrásokat forgalmi osztályokba soroljuk, és a forgalmi osztályokhoz különböző prioritási szintet rendelünk. Az azonos forgalmi osztályba tartozó csomagokat továbbra sem különböztetjük meg (egymáshoz képest továbbra is érkezesi sorrendben kerülnek kiszolgálásra), de a kiszolgáló minden esetben törekszik a nagyobb prioritású csomagokat előbb továbbítani. A rendszer működését tekinthetjük úgy, mintha a különböző osztályba tartozó csomagok külön tárolókban várakoznának, és a kiszolgálónak mindig azt kell eldöntenie, hogy melyik sorban lévő csomagot továbbítsa (26.22. ábra). Ezt a feladatot, ahol a tárolókban várakozó csomagok továbbításának sorrendjét kell eldönteni, szokták ütemezésnek is nevezni. A prioritásos kiszolgálás során az ütemezést egyetlen paraméter, a prioritási szint alapján dönti el a kiszolgáló.

3. SÚLYOZOTT ERŐFORRÁS MEGOSZTÁS
   A prioritásos kiszolgálás rugalmasságának növelése érdekében az ütemezési döntésnél a további szempontok is figyelembe vehetők. A súlyozott erőforrás megosztás során az ütemező úgy választ a kiszolgálandó csomagok közül, hogy egy vizsgált időintervallumra vonatkozóan minden forgalmi osztályra, amelyben van továbbítandó adatcsomag, a súlyának megfelelő kiszolgálási időarány jusson.

Az egymás után felsorolt kiszolgálási módszerek mind általánosabbak, és speciális esetként tartalmazzák az őket megelőző módszereket. Hiszen, ha egy prioritásos rendszerben minden forgalmat azonos forgalomosztályba sorolunk, akkor sorrendi kiszolgálást nyerünk, illetve, ha egy két osztályos súlyozott erőforrás megosztási rendszerben az egyik osztályhoz 1, a másikhoz 0 súlyt rendelünk, akkor prioritásos kiszolgáláshoz jutunk. Többszintű prioritás többlépcsős súlyozott rendszerrel valósítható meg.

26.12.1. Prioritásos kiszolgálás

A prioritásos kiszolgálás során az ütemezőnek semmilyen bonyolult számítási feladatot nem kell elvégeznie a következő kiszolgálandó igény kiválasztásához, hiszen mindig a legnagyobb prioritású nem üres sorból kell kiszolgálni az első igényt.

A módszernek két változata lehetséges, attól függően, hogy egy megkezdett kisebb prioritású igény kiszolgálását egy nagyobb prioritású igény érkezésekor felfüggesztjük-e vagy sem. Az első esetet megszakításos prioritásnak (preemptive priority), a másodikat megszakításmentes prioritásnak (non-preemptive priority) nevezzük.

A csomagtovábbító rendszerek többségében az egyszerű és gyors működés biztosítása érdekében megszakításmentes prioritást alkalmaznak, hiszen a csomagtovábbítás megszakítása esetén bonyolult párbeszédet (protokollt) kell kialakítani a forrás és a vevő között a csomagrészek megfelelő azonosítására.

A megszakításos rendszereket megkülönböztethetjük a megszakítás mechanizmusa alapján is. Az úgy nevezett munkamegörző (work conserving) rendszerekben a megszakításkor nem vész el a szerver által a megszakításig elvégzett munka, azaz a megszakításig átvitt adatokat nem kell megismételni a magasabb prioritású igény kiszolgálása után. A munkavesztéses (non work conserving) rendszerekben a megszakított igények átvitelét előről kell kezdeni a magasabb prioritású igény kiszolgálása után.

A munkamegőrző megszakításos prioritásos rendszerek elemzése exponenciális eloszlású csomagtovábbítási idők esetén egyszerűbb, mint az megszakításmentes rendszer elemzése, mivel a rendszerben lévő igények típusa egyértelműen meghatározza, hogy melyik igényt szolgáljuk ki éppen, ami a megszakításmentes kiszolgálás esetén nem teljesül. Nem exponenciális eloszlású csomagtovábbítási idők esetén azonban a megszakításmentes eset elemzése egyszerűbb, mert a megszakításos rendszerben a hátralévő kiszolgálási idők függenek a megszakításig eltelt időtől, ami (több prioritási szint esetén) több folytonos változótól való függőséget vezet be.

A Markovi (minden prioriás osztályban Poisson érkezési folyamat és exponenciális kiszolgálási idő eloszlás) munkamegőrző megszakításos prioritásos rendszerek elemzését elvégezhetjük a rendszert leíró Markov-lánc vizsgálatával, amire a 26.23. ábra mutat példát két prioritási szint és végtelen tárolók esetén. Az ábrán lévő Markov-lánc a 2-es osztály igényeit csak akkor tobábbítja, amikor nincs 1-es osztályú igény.

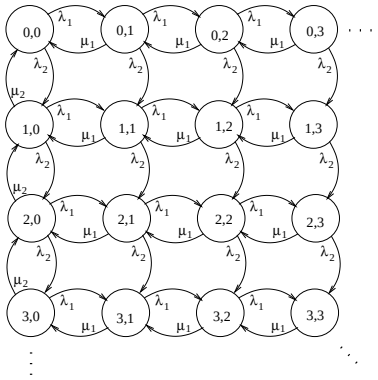
A Markovi munkamegőrző prioritásos rendszerek teljesítmény jellemzői közül az egyes osztályok várható késleltetési idejét elemezzük.

Jelölje $r \in \{1, \ldots, R\}$ a prioritás szintet, úgy, hogy a legnagyobb prioritás szint 1, λ_r és μ_r az r szintű csomagok érkezési és kiszolgálási intenzitását, ebből adódóan a prioritás szint által okozott terhelés $\rho_r = \lambda_r/\mu_r$. $\overline{K}_r$ és $\overline{T}_r$ jelöli a rendszerben lévő r szintű igények várható számát, és azok rendszerben eltöltött várható idejét.

Egy megszakításos munkamegőrző rendszerben egy adott szintű igény rendszerben eltöltött várható ideje ($\overline{T}_r$) a következő tényezőket tartalmazza:

- az igény kiszolgálási ideje:

$$\frac{1}{\mu_r},$$



26.23. ábra. Megszakításos prioritásos kiszolgálás Markov lánca 2 prioritásosztály esetén.

- az igény érkezésekor a rendszerben lévő vele azonos és magasabb prioritású igények kiszolgálási ideje:

$$\sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} ,$$

- az igény *kiszolgálási* ideje alatt érkezett magasabb prioritású igények kiszolgálási ideje:

$$\sum_{s=1}^{r-1} \frac{\overline{T}_r \lambda_s}{\mu_s} .$$

Kihasználva, hogy $\rho_r = \lambda_r/\mu_r$ a

$$\overline{T}_r = \frac{1}{\mu_r} + \sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} + \sum_{s=1}^{r-1} \frac{\overline{T}_r \lambda_s}{\mu_s}$$

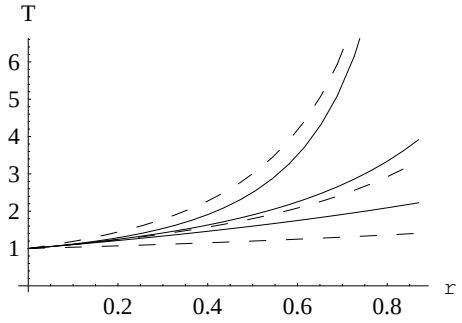
egyenlet megoldása $\overline{T}_r$-re

$$\overline{T}_r = \frac{\frac{1}{\mu_r} + \sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s}}{1 - \sum_{s=1}^{r-1} \rho_s} .$$

Egy hasonló, de megszakításmentes munkamegőrző rendszerben egy adott szintű igény rendszerben eltöltött várható ideje ($\overline{T}_r$) a következő tényezőket tartalmazza:

- az igény kiszolgálási ideje: $\dfrac{1}{\mu_r}$,
- az igény érkezésekor kiszolgálás alatt lévő igény hátralévő kiszolgálási ideje:

$$\sum_{s=1}^{R} \frac{\rho_s}{\mu_s} ,$$



26.24. ábra. Forgalom osztályok késleltetése megszakításos és megszakításmentes prioritás esetén a kihasználtság függvényében.

- az igény érkezésekor a rendszerben lévő vele azonos és magasabb prioritású igények kiszolgálási ideje:

$$\sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} ,$$

- az igény *várakozási* ideje alatt érkezett magasabb prioritású igények kiszolgálási ideje:

$$\sum_{s=1}^{r-1} \frac{(\overline{T}_r - \frac{1}{\mu_r})\lambda_s}{\mu_s} .$$

Az így nyert implicit egyenlet megoldása $\overline{T}_r$-re

$$\overline{T}_r = \frac{1}{\mu_r} + \frac{\sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} + \sum_{s=r+1}^{R} \frac{\rho_s}{\mu_s}}{1 - \sum_{s=1}^{r-1} \rho_s} .$$

A rendszerparaméterek (λ_r, μ_r) alapján a teljesítmény jellemzők $(\overline{K}_r, \overline{T}_r)$ meghatározása az első prioritási szintről indulva rekurzívan, a Little-szabály $(\overline{T}_r = \lambda_r \overline{K}_r)$ felhasználásával lehetséges.

A 26.24. ábra szemlélteti a rendszerben töltött átlagos idő alakulását a kihasználtság függvényében 3 prioritás szintre megszakításos és megszakításmentes esetben, amikor $\lambda_1 = \lambda_2 = \lambda_3 = \rho/3$, $\mu_1 = \mu_2 = \mu_3 = 1$. Az ábrán a szaggatott vonalak jelzik a prioritás osztályok késleltetését megszakításos és a folytonos vonalak megszakításmentes prioritás mellett. Az elvárásoknak megfelelően a magasabb prioritású forgalom késleltetési ideje kisebb az alacsonyabb prioritásúakénál. Látható továbbá, hogy a nagy prioritású forgalom késleltetése csökken és a kis prioritású forgalom késleltetése nő a megszakításos rendszerben a megszakításmentes rendszerhez képest.

A prioritásos rendszerek tárgyalásának végén összegezzük e rendszerek viselkedésének lényeges tulajdonságait.

- Előnyök:
  A prioritásos kiszolgálás kis bonyolultságú forgalommegkülönböztetést végző eljárás, amelynek jellemzője, hogy jelentős különbséget tesz az egyes forrásosztályok kiszolgálása között.
- Hátrányok:
  A prioritásos kiszolgálási módszerből adódóan a kis prioritású forgalom továbbítása teljesen megáll, amíg a nagyobb prioritási szinteken van továbbítandó adat. Ha a magas prioritási szintek forgalma olyan, hogy előfordulhatnak benne hosszú aktív periódusok, akkor ez az alacsony prioritású forgalom hosszú idejű blokkolását úgy nevezett kiéheztetését eredményezi. Az esetek többségében távközlési rendszerekben törekszenek az ilyen kiéheztetés elkerülésére akkor is, ha a kis prioritású adatokat nem kell nagyon szoros határidőn belül átvinni, mert a forgalomforrásokban a kiéheztetés és más hálózati hibák hatása nem különböztethető meg.

  A megszakításos prioritás esetén az alacsony prioritású forgalom semmilyen hatást nem gyakorol a magas prioritású forgalom továbbítására, azonban ez már nem igaz a megszakításmentes prioritásos rendszereknél. A legnagyobb gondot az okozza, ha a nagy prioritású (általában kis) csomagokat szoros határidővel kell továbbítani, de a nagy méretű, alacsony prioritású csomagok (a csomag méretével arányosan) hosszú ideig foglalják a kiszolgálót abban az esetben, ha a kiszolgálásuk megkezdődött már a nagy prioritású csomag érkezésekor.

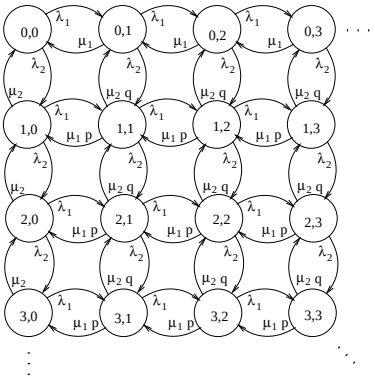
Főleg az első hátrány, a kiéheztetés megakadályozására dolgozták ki a súlyozott erőforrás megosztási eljárásokat. Megszakításmentes esetben a magas prioritású csomagok késleltetése úgy csökkenthető, hogy az alacsony prioritású csomagokat kisebb méretű adatcsomagokban továbbítjuk.

26.12.2. Súlyozott erőforrás megosztás

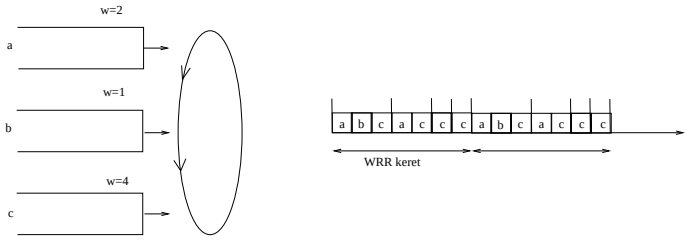
A súlyozott erőforrás megosztás elvileg a kiszolgáló kapacitás olyan szétosztására törekszik, amelyben a források súlyuk szerint osztoznak a kiszolgáló kapacitáson mindaddig, amíg van továbbítandó adatuk. Két forgalmi osztály, Markovi viselkedés és p, valamint $q = 1 - p$ súlyok mellett a 26.25. ábrán látható Markov-lánc írja le ezt a rendszerviselkedését.

Az ismertetett elvi módszert az angol irodalomban *processor sharing* vagy *fluid fair queueing* néven tárgyalják. Az utóbbi elnevezés arra utal, hogy a kiszolgáló úgy viselkedik, mintha az átviendő adatot infinitezimális elemekre bontaná, és ezeket a súlyoknak megfelelő ütemezéssel vinné át. A gyakorlati csomagátviteli rendszerekben azonban nem ezt az elvi módszert alkalmazzák, hanem ennek teljes csomagok átvitelén alapuló közelítését. Olyan ütemező eljárásokat dolgoztak ki, amelyek hosszabb idő átlagában törekszenek a súlyoknak megfelelő erőforrás felhasználási arányok biztosítására, de minden csomagot egészben, a teljes átviteli kapacitás felhasználásával továbbítanak. Ezek közül az ütemezési eljárások közül mutatunk be néhányat a következő jelölések felhasználásával:

- $w_r, r \in \{1, R\}$ az r. forgalmi osztály súlya,
- $v_r, r \in \{1, R\}$ az r. forgalmi osztály normalizált súlya ($\sum_r v_r = 1$),
- C a kiszolgáló kapacitása,
- $T_{r,k}$ az r. forgalmi osztály k. csomagjának időbélyege,



26.25. ábra. Súlyozott erőforrásmegosztás Markov-lánca 2 forgalmi osztály esetén.



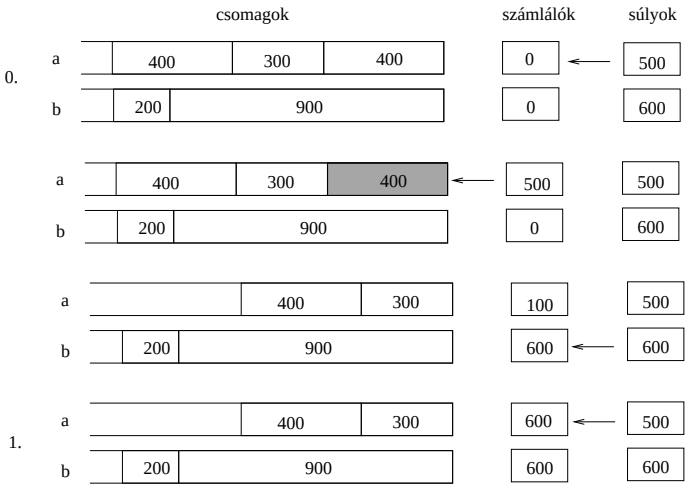
26.26. ábra. Súlyozott ciklikus ütemezés (WRR).

- t_{act} a vizsgált csomagérkezés ideje,
- $L_{r,k}$ az r. forgalmi osztály k. csomagjának mérete.

Az ütemezési eljárások egy része fix méretű csomagokkal (pl. ATM cellák) dolgozik. Ezeknek az eljárásoknak nem kell könyvelniük a kiszolgált csomagok méretét, csak azok számát. A változó méretű csomagokat továbbító eljárásoknak viszont a csomagok méretét figyelembe véve kell biztosítani a súlyoknak megfelelő erőforrás megosztást. Az ütemezők egy része a beérkező csomagokra *időbélyeget* számít ki, és az időbélyeg alapján ütemezi a kiszolgálást.

Súlyozott ciklikus ütemezés (Weighted round robin, WRR)
A WRR egy fix méretű csomagok továbbításánál alkalmazott ütemezési eljárás. Az eljárás egy WRR keretet állít össze, amely keretben minden forrás a súlyának megfelelő számú csomagot továbbíthat, és a kiszolgálási folyamatban ezt a keretet ciklikusan ismétli az ütemező (26.26. ábra).

Ciklikus ütemezés hátralék számítással (Deficit round robin, DRR)
A DRR eljárás a WRR módszer változó méretű csomagok kezelésére alkalmas kiegészített változata. Minden adási folyam kap egy hátralékszámlálót (*deficitcounter*) és egy küszöbszámot. A küszöbszám adja meg, hogy egy folyam mennyi bitet (vagy bájtot) adhat



26.27. ábra. Ciklikus ütemezés hátralék számítással (DRR).

egyszerre, amikor a sor rá kerül. Ha egy továbbításra váró csomag mérete meghaladja a küszöbszámot, akkor a küszöbszám értéke a hiányszámláló értékét növeli, így az adott folyam a következő sorrakerüléskor a megnövelt küszöbszám erejéig adhat. A szemléltetéshez tekintsük a 26.27. ábrát.

Látható, hogy ha az egyes esetekben egy adott folyam csomagja nagyobb, mint az aktuális adási méret (hiányszámláló érték), akkor az a nagy csomag előre engedi a másik folyam kisebb csomagját. A *Fair Queueing* filozófia ebben a formában érvényesül (a nagy csomag többet kénytelen várakozni a kisebb csomagnál, így arányos a várakozás időtartama a csomagmérettel). Az egyes források körbenforgó sorrendben követik egymást. A módszer lehetőséget ad súlyozásra az egyes folyamok küszöbértékének megkülönböztetésével.

Fontos szempont az ütemezési módszer kiválasztásánál az adott módszer számítási komplexitása, mivel a gyakorlatban az ütemezést nagyon gyorsan kell elvégezni. A *DRR* módszer előnye a számítás egyszerűsége, hiszen az ütemezési sorrend megállapításakor konstans számú művelet elvégzése szükséges. További előnye a módszernek, hogy változó csomagméret esetén is egyszerűen implementálható. A szakirodalomban megtalálható módszerek közül ez nem igaz mindegyikre.

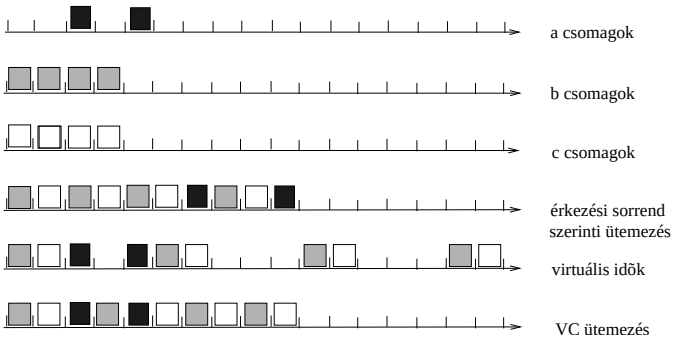
Virtuális óra alapú ütemezés (Virtual clock, VC)

A VC ütemezési eljárás mindegyik forgalmi osztályhoz saját, virtuális órát rendel. A virtuális óra minden egyes érkezéskor ugrik az adott forgalmi osztályra jellemző mértékben a

$$T_{r,k+1} = \max(t_{act}, T_{r,k}) + \frac{L_{r,k+1}}{C\ v_i}$$

összefüggés szerint. Az egyes csomagokat osztályuk virtuális órájának aktuális értékével időbélyegezzük. Az ütemező az időbélyegek növekvő sorrendjében szolgálja ki az egyes igényeket.

Az óra lépésköze forgalmi osztályonként változhat az adott osztályhoz rendelt kapaci-



26.28. ábra. Virtuális óra alapú ütemezés szemléltetése.

táshányadnak megfelelő mértékben. Szemléltetésre tekintsük a következő példát.

A kiszolgálandó igények három forgalmi osztályba tartoznak. Az egyszerűség kedvéért tegyük fel, hogy az egyes igények (csomagok) mérete állandó – így a kiszolgálás időtartama is. Az *a* osztályhoz rendeljük a kiszolgáló kapacitásának felét, a *b* és a *c* osztályhoz a kiszolgáló kapacitásának ötödét. Másként megfogalmazva: az *a* osztályból átlagosan két időegységenként továbbíthatunk csomagot, míg a másik két osztályból átlagosan öt időegységenként.

A 26.28. ábra mutatja mi történik, ha az egyes osztályok a dedikálttól eltérő intenzitással akarnak adni. Látható, hogy a dedikált kiszolgálás minőséget a VC elv jobban megvalósítja, mint például az érkezési sorrend szerinti kiszolgálás. Érkezési sorrend szerinti kiszolgálás esetén, ha egy osztály a megengedettnél több forgalmat generál, az a többi osztály kiszolgálásának a rovására megy, míg a VC ütemezésnél ez nem történik meg.

A stratégia szemléletes értelmezése lehet, hogy a szerver tulajdonképpen az egyes osztályok kiszolgált csomagjait számolja (a megfelelő súlyozás figyelembe vételével). Az elv ilyenformán az állandó csomagméretű hálózati technológiákra (pl. ATM) használható. Változó csomagméret esetén (pl. IP) annyi módosítást szükséges tenni, hogy ne a csomagok, hanem a bitek számát tartsa nyilván a kiszolgáló egység. Ebben az esetben csomagérkezéskor a virtuális óra lépésközét súlyozni kell a beérkezett csomag méretével, mint ahogy az a fenti képletben történik.

Több hasonló elven alapuló ütemezési eljárást alkalmaznak még a gyakorlatban, pl. starting potential fair queueing (SPFQ), worst-case fair weighted fair queueing (WF^2Q), self clock fair queueing (SCFQ), self clock fair queueing (SCFQ), amelyek ismeretetésére itt nem térünk ki.

Gyakorlatok

26.12-1. Határozza meg a csomagtovábbítás sorrendjét három forgalmi osztály esetén DRR eljárás mellett, ha az átviendő csomagok mérete:

- 1. osztály: 254, 234, 50, 654, 51,
- 2. osztály: 42, 134, 62, 54, 612, 57,
- 3. osztály: 155, 82, 69, 63, 152,

és az osztályok küszöbszámai rendre 40, 20, 50, illetve 20, 40, 50.

26.12-2. Mely csomagokat dobja el az $L = 3$, $I = 2$ paraméterű GCRA algoritmus az alábbi időpontokban érkező csomagok közül:

$t = 0, 1, 4, 5, 7, 8, 9, 10, 12, 13, 14, 15, 17, \ldots$

Feladatok

26-1. Sorhossz a távozási időpontokban

Tekintsük a korábban tárgyalt $G|G|1|\infty$ rendszert. Jelölje a korábbiak szerint s_n $(n = 1, 2, \ldots)$ azt az időpontot, amikor az n-edik igény kiszolgálása befejeződik és legyen $L_n = L(s_n)$, $n \geq 1$. Jelölje Z_n, $n = 1, 2, \ldots$ azon igények számát, amelyek az n-edik igény Y_n kiszolgálási ideje alatt lépnek be a rendszerbe, azaz $Z_n = N(s_n) - N(s_n - Y_n)$, $n \geq 1$. Igazoljuk, hogy az $\{L_n,\ n \geq 1\}$ sorozatra érvényes a következő rekurrens szabály:

$$L_n = I(L_{n-1} > 0)(L_{n-1} - 1) + Z_n = (L_{n-1} - 1)^+ + Z_n \, .$$

26-2. Várakozási idő

Tekintsük a $G|G|m|\infty$ tömegkiszolgálási rendszert, amely m kiszolgáló egységből és korlátlan számú várakozó helyből áll. Vezessük be az n-edik igény *várakozási vektorát*

$$W_n = (W_{n,1}, \ldots, W_{n,m}), \qquad n = 1, 2, \ldots,$$

ahol $W_{n,j}$ jelenti azt a véletlen időmennyiséget (v.v.-t), amennyit várakoznia kellene a t_n időpillanatban beérkező n-edik igénynek ahhoz, hogy i darab kiszolgáló egység felszabaduljon az összes korábban beérkezett $(1, \ldots, n-1$ sorszámú) igénytől. Legyen $W_1 = (W_{1,1}, \ldots, W_{1,m}) = 0$, vagyis a rendszer üresen kezd dolgozni. Jelölje tetszőleges $x = (x_1, \ldots, x_m) \in \mathbb{R}^m$ mellett

$$x^+ = (x_1^+, \ldots, x_m^+), \quad \text{ahol} \quad s^+ = \max(s, 0), \ \ s \in \mathbb{R}$$

és

$$R(x) = (x_{i_1}, \ldots, x_{i_n}), \quad x_{i_1} \leq x_{i_2}, \ldots \leq x_{i_n} \, ,$$

vagyis az $R(x)$ függvény az x vektorból komponenseiben nagyság szerint rendezett vektort képez. Vezessük be még az alábbi vektorokat

$$e = (\overset{(1)}{1},\ \overset{}{0}, \ldots,\ \overset{(m)}{0}), \qquad i = (\overset{(1)}{1},\ \ldots,\ \overset{(m)}{1}) \, .$$

Bizonyítandó, hogy a $\{W_n, n \geq 1\}$ sorozat a következő módon határozható meg:

$$W_{n+1} = R([(W_n + Y_n e) - X_n i]^+), \qquad n = 1, 2, \ldots .$$

26-3. Elutasításos rendszerek

$G|G|m|0$ elutasításos rendszerek. Ez a TKR m kiszolgáló egységből áll, egyáltalán nincs várakozó hely. Ebben az esetben csak virtuális értelemben beszélhetünk várakozásról, hiszen a rendszer elutasításos. Ekkor a W_n vektor $W_{n,j}$ komponense azt jelenti, hogy mennyi időt kellene várakoznia a t_n időpillanatban beérkező n-edik igénynek ahhoz, hogy i kiszolgáló egység felszabaduljon az összes korábban beérkezett ($1, \ldots, n-1$ sorszámú) igénytől (ha $W_{n,1} > 0$, akkor az n-edik nem kerül kiszolgálásra). Bizonyítandó, hogy ebben az esetben igaz a következő rekurrens összefüggés:

$$W_{n+1} = R([(W_n + Y_n I(W_{n,1} = 0)) - X_n i]^+) \; .$$

26-4. ALOHA teljesítmény elemzés

Tetszőleges szimulációs környezetben írjon szimulációs programot folytonos idejű ALOHA rendszer teljesítmény elemzésére. N felhasználó közül azok akik nincsenek blokkolt állapotban λ paraméterű exponenciális gondolkodási idő után generál egy csomagot, és annak sikeres átviteléig blokkolt állapotba kerül. A csomagok fix méretűek. Továbbításuk ideje T. Az ütközéseket követően az ütközött csomagok újraküldési késleltetése a $(0, b)$ intervallumon egyenletes eloszlású.

Szimulációs vizsgálat segítségével határozza meg b optimális értéket, amely mellett a a renszer átvitele maximális, ha $\lambda = 1$, $N = 3$ és $T = 0.1$.

26-5. GRCA algoritmus

Szimulációs vizsgálattal határozza meg, hogy független p paraméterű geometriai eloszlású ($a_i = p(1-p)^{i-1}$) időnként érkező csomagok hányad részét dobja el az $L = 10$, $I = 10$ paraméterű GCRA algoritmus, ha $p = 0.1$.

Megjegyzések a fejezethez

A tömegkiszolgálási rendszerek elméletének hatalmas szakirodalma van, ennek ellenére magyar nyelven viszonylag kevés mű jelent meg, pl. [150, 202, 329, 339]. A klasszikus tömegkiszolgálási rendszerekkel kapcsolatos eredményeket részletesen tárgyalja Kleinrock [202] könyve, az elmélet kezdeteinek részletes bibliográfiája megtalálható Saaty [303] könyvében. A speciális területek közül megemlítjük az ismétléses (retrial) rendszereket, amelyek esetén a visszautasított igény újabb kiszolgálást kezdeményezhet véletlen idő múlva (ld. Falin–Templeton [100]), vagy valamilyen szabályozott módon (lásd Lakatos [217]), valamint a centrális zárt rendszereket. Szeidl [327, 328] megmutatta, hogy amennyiben a centrális zárt rendszerben az összes kiszolgálás várható értéke véges, akkor a rendszer különböző mutatóinak létezik határeloszlása. A használt valószínűségszámítási eszközöket illetően a [288, 298], a Bessel-függvényekkel kapcsolatban a [234] könyvet ajánljuk.

A gyakorlatban felmerülő tömegkiszolgálási problémák bonyolultsága miatt gyakran kell a szimuláció eszközéhez nyúlnunk. Ekkor pszeudóvéletlenszám-generátor felhasználásával hozzuk létre a rendszert jellemző eloszlással (lásd pl. Deák [85], Gentle [134])

a beérkezési folyamatot és az egyes igények kiszolgálási idejét megadó véletlen számsorozatot. Számos tömegkiszolgálási rendszer rendelkezik a regeneratív tulajdonsággal, a valószínűségszámításból ismert felújításelméletre támaszkodva (lásd Feller, Karlin–Taylor [104, 186]) hatékonyan vizsgálható ezzel a módszerrel (ld. Crane, Lemoine [73] és Shedler [310]). Példaként említhetjük a centrális zárt rendszert (Szeidl [328, 327]) és az M|G|1 rendszer egyensúlyi eloszlásának meghatározását (Lakatos [218].)

A távközlési protokollok (a protokoll a távközlési kapcsolat kialakítását és működését leíró szabályrendszer) hierarchiájának elemeivel és megoldásaival foglalkozik [331], a témakörhöz kapcsolódó magyar nyelvű források közé tartozik [150, 179, 202, 329]. A sorbanállás-elmélet egy speciális részterületét alkotja a csomagok továbbítása, ebben az esetben megkülönböztetjük az egyes forrásokból érkező csomagokat és vizsgálataink célja az egyes források csomagjaira vonatkozó teljesítményjellemzők elemzése. A véletlen erőforrásokhoz való hozzáférés konfliktus feloldó ALOHA eljárásnak különböző algoritmusai ismertek [44, 79, 117]. Sorbanállásos munkamegőrző prioritásos rendszerek teljesítményjellemzői közül az egyes osztályok késleltetési idejét elemezzük [49]. Több ütemezési eljárást alkalmaznak a gyakorlatban, például [322] ismerteti ezen eljárások egy teljesítményelemzéssel kiegészített részletes összefoglalását.

IX. DISZKRÉT OPTIMALIZÁCIÓ

Bevezetés

Az első kötetben az *Ütemezéselmélet* című fejezet képviselte a diszkrét optimalizációt. Abban a fejezetben a klasszikus eredmények összefoglalását találja meg az Olvasó.

Ebben a második kötetben az ütemezéselmélet azon – viszonylag fiatal – részterületének eredményeit tárgyaljuk, ahol az ütemezési döntéseket a vizsgált rendszer eseményeire közvetlenül reagálva kell meghozni, és nincs lehetőség arra, hogy több (esetleg az összes) körülményt mérlegelve válasszunk optimális, vagy legalább viszonylag jó megoldást.

27. Versenyképességi elemzés

A gyakorlatban gyakran fordulnak elő olyan optimalizálási feladatok, ahol a bemenetet csak részenként ismerjük meg és a döntéseinket a már megkapott információk alapján, a további adatok ismerete nélkül kell meghoznunk.

Ilyen esetekben ***on-line feladatról*** beszélünk. Az on-line algoritmusok elméletének igen sok alkalmazása van a számítástudomány, a közgazdaságtan és az operációkutatás különböző területein.

Az on-line algoritmusok elméletéhez kapcsolódó első eredmények az 1970-es évekből származnak, majd a 90-es évek elejétől kezdve egyre több kutató kezdett el az on-line algoritmusok területéhez kapcsolódó feladatokkal foglalkozni. Számos részterület alakult ki és a legfontosabb, algoritmusokkal foglalkozó konferenciákon napjainkban is rendszeresen ismertetnek új eredményeket ezen témakörből. Jelen fejezetnek nem célja a témakör részletes áttekintése, ezen keretek között terjedelmi okokból ez nem is lenne lehetséges. Célunk néhány részterület részletesebb ismertetésén keresztül a legfontosabb algoritmustervezési technikák és bizonyítási módszerek bemutatása.

A következő alfejezetben az on-line algoritmusok elemzésénél használt alapvető fogalmakat definiáljuk. A legfontosabb definíciók ismertetését követően az egyik legismertebb on-line feladatot – a k-szerver feladatot – és néhány kapcsolódó eredményt ismertetünk. Ezt követően egy új területet mutatunk be, a számítógépes hálózatok vizsgálatához kapcsolódó on-line algoritmusok területét. Majd a ládapakolási feladatot és annak többdimenziós változatait ismertetjük. Végül a fejezet utolsó részében az ütemezéssel foglalkozunk és a területre vonatkozó alapvető eredményeket tekintjük át.

27.1. Fogalmak, definíciók

Mivel egy on-line algoritmusnak részenként kell meghozni a döntéseit a teljes bemenet ismerete nélkül, ezért egy ilyen algoritmustól nem várhatjuk el, hogy a teljes információval rendelkező algoritmusok által előállítható optimális megoldást szolgáltassa. Azon algoritmusokat, amelyek ismerik a teljes bemenetet, ***off-line algoritmusoknak*** nevezzük.

A többi algoritmushoz hasonlóan az on-line algoritmusok hatékonyságának vizsgálatára két alapvető módszert használnak. Az egyik lehetőség az ***átlagos eset elemzése.***

Ezen megközelítés hátránya, hogy általában nincs információnk arról, hogy a lehetséges bemenetek milyen valószínűségi eloszlást követnek. Mi ebben a fejezetben az átlagos eset elemzésével nem foglalkozunk.

A másik megközelítés egy legrosszabb eset korlát elemzés, amelyet ***versenyképességi elemzésnek*** nevezünk. Ebben az esetben az on-line algoritmus által kapott megoldás célfüggvényértékét hasonlítjuk össze az optimális off-line célfüggvényértékkel. A továbbiakban feltételezzük, hogy a célfüggvény pozitív. Mivel ebben a fejezetben ezt az elemzést használjuk, ezért az alábbiakban pontosan definiáljuk az alapvető fogalmakat.

Egy on-line minimalizálási feladat esetén egy on-line algoritmust ***C-versenyképesnek*** nevezünk, ha tetszőleges bemenetre teljesül, hogy az algoritmus által kapott megoldás költsége nem nagyobb, mint C-szer az optimális off-line költség. Egy ***algoritmus versenyképességi hányadosa*** a legkisebb olyan C szám, amelyre az algoritmus C-versenyképes.

A továbbiakban egy tetszőleges ALG on-line algoritmusra az I bemeneten felvett célfüggvényértéket ALG(I)-vel jelöljük. Az I bemeneten felvett optimális off-line célfüggvényértéket OPT(I)-vel jelöljük. Ezt a jelölésrendszert használva a versenyképességet minimalizálási feladatokra a következőképpen definiálhatjuk.

Az ALG algoritmus C-versenyképes, ha $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I)$ teljesül minden I bemenet esetén.

Szokás használni a versenyképesség további két változatát. Egy minimalizálási feladat esetén az ALG algoritmus ***enyhén C-versenyképes,*** ha van olyan B konstans, hogy $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I) + B$ teljesül minden I bemenet esetén.

Egy ***algoritmus enyhe versenyképességi hányadosa*** a legkisebb olyan C szám, amelyre az algoritmus enyhén C-versenyképes.

A versenyképességi hányados egy további változata az aszimptotikus versenyképességi hányados. Minimalizálási feladatok esetén az ALG algoritmus ***aszimptotikus versenyképességi hányadosa*** ($R^{\infty}_{\mathrm{ALG}}$) a következő formulákkal definiálható:

$$R^{n}_{\mathrm{ALG}} = \sup\{\frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \mid \mathrm{OPT}(I) = n\} ,$$

$$R^{\infty}_{\mathrm{ALG}} = \limsup_{n\to\infty} R^{n}_{\mathrm{ALG}} .$$

Egy algoritmust ***aszimptotikusan C-versenyképesnek*** nevezünk, ha az aszimptotikus versenyképességi hányadosa nem nagyobb mint C.

Megjegyezzük, hogy R^{n}_{ALG} és $R^{\infty}_{\mathrm{ALG}}$ az első kötet 11.3. alfejezetében bevezetett *hibafüggvény,* illetve *aszimptotikus hiba* speciális esetei (A = ALG, B = OPT).

Az aszimptotikus hányados fő tulajdonsága az, hogy azt vizsgálja, miként viselkedik az algoritmus akkor, ha a bemenet mérete végtelenhez tart. Ez azt jelenti, hogy az algoritmus – kis méretű bemenetekre mutatott – viselkedése nem befolyásolja az aszimptotikus hányadost.

A fentiekben a minimalizálási feladatokra definiáltuk a versenyképességi elemzés fogalmait. A definíciók hasonlóan értelmezhetők maximalizálási feladatok esetén is. Ekkor az ALG algoritmus C-versenyképes, ha $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I)$ teljesül minden I bemenet esetén, illetve enyhén C-versenyképes, ha valamely B konstans mellett $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I) + B$ teljesül minden I bemenetre. Az aszimptotikus hányadost maximalizálási feladatokra a következőképpen definiálhatjuk:

$$R^{n}_{\mathrm{ALG}} = \inf\{\frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \mid \mathrm{OPT}(I) = n\} ,$$

$$R^{\infty}_{\mathrm{ALG}} = \liminf_{n\to\infty} R^{n}_{\mathrm{ALG}} .$$

Ekkor egy algoritmust ***aszimptotikusan C-versenyképesnek*** nevezünk, ha az aszimptotikus versenyképességi hányadosa nem kisebb mint C.

Számos tudományos dolgozat vizsgál ***véletlenített on-line algoritmusokat,*** ebben az esetben az algoritmus által kapott célfüggvényérték egy valószínűségi változó és a versenyképességi hányados definíciójában ezen valószínűségi változó várható értéke szerepel. Mivel a fejezetben csak determinisztikus on-line algoritmusokkal fogunk foglalkozni, ezért a véletlenített algoritmusokra vonatkozó fogalmakat nem részletezzük.

27.2. A k-szerver feladat

Az egyik legismertebb on-line feladat a k-szerver feladat. A feladat általános definíciójának megadásához szükség van a metrikus tér fogalmára. Egy (M, d) párost, ahol M a metrikus tér pontjait tartalmazza, d pedig az $M \times M$ halmazon értelmezett távolságfüggvény, metrikus térnek nevezünk, ha a távolságfüggvényre teljesülnek az alábbi tulajdonságok:

- $d(x, y) \geq 0$ minden $x, y \in M$ esetén,
- $d(x, y) = d(y, x)$ minden $x, y \in M$ esetén,
- $d(x, y) + d(y, z) \geq d(x, z)$ minden $x, y, z \in M$ esetén,
- $d(x, y) = 0$ akkor és csak akkor teljesül, ha $x = y$.

A k-szerver feladatban adott egy metrikus tér és van k darab szerverünk, amelyek a térben mozoghatnak. A tér pontjaiból álló kérések egy listáját kell kiszolgálni azáltal, hogy a megfelelő kérések helyére odaküldünk egy-egy szervert.

A feladat on-line, ami azt jelenti, hogy a kéréseket egyenként kapjuk meg és az egyes kéréseket a további kérések ismerete nélkül azok érkezése előtt kell kiszolgálnunk. A cél a szerverek által megtett össztávolság minimalizálása. Ezen modellnek és speciális eseteinek számos alkalmazása van. A továbbiakban azt a metrikus tér pontjaiból álló multihalmazt, amely megadja mely pontokban helyezkednek el a szerverek (azért kell multihalmazokat használnunk, mert egy pontban több szerver is lehet) a ***szerverek konfigurációjának*** nevezzük.

Az első fontos eredményeket a k-szerver feladatra Manasse McGeoch és Sleator érték el. Az általuk javasolt első eljárás a következő Egyensúly algoritmus, amelyet a továbbiakban ES-sel jelölünk. Az eljárás során a szerverek mindig különböző pontokban helyezkednek el. Az algoritmus futása során minden szerverre számon tartja, hogy az aktuális időpontig összesen mekkora távolságot tett meg. Jelölje rendre $s_1, \ldots, s_k$ a szervereket és a pontokat is, ahol a szerverek elhelyezkednek. Továbbá jelölje $D_1, \ldots, D_k$ rendre a szerverek által az adott időpontig megtett összutat. Ekkor, amennyiben egy P pontban megjelenik egy kérés, akkor az ES algoritmus azt az i szervert választja a kérés kiszolgálására, ahol a $D_i + d(s_i, P)$ érték minimális, azaz az algoritmus igyekszik az egyes szerverek által megtett utakat egyensúlyban tartani. Tehát az algoritmus számon tartja a szerverek $S = \{s_1, \ldots, s_k\}$ szerver konfigurációját és a szerverekhez rendelt távolságokat, amelyek kezdeti értékei $D_1 = \cdots = D_k = 0$. Ezt követően az algoritmus egy $I = P_1, \ldots, P_n$ pontsorozatra a következő pszeudokóddal írható le:

ES(I)

1 **for** $j \leftarrow 1$ **to** n
2 $\quad i \leftarrow \text{argmin}\{D_i + d(s_i, P_j)\}$
3 $\quad$ szolgáljuk ki a kérést az i-edik szerverrel
4 $\quad D_i \leftarrow D_i + d(s_i, P_j)$
5 $\quad s_i \leftarrow P_j$

27.1. példa. Tekintsük a kétdimenziós euklideszi teret, mint metrikus teret. A pontok (x, y) valós számpárokból állnak, két (a, b) és (c, d) pontnak a távolsága $\sqrt{(a-c)^2 + (b-d)^2}$. Legyen két szerverünk, kezdetben a $(0, 0)$ és $(1, 1)$ pontokban. Tehát a kiindulási állapotban $D_1 = D_2 = 0$, $s_1 = (0, 0)$, $s_2 = (1, 1)$. Az első kérés legyen az $(1, 4)$ pontban. Ekkor $D_1 + d((0, 0), (1, 4)) = \sqrt{17} > D_2 + d((1, 1), (1, 4)) = 3$, így a második szervert használjuk és a kérés kiszolgálása után $D_1 = 0$, $D_2 = 3$, $s_1 = (0, 0)$, $s_2 = (1, 4)$ teljesül. Legyen a második kérés $(2, 4)$, ekkor $D_1 + d((0, 0), (2, 4)) = \sqrt{20} > D_2 + d((1, 4), (2, 4)) = 3 + 1 = 4$, így ismét a második szervert használjuk, és a kérés kiszolgálása után $D_1 = 0, D_2 = 4$, $s_1 = (0, 0)$, $s_2 = (2, 4)$ teljesül. A harmadik kérés legyen ismét az $(1, 4)$ pontban, ekkor $D_1 + d((0, 0), (1, 4)) = \sqrt{17} < D_2 + d((2, 4), (1, 4)) = 4 + 1 = 5$, így az első szervert használjuk és a kérés kiszolgálása után $D_1 = \sqrt{17}, D_2 = 4$, $s_1 = (1, 4)$, $s_2 = (2, 4)$ teljesül.

Az algoritmus hatékony speciális terek esetén, miként ezt a következő állítás mutatja. A tétel bizonyítására vonatkozó hivatkozások megtalálhatók a fejezet végén szereplő megjegyzésekben.

27.1. tétel. *Amennyiben a metrikus tér $k + 1$ pontot tartalmaz, akkor az* Egyensúly *algoritmus enyhén k-versenyképes.*

Az alábbi állítás mutatja, hogy a k-szerver feladatra általában nem adható meg k-versenyképesnél jobb algoritmus.

27.2. tétel. *Nincs olyan legalább $k + 1$ pontból álló metrikus tér, ahol megadható olyan on-line algoritmus, amelynek kisebb a versenyképességi hányadosa, mint k.*

Bizonyítás. Tekintsünk egy tetszőleges legalább $k + 1$ pontból álló teret, és egy tetszőleges on-line algoritmust. Jelölje az algoritmust ONL, a pontokat, ahol kezdetben ONL szerverei állnak, $P_1, P_2, \ldots, P_k$, a térnek egy további pontját jelölje P_{k+1}. Vegyük kéréseknek egy hosszú $I = Q_1, \ldots, Q_n$ sorozatát, amelyet úgy kapunk, hogy a következő kérés mindig a $P_1, P_2, \ldots, P_{k+1}$ pontok közül abban a pontban keletkezik, ahol az ONL algoritmusnak nem tartózkodik szervere.

Vizsgáljuk elsőként az ONL(I) költséget. Mivel a Q_j pont kiszolgálása után a Q_{j+1} pont lesz szabad, ezért a Q_j pontot mindig a Q_{j+1} pontban álló szerver szolgálja ki, így a kiszolgálás költsége $d(Q_j, Q_{j+1})$. Következésképpen

$$\text{ONL}(I) = \sum_{j=1}^{n} d(Q_j, Q_{j+1}) \ ,$$

ahol Q_{n+1} azt a pontot jelöli, ahol az $(n + 1)$-edik kérés lenne, azaz azt a pontot, amelyről kiszolgáltuk az n-edik kérést.

Most vizsgáljuk az $\mathrm{OPT}(I)$ költséget. Az optimális off-line algoritmus meghatározása helyett definiálunk k darab off-line algoritmust, és ezek költségeinek az átlagát használjuk. Mivel mindegyik algoritmus költsége legalább akkora, mint a minimális költség, ezért a költségek átlaga is felső korlátja lesz az optimális költségnek.

Definiáljunk k darab off-line algoritmust, jelölje őket $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$. Tegyük fel, hogy a kiindulási állapotban az OFF_j algoritmus szerverei a $P_1, P_2, \ldots, P_{k+1}$ pontok közül a P_j pontot szabadon hagyják, és a halmaz további pontjainak mindegyikén egy szerver helyezkedik el. Ez a kiindulási állapot elérhető egy C_j konstans extra költség felhasználásával.

A kérések kiszolgálása a következőképpen történik. Ha a Q_i ponton van az OFF_j algoritmusnak szervere, akkor nem mozgat egyetlen szervert sem, ha nincs, akkor a Q_{i-1} ponton levő szervert használja. Az algoritmusok jól definiáltak, hiszen ha nincs szerver a Q_i ponton, akkor ezen pont kivételével a $P_1, P_2, \ldots, P_{k+1}$ pontok mindegyikén, így Q_{i-1}-en is van szerver. Továbbá a $Q_1 = P_{k+1}$ ponton az OFF_j algoritmusok mindegyikének áll szervere a kezdeti konfigurációban.

Vegyük észre, hogy az $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$ algoritmusok szerverei rendre különböző konfigurációkban vannak. Kezdetben ez az állítás a definíció alapján igaz. Utána a tulajdonság azért marad fenn, mert amely algoritmusok nem mozdítanak szervert a kérés kiszolgálására, azoknak a szerver konfigurációja nem változik. Amely algoritmusok mozgatnak szervert, azok a Q_{i-1} pontról viszik el a szervert, amely pont az előző kérés volt, így ott minden algoritmusnak van szervere. Következésképp ezen algoritmusok szerver konfigurációja nem állítódhat azonos pozícióba olyan algoritmus szerver konfigurációjával, amely nem mozdított szervert. Másrészt, ha több algoritmus is mozgatná Q_{i-1}-ről Q_i-be a szervert, azok szerver konfigurációja se válhat azonossá, hisz a mozgatást megelőzőleg különböző volt.

Következésképp egy Q_i kérés esetén minden OFF_j algoritmusra más a szerver konfiguráció. Továbbá minden konfigurációnak tartalmaznia kell Q_{i-1}-et, tehát pontosan egy olyan OFF_j algoritmus van, amelynek nincsen szervere a Q_i ponton. Tehát a Q_i kérés kiszolgálásának a költsége az OFF_j algoritmusok egyikénél $d(Q_{i-1}, Q_i)$, a többi algoritmus esetén pedig 0.

Következésképpen

$$\sum_{j=1}^{k} \mathrm{OFF}_j(I) = C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \ ,$$

ahol $C = \sum_{j=1}^{k} C_j$ egy a bemeneti sorozattól független konstans, amely az off-line algoritmusok kezdeti konfigurációinak beállításának költsége.

Másrészt az off-line optimális algoritmus költsége nem lehet nagyobb semelyik off-line algoritmus költségénél sem, így $k \cdot \mathrm{OPT}(I) \leq \sum_{j=1}^{k} \mathrm{OFF}_j(I)$. Következésképpen

$$k \cdot \mathrm{OPT}(I) \leq C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \leq C + \mathrm{ONL}(I) \ ,$$

amely egyenlőtlenségből következik, hogy az ONL algoritmus versenyképességi hányadosa nem lehet kisebb, mint k, hiszen a bemeneti sorozat hosszúságának növelésével a $\mathrm{OPT}(I)$ érték tetszőlegesen nagy lehet. ■

A kérdés felvetése nagy érdeklődést keltett, az elkövetkező néhány évben számos eredmény született. Az általános esetre konstans-versenyképes algoritmust ($O(2^k)$-versenyképes algoritmust) elsőként Fiat, Rabani és Ravid fejlesztettek ki. Ezt követően hosszú időn át nem sikerült lényegesen csökkenteni a felső és az alsó korlát közötti rést. Az áttörést Koutsoupias és Papadimitriou eredménye hozta meg, sikerült a feladat megoldására a Chrobak és Larmore által javasolt munkafüggvényen alapuló algoritmust elemezniük és igazolniuk, hogy az algoritmus $(2k-1)$-versenyképes. Nem sikerült meghatározniuk az algoritmus pontos versenyképességi hányadosát, bár általánosan elfogadott sejtés, hogy az algoritmus valójában k-versenyképes. Ezen versenyképességi hányados pontos meghatározása, illetve egy k-versenyképes algoritmus kifejlesztése azóta is az on-line algoritmusok elméletének legismertebb és sokak által legfontosabbnak tartott nyílt problémája. Az alábbiakban ismertetjük a munkafüggvény algoritmust alapötletét.

Legyen A_0 az on-line szerverek kezdeti konfigurációja. Ekkor a t-edik kérés utáni, X multihalmazra vonatkozó ***munkafüggvény*** $w_t(X)$ az a minimális költség, amellyel kiszolgálható az első t kérés az A_0 konfigurációból kiindulva úgy, hogy a szerverek az X konfigurációba kerüljenek a kiszolgálás végén. A munkafüggvény értékeinek meghatározása egy off-line optimalizálási feladat, amely dinamikus programozási módszerrel megoldható. A MUNKAFÜGGVÉNY algoritmus a munkafüggvényt használja. Legyen A_{t-1} a szervereknek a konfigurációja közvetlenül a t-edik kérés érkezése előtt. Ekkor a MUNKAFÜGGVÉNY algoritmus azzal az s szerverrel szolgálja ki az R_t pontban megjelent kérést, amelynek a P helyére a $w_{t-1}(A_{t-1} \setminus \{P\} \cup \{R_t\}) + d(P, R_t)$ érték a minimális.

Az algoritmusra a bemenet pontoknak egy $I = P_1, \ldots, P_k$ sorozata, amely a kérések listája, és a szerverek kezdeti S konfigurációja, ami szintén pontoknak egy halmaza, az algoritmus azt munkafüggvényt használja, amelyre a kiindulási A_0 halmaz S kezdeti értéke. Az algoritmust a következő pszeudokóddal adhatjuk meg.

MUNKAFÜGGVÉNY(I, S)

```
1  for j ← 1 to n
2      do i ← argmin_s w_{j-1}(S \ {s} ∪ {P_j}) + d(s, P_j)
3         a kérést az i-edik szerverrel szolgáljuk ki
4         s_i ← P_j
```

27.2. példa. Tekintsük azt a metrikus teret, amelyben három pont van, A, B és C, a távolságok $d(A,B) = 1$, $d(B,C) = 2$, $d(A,C) = 3$. A kezdeti szerver konfiguráció legyen $\{A, B\}$. Ekkor a kezdeti munkafüggvények $w_0(\{A,A\}) = 1$, $w_0(\{A,B\}) = 0$, $w_0(\{A,C\}) = 2$, $w_0(\{B,B\}) = 1$, $w_0(\{B,C\}) = 3$, $w_0(\{C,C\}) = 4$, Legyen az első kérés a C pontban. Ekkor $w_0(\{A,B\} \setminus \{A\} \cup \{C\}) + d(A,C) = 3 + 3 = 6$ és $w_0(\{A,B\} \setminus \{B\} \cup \{C\}) + d(B,C) = 2 + 2 = 4$, így a MUNKAFÜGGVÉNY a B pontban levő szervert küldi a kérés kiszolgálására.

Az algoritmusra teljesül a következő állítás.

27.3. tétel. *A* MUNKAFÜGGVÉNY *algoritmus enyhén* $(2k-1)$*-versenyképes.*

Az általános feladat vizsgálata mellett a feladat speciális eseteit számos dolgozatban vizsgálták. Amennyiben bármely két pont távolsága 1, akkor a lapozási feladat on-line változatához jutunk, amely a számítógépek memóriakezelését modellezi. Egy másik vizsgált speciális tér az egyenes. Az egyenes pontjait a valós számoknak feleltetjük meg, és két pontnak,

a-nak és b-nek a távolsága $|a-b|$. Erre az esetre, ahol a metrikus tér egy egyenes, Chrobak és Larmore kifejlesztett egy k-versenyképes algoritmust, amelyet DUPLA-LEFEDŐ algoritmusnak nevezünk. Az algoritmus a P kérést a P-hez legközelebb eső s szerverrel szolgálja ki, és amennyiben vannak szerverek P-nek az s-sel átellenes oldalán is, akkor azon szerverek közül a P-hez legközelebbit $d(s,P)$ egységnyit mozdítja P felé. A továbbiakban a DUPLA-LEFEDŐ algoritmust DL-lel jelöljük. Az algoritmusra a bemenet pontok (valós számok) egy $I = P_1, \ldots, P_k$ sorozata, amely a kérések listája, és a szerverek kezdeti $S = (s_1, \ldots, s_k)$ konfigurációja, ami szintén pontok (valós számok) egy halmaza. Az algoritmust a következő pszeudokóddal adhatjuk meg.

DL(I, S)

```
1  for j ← 1 to n
2      do i ← argmin_l d(P_j, s_l)
3  if s_i = min_l s_l vagy s_i = max_l s_l
4    then a kérést az i-edik szerverrel szolgáljuk ki
5         s_i ← P_j
6  else if s_i ≤ P_j
7         then m ← argmin_{l:s_l>P_j} d(s_l, P_j)
8              a kérést az i-edik szerverrel szolgáljuk ki
9              s_m ← s_m − d(s_i, P_j)
10             s_i ← P_j
11 else if s_i ≥ P_j
12        then n ← argmin_{l:s_l<P_j} d(s_l, P_j)
13             a kérést az i-edik szerverrel szolgáljuk ki
14             s_n ← s_n + d(s_i, P_j)
15             s_i ← P_j
```

27.3. példa. Tegyük fel, hogy három szerverünk van $(s_1,\ s_2,\ s_3)$, amelyek az egyenes 0, 1, 2 pontjaiban helyezkednek el. Amennyiben a következő kérés a 4 pontban jelenik meg, akkor DL a legközelebbi s_3 szervert küldi a kérés kiszolgálására. A többi szerver helye nem változik, a költség 2 és a kérés kiszolgálása után a szerverek a 0, 1, 4 pontokban lesznek. Amennyiben ezt követően a következő kérés a 2 pontban jelenik meg, akkor DL a legközelebbi s_2 szervert küldi a kérés kiszolgálására, de mivel a kérés másik oldalán is van szerver, ezért s_3 is megtesz egy egységnyi utat a kérés felé, így a költség 2 és a kérés kiszolgálása után a szerverek a 0, 2, 3 pontokban lesznek.

A DL algoritmusra teljesül a következő állítás, amelynek a bizonyításához az on-line algoritmusok elemzése során gyakran használt potenciálfüggvény technikát alkalmazzuk.

27.4. tétel. *Ha a metrikus tér egy egyenes, akkor a* DUPLA-LEFEDŐ *algoritmus enyhén k-versenyképes.*

Bizonyítás. Vegyük egy tetszőleges sorozatát a kéréseknek, jelölje ezt az bemenetet I. Az eljárás elemzése során feltételezzük, hogy párhuzamosan fut a DL algoritmus és egy optimális off-line algoritmus. Szintén feltesszük, hogy minden kérést elsőként az off-line algoritmus szolgál ki, utána pedig az on-line algoritmus. Az on-line algoritmus szervereit és egyben a szerverek pozícióit (amelyek valós számok az egyenesen) $s_1, \ldots, s_k$ jelöli, az optimális off-line algoritmus szervereit és egyben a szerverek pozícióit $x_1, \ldots, x_k$ jelöli. Mivel

a szerverek rendszeres átjelölésével ez elérhető, feltételezzük, hogy $s_1 \leq s_2 \leq \cdots \leq s_k$ és $x_1 \leq x_2 \leq \cdots \leq x_k$ mindig teljesül.

A tétel állítását a potenciálfüggvény technikájával igazoljuk. A potenciálfüggvény a szerverek aktuális pozíciójához rendel egy értéket, az on-line és az off-line költségeket a potenciálfüggvény változásainak alapján hasonlítjuk össze. Legyen a potenciálfüggvény

$$\Phi = k \sum_{i=1}^{k} |x_i - s_i| + \sum_{i<j} (s_j - s_i) \ .$$

Az alábbiakban igazoljuk, hogy a potenciálfüggvényre teljesülnek a következő állítások.

- Amíg OPT szolgálja ki a kérést, addig a potenciálfüggvény növekedése legfeljebb k-szor az OPT szerverei által megtett távolság.
- Amíg DL szolgálja ki a kérést, addig Φ legalább annyival csökken, mint amennyi a kérés kiszolgálásának költsége.

Amennyiben a fenti tulajdonságok teljesülnek, akkor a tétel állítása következik, hiszen ebben az esetben adódik, hogy $\Phi_v - \Phi_0 \leq k \cdot \mathrm{OPT}(I) - \mathrm{DL}(I)$, ahol Φ_v és Φ_0 a potenciálfüggvény kezdeti és végső értékei. Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $\mathrm{DL}(I) \leq k\mathrm{OPT}(I) + \Phi_0$, azaz azt kapjuk, hogy a DL algoritmus enyhén k-versenyképes.

Most igazoljuk a potenciálfüggvény tulajdonságait.

Elsőként vizsgáljuk azt az esetet, amikor OPT valamely szervere d távolságot mozog. Ekkor a potenciálfüggvényben szereplő első részben az összegnek egyetlen tagja változhat, annak az értéke legfeljebb d-vel növekszik, így ez a rész legfeljebb kd-vel növekszik. A második rész nem változik, tehát az első tulajdonsága a potenciálfüggvénynek valóban fennáll.

Most vizsgáljuk DL szervereit. Legyen P a kérés, amelyet ki kell szolgálni. Mivel elsőként OPT szolgálta ki a kérést, ezért $x_j = P$ valamely szerverre. Most különböztessünk meg két esetet DL szervereinek elhelyezkedésétől függően.

Elsőként tegyük fel, hogy minden szerver P-nek ugyanarra az oldalára esik. Feltehetjük, hogy minden szerver pozíciója nagyobb P-nél, a másik eset teljesen hasonló. Ekkor s_1 a legközelebbi szerver P-hez és DL s_1-et küldi P-be, más szervert nem mozgat. Tehát DL költsége $d(s_1, P)$. A potenciálfüggvényben szereplő első összegben csak az $|x_1 - s_1|$ tag változik és ez csökken $d(s_1, P)$ egységgel, tehát az első rész csökken $kd(s_1, P)$ egységgel. A második tag növekszik $(k-1)d(s_1, P)$ egységgel, így Φ értéke csökken $d(s_1, P)$ egységgel.

Most tekintsük a másik esetet. Ekkor P-nek mindkét oldalára esik szerver, legyenek ezek a szerverek s_i és s_{i+1}. Tegyük fel, hogy s_i esik közelebb P-hez, a másik eset teljesen hasonló. Tehát DL költsége $2d(s_i, P)$. Vizsgáljuk a potenciálfüggvény első részének változásait. Az i-edik és az $i+1$-edik tag változik. Az egyik tag növekszik, a másik csökken $d(s_i, P)$ egységgel, tehát az első rész összességében nem változik. A Φ függvény második részének változása

$$d(s_i, P)\big(-(k-i) + (i-1) - (i) + (k-(i+1))\big) = -2d(s_i, P) \ .$$

Tehát ebben az esetben is fennáll a potenciálfüggvény második tulajdonsága. Mivel több eset nem lehetséges, ezért igazoltuk a potenciálfüggvény tulajdonságainak fennállását, amivel a tétel állítását is bebizonyítottuk. ■

Gyakorlatok

27.2-1. Legyen (M, d) egy metrikus tér. Igazoljuk, hogy (M, q) is egy metrikus tér, ahol $q(x, y) = \min\{1, d(x, y)\}$.

27.2-2. Vegyük azt az algoritmust, amely minden kérést a hozzá legközelebbi szerverrel szolgál ki. Igazoljuk, hogy az algoritmus nem konstans versenyképes, ha a metrikus tér egy egyenes.

27.2-3. Igazoljuk, hogy tetszőleges X és Z k-elemű multihalmazokra, továbbá minden t-re teljesül a $w_t(Z) \leq w_t(X) + d(X, Z)$ egyenlőtlenség, ahol $d(X, Z)$ az X és Z multihalmazok közötti minimális párosítás költsége, azaz a minimális költség, amellyel a szerverek az X konfigurációból a Z konfigurációba átvihetők.

27.2-4. Vegyük az egyenest, mint metrikus teret. Legyenek az on-line algoritmus szerverei a 2, 4, 5, 7 pontokban, az off-line algoritmus szerverei az 1, 3, 6, 9 pontokban. Határozzuk meg a 27.4. tétel bizonyításában használt potenciálfüggvény értékét. Hogy változik ez a függvényérték, ha a 7 pontban levő on-line szerver a 8 pontba mozdul?

27.3. Számítógépes hálózatokhoz kapcsolódó modellek

A számítógépes hálózatok elmélete az alkalmazások rendkívüli fontosságából adódóan az informatika egyik legjelentősebb kutatási területévé vált. A hálózatok megtervezésénél és a hálózatok működése során számos optimalizálási feladat merül fel, amely feladatok többsége lényegében on-line, hiszen sem a forgalom, sem pedig a hálózati topológia esetleges változásai nem láthatók előre. A 90-es évek végén az on-line algoritmusok területén kutató szakemberek on-line matematikai modelleket dolgoztak ki a számítógépes hálózatok témakörébe tartozó on-line feladatokra. Ebben a részben ezzel a területtel fogunk foglalkozni, három kérdést vizsgálunk meg, bemutatjuk az alapvető eredményeket. Elsőként a nyugtázás feladatára kidolgozott modellt ismertetjük, majd a lapletöltés feladatával, végül az on-line forgalomirányítás területével foglalkozunk.

27.3.1. A nyugtázási feladat modellje

A számítógépes hálózatok kommunikációja során a küldő a címzettnek adatcsomagokat küld. Amennyiben a kommunikációs csatorna nem teljesen megbízható (például vezeték nélküli), akkor lényeges a csomagok megérkezéséről a címzettnek nyugtát küldeni, így lehetővé tehető az elveszett adatcsomagok újraküldése. A nyugtázási feladat során felmerülő kérdések egyike, hogy mikor nyugtázzuk a csomag megérkezését. Egy nyugtával több csomag megérkezését igazolhatjuk. Minden csomagra külön nyugta küldése nagy mértékben növelné a kommunikációs csatornák telítettségét. Másrészt, a nyugta küldésével hosszan várakozni sem lehet, hiszen az igazolás késedelme a csomag újraküldéséhez vezethet, ami ismét a csatorna túltelítettségét eredményezheti. A nyugtaküldések idejének megállapítására az első optimalizálási modellt Dooly, Goldman és Scott fejlesztették ki 1998-ban. Az alábbiakban ismertetjük az általuk kifejlesztett modell lényegét és az alapvető eredményeket.

A nyugtázási feladat matematikai modelljében a bemenetet a csomagok $a_1, \ldots, a_n$ érkezési idejei adják. Az algoritmusnak meg kell határoznia, mikor küld nyugtákat, ezeket az

időpontokat $t_1, \ldots, t_k$ jelöli. A modellben a költségfüggvény:

$$k + \sum_{j=1}^{k} \nu_j \ ,$$

ahol k a nyugták száma és $\nu_j = \sum_{t_{j-1} < a_i \leq t_j} (t_j - a_i)$ a j-edik nyugta által összegyűjtött teljes késedelem. A feladat on-line, azaz egy adott t időpontban csak a t-ig megérkezett csomagok érkezési idejét ismerjük és nincs semmi információnk a további csomagokról. Az a_i csomag érkezése után σ_i jelöli a nyugtázatlan csomagok halmazát.

A feladat megoldására az ébresztő beállításán alapuló algoritmusokat dolgoztak ki. Egy ***ébresztő algoritmus*** a következőképpen működik. Az a_j csomag érkezésekor beállítunk egy ébresztőt valamilyen $a_j + e_j$ időpontra. Ha az $a_j + e_j$ időpontig nem érkezik új csomag, akkor az $a_j + e_j$ időpontban nyugtát küldünk, egyébként a következő csomag érkezésekor az a_{j+1} időpontban átállítjuk az ébresztőt egy $a_{j+1} + e_{j+1}$ időpontra. Az alábbiakban egy ébresztő algoritmust elemzünk részletesebben, azt az algoritmust, amely úgy állítja be az ébresztőt, hogy az első nyugtázatlan csomagtól legyen a teljes késedelem költsége 1. Ezt az algoritmust ÉBRESZT algoritmusnak nevezzük. A fenti szabály azt jelenti, hogy az általános definícióban az e_j érték a következő egyenlet megoldása:

$$1 = |\sigma_j| e_j + \sum_{a_i \in \sigma_j} (a_j - a_i) \ .$$

27.4. példa. Tekintsük a következő példát. Az első csomag a 0 időpontban érkezik, azaz $a_1 = 0$, ekkor ÉBRESZT beállítja az ébresztőt az $e_1 = 1$ értékkel. Legyen a következő csomag érkezési ideje $a_2 = 1/2$, ekkor még nem járt le az ébresztő, így nem küldtünk nyugtát, tehát átállítjuk az ébresztőt az $e_2 = (1 - 1/2)/2 = 1/4$ értékkel az $1/2 + 1/4$ időpontra. Legyen a következő csomag érkezési ideje $a_3 = 5/8$. Ekkor még nem járt le az ébresztő és nem küldtünk nyugtát, így újra állítjuk az ébresztőt az $e_3 = (1 - 5/8 - 1/8)/3 = 1/12$ értékkel az $5/8 + 1/12$ időpontra. Legyen a következő csomag érkezési ideje $a_4 = 1$, mivel az $5/8 + 1/12$ időpontig nem érkezett újabb csomag, ezért abban az időpontban nyugtát küldtünk, és most az ébresztőt az $e_4 = 1$ értékkel a 2 időpontra állítjuk be.

Az algoritmus versenyképességére teljesül a következő állítás.

27.5. tétel. *Az* ÉBRESZT *algoritmus 2-versenyképes.*

Bizonyítás. Tegyük fel, hogy az ÉBRESZT algoritmus k darab nyugtát küld. Ezek a nyugták k darab intervallumot határoznak meg. Az algoritmus költsége legfeljebb $2k$, hiszen k a nyugtákból keletkező költség, és az algoritmus úgy állítja be az ébresztőt, hogy a teljes késedelemből adódó költség minden nyugtára pontosan 1 legyen.

Legyen k^* az optimális off-line algoritmus által küldött nyugták száma. Ha $k^* \geq k$, akkor $\mathrm{OPT}(I) \geq k = \text{ÉBRESZT}(I)/2$ nyilvánvalóan teljesül és adódik, hogy az algoritmus valóban 2-versenyképes. Amennyiben $k^* < k$, akkor az ÉBRESZT algoritmus nyugtái által meghatározott k intervallum közül legalább $k - k^*$ intervallumban OPT-nak nincs nyugtája ez legalább $k - k^*$ késedelemből származó költséget jelent OPT számára, így ismét $\mathrm{OPT}(I) \geq k$ adódik, amely egyenlőtlenségből következik, hogy az algoritmus 2-versenyképes. ■

A fentiekben ismertetett ÉBRESZT algoritmus a lehetséges legjobb algoritmus a versenyképességi analízis szempontjából, hiszen miként a következő állítás mutatja, nem létezik olyan algoritmus, amelynek kisebb lenne a versenyképességi hányadosa.

27.6. tétel. *Nem létezik olyan on-line algoritmus az on-line nyugtázási feladatra, amelynek kisebb a versenyképességi hányadosa, mint 2.*

Bizonyítás. Vegyünk egy tetszőleges on-line algoritmust, jelölje ONL. Tekintsük a következő bemenetet. Vegyük csomagok egy hosszú sorozatát, amelyet úgy kapunk, hogy minden esetben, amikor ONL egy nyugtát küld, azonnal egy új csomag érkezik. Ekkor egy $2n$ csomagból álló sorozat esetén az on-line algoritmus költsége $\mathrm{ONL}(I_{2n}) = 2n + t_{2n}$, hiszen a nyugtákból adódó költsége $2n$, és az i-edik nyugtánál fellépő késedelem $t_i - t_{i-1}$, ahol a $t_0 = 0$ értéket használjuk.

Vizsgáljuk meg a következő két algoritmust. ODD a páros sorszámú csomagok után küld nyugtát, EV pedig a páratlan sorszámú csomagok és közvetlenül az utolsó, $2n$-edik csomag után.

Ekkor ezen algoritmusok költségei

$$\mathrm{EV}(I_{2n}) = n + \sum_{i=0}^{n-1}(t_{2i+1} - t_{2i}) + 1 \ ,$$

és

$$\mathrm{ODD} = n + \sum_{i=1}^{n}(t_{2i} - t_{2i-1}) \ .$$

Következésképpen $\mathrm{EV}(I_{2n}) + \mathrm{ODD}(I_{2n}) = \mathrm{ONL}(I_{2n}) + 1$. Másrészt az optimális off-line algoritmusra $\mathrm{OPT}(I_{2n}) \leq \min\{\mathrm{EV}(I_{2n}), \mathrm{ODD}(I_{2n})\}$, így azt kapjuk, hogy $\mathrm{ONL}(I_{2n})/\mathrm{OPT}(I_{2n}) \geq 2 - 1/\mathrm{OPT}(I_{2n})$. Ezen egyenlőtlenség alapján adódik, hogy ONL nem lehet jobb, mint 2-versenyképes, hiszen a csomagok elegendően hosszú sorozatát véve az $\mathrm{OPT}(I_{2n})$ érték tetszőlegesen nagy lehet. ■

27.3.2. A lapletöltési feladat

A 11.2. alfejezetben tárgyalt lapcserélési feladat általánosítása a lapletöltési feladat. A világhálón a böngészők is használnak egy memóriát, amelyben a letöltött lapokat tárolják, hogy amennyiben a felhasználó egy oldalt rövid időn belül többször meg akar nézni, ne kelljen minden alkalommal letölteni. Amennyiben a memória megtelik és az új lap nem helyezhető el benne, akkor valamilyen stratégia alapján ki kell rakni bizonyos lapokat a memóriából. A lapletöltési feladat a megfelelő cserélési stratégiák megkeresése. A különbség a lapozás feladatához képest, hogy nem minden lap mérete egyforma, továbbá az egyes lapok letöltési költsége is különböző. Tehát a modellt a következőképpen fogalmazhatjuk meg.

Adott egy k méretű memória. A bemenet a letöltendő lapok sorozata. Minden p lapnak van egy $s(p)$ ***mérete*** és egy $c(p)$ ***letöltési költsége.*** A letöltendő lapot a memóriába kell tennünk. Ha nem fér el, akkor fel kell szabadítani elegendő helyet a memóriában szereplő lapok kihelyezésével. Ha a kért lap a memóriában van, akkor a letöltés költsége 0, egyébként $c(p)$. Célunk az összköltség minimalizálása. A feladat on-line, ami azt jelenti, hogy a döntéseket

(telített memória esetén mely lapokat dobjuk ki) csak az adott kérésig megtörtént kérések ismerete alapján kell meghozni, a további kérésekre vonatkozó információk nélkül. A továbbiakban feltesszük, hogy mind a memória mérete, mind pedig a lapok méretei pozitív egész számok.

A feladatnak és speciális eseteinek megoldására több eljárást is javasoltak. Az alábbiakban a Young által kifejlesztett enyhén k-versenyképes HÁZIÚR algoritmust és annak elemzését ismertetjük.

Az algoritmus minden lapra, amely az algoritmus memóriájában van, számon tart egy $0 \leq cr(f) \leq c(f)$ értéket. Az algoritmus futása során a HÁZIÚR algoritmus memóriájában aktuálisan szereplő lapok halmazát HA jelöli. Amennyiben egy g fájlt kell letöltenünk, a következő lépéseket hajtjuk végre:

HÁZIÚR(HA, g)

1 **if** g nincs a memóriában
2 **then while** nincs elég hely
3 $\Delta \leftarrow \min_{f \in HA} cr(f)/s(f)$
4 minden $f \in HA$-ra $cr(f) \leftarrow cr(f) - \Delta \cdot s(f)$
5 tegyünk ki olyan lapokat, amelyekre $cr(f) = 0$
6 tegyük be g-t a HA memóriába, $cr(g) \leftarrow c(g)$
7 **else** állítsuk át $cr(g)$ értékét valamelyik $cr(g)$ és $c(g)$ közötti értékre

27.5. példa. Tegyük fel, hogy $k = 10$ és az adott pillanatban a HA memória három lapot tartalmaz: g_1-re $s(g_1) = 2, cr(g_1) = 1$, g_2-re $s(g_2) = 4, cr(g_2) = 3$, g_3-ra $s(g_3) = 3, cr(g_3) = 3$. Legyen a következő letöltendő lap g_4, amelyre $s(g_4) = 4, c(g_4) = 4$. Ekkor ez a lap nem fér el a HA memóriában, ki kell tennünk lapokat. A HÁZIÚR algoritmus által számolt érték $\Delta = 1/2$ és a megváltoztatott cr értékek: $cr(g_1) = 0, cr(g_2) = 1, cr(g_3) = 3/2$, így g_1-et eltávolítjuk a HA memóriából. Ekkor a g_4 lap még mindig nem fér el a HA memóriában. Az új Δ érték $\Delta = 1/4$ és a megváltoztatott cr értékek: $cr(g_2) = 0, cr(g_3) = 3/4$, így a g_2 lapot is eltávolítjuk a HA memóriából. Ekkor már van hely g_4 számára, elhelyezzük a memóriában a $cr(g_4) = 4$ értékkel.

Az algoritmus enyhén k-versenyképes, de ennél erősebb állítás is igaz. A lapletöltési feladatra egy ALG on-line algoritmust enyhén (C, k, h)-versenyképesnek nevezünk, ha van olyan B konstans, hogy $\mathrm{ALG}_k(I) \leq C \cdot \mathrm{OPT}_h(I) + B$ minden bemenetre teljesül, ahol $\mathrm{ALG}_k(I)$ az algoritmus által elvégzett összes letöltés költsége k méretű memória mellett, $\mathrm{OPT}_h(I)$ pedig a minimális elérhető teljes letöltési összeg h méretű memória mellett. A HÁZIÚR algoritmusra teljesül a következő állítás.

27.7. tétel. *Ha $h \leq k$, akkor a* HÁZIÚR *algoritmus $(k/(k-h+1), k, h)$-versenyképes.*

Bizonyítás. Tekintsük kérések egy tetszőleges sorozatát, jelölje ezt a bemenetet I. Miként a 27.4. tétel bizonyítása során, itt is a potenciálfüggvény technikát alkalmazzuk. Párhuzamosan hajtjuk végre az OPT és a HÁZIÚR algoritmusokat, minden kérésnél először az OPT és utána a HÁZIÚR algoritmus lépését hajtjuk végre.

Jelölje az optimális off-line algoritmusnak a memóriájában aktuálisan szereplő lapjainak halmazát OPT. Tekintsük a következő potenciálfüggvényt.

$$\Phi = (h-1)\sum_{f\in HA} cr(f) + k\sum_{f\in OPT}(c(f)-cr(f))\ .$$

Vizsgáljuk a potenciálfüggvény változásait egy g lap letöltése során.

- OPT elhelyez egy g lapot a memóriájában.

  Ekkor OPT költsége $c(g)$, a potenciálfüggvénynek csak a második része változhat, mivel $cr(g) \geq 0$, ezért a potenciálfüggvény növekedése legfeljebb $k \cdot c(g)$.

- Háziúr csökkenti minden $f \in HA$-ra a $cr(f)$ értéket.

  Ekkor minden $f \in HA$ esetén a $cr(f)$ érték csökkenése $\Delta \cdot s(f)$, így összességében Φ a

  $$\Delta((h-1)s(HA) - ks(OPT \cap HA))$$

  értékkel csökken, ahol $s(HA)$ és $s(OPT \cap HA)$ rendre a HA illetve az $OPT \cap HA$ halmazokban levő lapoknak a méreteinek az összege. Abban az időpontban, mikor ez a lépés bekövetkezik, OPT már elhelyezte a g lapot OPT-ban, de a lap még nincs a HA halmazban. Következésképp $s(OPT \cap HA) \leq h - s(g)$. Másrészt ez a lépés azért hajtódik végre, mert nem fért el a g lap a HA memóriában, tehát $s(HA) > k - s(g)$, és így, mivel a lapok méretei egészek, ezért $s(HA) \geq k - s(g) + 1$. Következésképpen azt kapjuk, hogy a Φ potenciálfüggvény csökkenése legalább

  $$\Delta\big((h-1)(k-s(g)+1) - k(h-s(g))\big)\ .$$

  Mivel $s(g) \geq 1$ és $k \geq h$, ezért a fenti érték legalább $\Delta((h-1)(k-1+1) - k(h-1)) = 0$.

- Háziúr kirak egy f lapot a HA memóriából.

  Mivel Háziúr csak akkor rak ki egy f lapot a memóriából, ha arra $cr(f) = 0$, ezért ebben a részben nem változik Φ.

- Háziúr elhelyezi a g lapot a HA memóriában és beállítja a $cr(g) = c(g)$ értéket.

  Ekkor Háziúr költsége $c(g)$. Másrészt g nem volt eddig benne a HA memóriában így $cr(g) = 0$ teljesült. Továbbá elsőként OPT helyezte el a lapot, így $g \in OPT$ teljesül. Következésképpen a Φ függvény értékének csökkenése ebben a lépésben $-(h-1)c(g) + kc(g) = (k-h+1)c(g)$.

- Háziúr átállítja a $g \in HA$ lapra a $cr(g)$ értéket, valamely $cr(g)$ és $c(g)$ közötti értékre.

  Ebben az esetben is fennáll $g \in OPT$, hisz OPT már hamarabb elhelyezte g-t a memóriájába. Mivel $cr(g)$ nem csökkenhet, és $k > h-1$, ezért ebben az esetben Φ nem növekedhet.

Végignéztük az algoritmusok lehetséges lépéseit és a Φ függvény következő tulajdonságai adódtak.

- Ha OPT elhelyez egy lapot a memóriájában, akkor a potenciálfüggvény növekedése legfeljebb k-szor az OPT algoritmusnál fellépő költség.
- Ha Háziúr elhelyez egy lapot a memóriájában, akkor a Φ függvény $(k-h+1)$-szer annyival csökken, mint amennyi az algoritmusnál fellépő költség.

- A többi esetben Φ nem növekszik.

A fentiek alapján azt kapjuk, hogy $\Phi_v - \Phi_0 \leq k \cdot \text{OPT}_h(I) - (k-h+1) \cdot \text{HÁZIÚR}_k(I)$, ahol Φ_0 és Φ_v a potenciálfüggvény kezdeti és végső értékei. Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $(k-h+1) \cdot \text{HÁZIÚR}_k(I) \leq k \cdot \text{OPT}_h(I) + \Phi_0$, azaz azt kapjuk, hogy a HÁZIÚR algoritmus enyhén $(k/(k-h+1), k, h)$-versenyképes. ■

27.3.3. Forgalomirányítási algoritmusok

A számítógépes hálózatok esetén az egyes kommunikációs csatornák túltelítettsége a számítógépes forgalom nagymértékű lassulásához és információk elvesztéséhez vezethet. Ezért a számítógépes hálózatok elméletének egyik legalapvetőbb feladata a forgalom szabályozása. A forgalom szabályozásának témakörébe tartozik a forgalomirányítás is, melynek során azt kell meghatároznunk, hogy az egyes üzenetek az üzenet küldőjétől a címzetthez milyen útvonalon jussanak el, mely közbenső állomásokon keresztül. A számítógépes hálózatok esetén nincs teljes információnk az összes jelenlegi és jövőbeli üzenetről, amely a rendszerbe kerül, így valóban egy on-line feladatról van szó. Az alábbiakban a forgalomirányítás feladatának két on-line modelljét mutatjuk be, amely modellek nem csak a számítógépes hálózatok modellezésére alkalmasak, hanem általánosabb forgalomirányítási feladatok tanulmányozására is.

A matematikai modell

A hálózatot egy gráf ábrázolja és minden e élnek van egy $u(e)$ maximális felhasználható sávszélessége, az élek számát m-el jelöljük. A feladat az, hogy sorban kérések érkeznek, a j-edik kérés egy $(s_j, t_j, r_j, d_j, b_j)$ vektor, a feladat pedig az s_j pontból a t_j pontba egy kiválasztott úton r_j sávszélességet lefoglalni d_j időtartamra a kérés megjelenésétől kezdve. Amennyiben a kérést elfogadjuk, a nyereség b_j. A továbbiakban mindig feltesszük, hogy $d_j = \infty$ minden j kérés esetén. A feladat on-line, ami azt jelenti, hogy a kérés időpontjában nem tudunk semmit a későbbi kérésekről. Két különböző modellt ismertetünk.

Terhelést kiegyensúlyozó modell. Ebben a modellben minden kérést el kell fogadni és a célfüggvény az élek túltelítettségének, ami a hozzájuk rendelt teljes sávszélességnek és a megengedett maximális felhasználható sávszélességnek a hányadosa, a maximumának a minimalizálása.

Nyereség maximalizáló modell. Ebben a modellben visszautasíthatunk kéréseket, a teljes sávszélesség egyetlen élen sem lehet nagyobb, mint a megengedett maximális sávszélesség, a cél az elfogadott kérésekhez rendelt nyereségek összegének maximalizálása. Mivel a továbbiakban ezzel a modellel foglalkozunk ezért fontosnak tartjuk kiemelni, hogy az eddigi részekkel ellentétben itt maximalizálási feladatról van szó, így a versenyképesség fogalmát a maximalizálási feladatokra megadott formában fogjuk használni.

Az alábbiakban ismertetjük az exponenciális algoritmust. Az algoritmus pontos megfogalmazásához és elemzéséhez szükségünk lesz az alábbi jelölésekre. Minden elfogadott i kérésre a kéréshez lefoglalt utat P_i jelöli. Legyen A az algoritmus által elfogadott kérések halmaza. Ekkor az $l_e(j) = (\sum_{i \in A, i<j, e \in P_i} r_i)/u(e)$ érték azt adja meg, hogy a j-edik kérés előtt az e-n keresztül lefoglalt sávszélesség a megengedett sávszélességnek mekkora része.

Az exponenciális algoritmusok alapötlete, hogy minden élhez egy $l_e(j)$-ben exponenciális költséget rendelnek, és minimális költségű utat választanak. Az alábbiakban részletesen

ismertetjük és elemezzük az exponenciális algoritmust a nyereségmaximalizáló modellre. Ehhez legyen μ egy későbbiekben definiált, a feladat paramétereitől függő konstans és legyen $c_e(j) = \mu^{l_e(j)}$, minden j kérés és e él esetén. Ekkor az algoritmus egy (s_j, t_j, r_j, b_j) kérést a következőképpen bírál el.

EXP(s_j, t_j, r_j, b_j)

1 U_j legyen az (s_j, t_j) utak halmaza
2 $P_j \leftarrow \operatorname{argmin}_{P \in U_j} \{\sum_{e \in P} \frac{r_j}{u(e)} c_e(j)\}$
3 **if** $C(P_j) = \sum_{e \in P_j} \frac{r_j}{u(e)} c_e(j) \leq 2mb_j$
4 **then** lefoglaljuk a P_j úton az r_j sávszélességet
5 **else** visszautasítjuk a kérést

Megjegyzés. Amennyiben a fenti algoritmusban minden kérést elfogadunk, akkor a terhelést kiegyensúlyozó modellre kapjuk meg az exponenciális algoritmust.

27.6. példa. Tekintsük azt a hálózatot, amely az A, B, C és D pontokból, valamint az (A, B), (B, D), (A, C) és (C, D) élekből áll, ahol az élek maximális sávszélességére $u(A, B) = 1$, $u(B, D) = 3/2$, $u(A, C) = 2$,$u(C, D) = 3/2$. Tegyük fel, hogy $\mu = 10$ és a j-edik kérés előtt az eddig lefoglalt sávszélességek 3/4 az A, B, D úton, 5/4 az A, C, D úton, 1/2 a (B, D) élen, 1/2 az (A, C) élen. Ekkor az $l_e(j)$ értékek: $l_{(A,B)}(j) = (3/4) : 1 = 3/4$, $l_{(B,D)}(j) = (3/4 + 1/2) : (3/2) = 5/6$, $l_{(A,C)}(j) = (5/4 + 1/2) : 2 = 7/8$, $l_{(C,D)}(j) = (5/4) : (3/2) = 5/6$. Legyen a következő kérés 1/8 sávszélesség lefoglalása A és D között. A két lehetséges útra a $C(P)$ értékek

$$C(A, B, D) = 1/8 \cdot 10^{3/4} + 1/12 \cdot 10^{5/6} = 1.269 \,,$$

$$C(A, C, D) = 1/16 \cdot 10^{7/8} + 1/12 \cdot 10^{5/6} = 1.035 \,.$$

A minimális érték a (A, C, D) úton van. Így, amennyiben a tárgyra $2mb_j = 8b_j \geq 1,035$ a kérést az (A, C, D) úton elfogadjuk.

Az algoritmus elemzéséhez tekintsük kérések egy tetszőleges I bemeneti sorozatát. Jelölje A az EXP algoritmus által elfogadott kérések halmazát, A^* az OPT algoritmus által elfogadott, de az EXP algoritmus által elutasított kérések halmazát. Továbbá minden olyan j kérésre, amelyet az OPT algoritmus elfogad, jelölje az OPT által hozzárendelt utat $P_j{}^*$. Az $l_e(j)$ értékekhez hasonlóan definiáljuk az $l_e(v) = (\sum_{i \in A, e \in P_i} r_i)/u(e)$ értéket, amely azt adja meg, hogy az e-n keresztül lefoglalt sávszélesség a megengedett sávszélességnek mekkora része az eljárás végén.

Most tegyük fel, hogy $\mu = 4mPB$, ahol B felső korlátja a nyereségeknek, továbbá minden kérésre és élre teljesül

$$\frac{1}{P} \leq \frac{r(j)}{u(e)} \leq \frac{1}{\lg \mu} \,.$$

Ekkor az algoritmus versenyképességére vonatkozó állítás kimondása előtt igazoljuk a következő lemmákat.

27.8. lemma. *Az* EXP *algoritmus által kapott megoldás lehetséges, azaz egyetlen élen sem lépjük túl a megengedett sávszélességet.*

Bizonyítás. Az állítást indirekt igazoljuk. Tegyük fel, hogy a megengedett sávszélességet az f élen túllépjük. Legyen j az első olyan kérés, amelynek az elfogadásával túlléptük a megengedett sávszélességet.

Mivel minden élre és kérésre, így j-re és f-re is teljesül, hogy $r_j/u(f) \leq 1/\lg\mu$ és a j kérés elfogadása után az f élen túllépjük a megengedett sávszélességet, ezért adódik, hogy $l_f(j) > 1 - 1/\lg\mu$. Másrészt ekkor az EXP algoritmus második lépésében számolt $C(P_j)$ értékre

$$C(P_j) = \sum_{e\in P_j} \frac{r_j}{u(e)} c_e(j) \geq \frac{r_j}{u(f)} c_f(j) > \frac{r_j}{u(f)} \mu^{1-1/\lg\mu} .$$

Szintén a tétel feltételei alapján $\frac{r_j}{u(e)} \geq \frac{1}{P}$, továbbá $\mu^{1-1/\lg m} = \mu/2$, így a fenti egyenlőtlenség alapján azt kapjuk, hogy

$$C(P) > \frac{1}{P}\frac{\mu}{2} = 2mB .$$

Másrészt ezzel az egyenlőtlenséggel ellentmondáshoz jutottunk, hiszen amennyiben a fenti egyenlőtlenség fennáll, akkor EXP visszautasítja a kérést. Mivel ellentmondáshoz jutottunk, ezért a kiinduló feltevésünk hamis kell legyen, azaz a lemma állítását igazoltuk. ■

27.9. lemma. *Az* OPT *algoritmus által kapott megoldásra teljesül*

$$\sum_{j\in A^*} b_j \leq \frac{1}{2m} \sum_{e\in E} c_e(v) .$$

Bizonyítás. Mivel EXP minden $j \in A^*$ kérést visszautasított, ezért minden $j \in A^*$-ra $b_j < \frac{1}{2m} \sum_{e\in P_j^*} \frac{r_j}{u(e)} c_e(j)$, hiszen a fenti egyenlőtlenség minden (s_j, t_j) útra teljesül, így az optimális megoldás által választott útra is. Tehát

$$\sum_{j\in A^*} b_j < \frac{1}{2m} \sum_{j\in A^*} \sum_{e\in P_j^*} \frac{r_j}{u(e)} c_e(j) .$$

Másrészt minden e élre $c_e(j) \leq c_e(v)$, így a fenti egyenlőtlenség alapján adódik, hogy

$$\sum_{j\in A^*} b_j < \frac{1}{2m} \sum_{e\in E} c_e(v) \Big(\sum_{j\in A^*: e\in P_j^*} \frac{r_j}{u(e)} \Big) .$$

Mivel minden e élre az OPT által elfogadott teljes sávszélesség legfeljebb $u(e)$, ezért minden e-re $\sum_{j\in A^*: e\in P^*(j)} \frac{r_j}{u(e)} \leq 1$. Következésképpen

$$\sum_{j\in A^*} b_j \leq \frac{1}{2m} \sum_{e\in E} c_e(v)$$

adódik, amely pontosan a lemma állítása. ■

27.10. lemma. *Az* EXP *algoritmus által kapott megoldásra teljesül*

$$\frac{1}{2m}\sum_{e\in E} c_e(v) \le (1+\lg\mu)\sum_{j\in A} b_j \ .$$

Bizonyítás. A lemma igazolásához elegendő belátnunk, hogy minden $j \in A$ kérésre teljesül $\sum_{e\in P_j}(c_e(j+1)-c_e(j)) \le 2mb_j\log_2\mu$. Másrészt

$$c_e(j+1)-c_e(j) = \mu^{l_e(j)+\frac{r_j}{u(e)}} - \mu^{l_e(j)}\mu^{l_e(j)}(2^{\log_2\mu\frac{r_j}{u(e)}}-1) \ .$$

Mivel $2^x - 1 < x$, ha $0 \le x \le 1$, és a feltételeink alapján $0 \le \log_2\mu\frac{r_j}{u(e)} \le 1$, ezért azt kapjuk, hogy

$$c_e(j+1)-c_e(j) \le \mu^{l_e(j)}\log_2\mu\frac{r_j}{u(e)}.$$

A fentiekben kapott felső becsléseket összegezve adódik, hogy

$$\sum_{e\in P_j}(c_e(j+1)-c_e(j)) \le \log_2\mu\sum_{e\in P_j}\mu^{l_e(j)}\frac{r_j}{u(e)} = \log_2\mu\cdot C(P_j) \ .$$

Mivel EXP azokat a kéréseket fogadja el, amelyekre $C(P_j) \le 2mb_j$, ezért a fenti egyenlőtlenség alapján adódik a lemma állítása. ■

A fenti lemmák alapján igazoljuk a következő tételt.

27.11. tétel. *Az* EXP *algoritmus* $1/O(\lg\mu)$*-versenyképes, ha* $\mu = 4mPB$*, ahol* B *felső korlátja a nyereségeknek, továbbá minden kérésre és élre teljesül*

$$\frac{1}{P} \le \frac{r(j)}{u(e)} \le \frac{1}{\lg\mu} \ .$$

Bizonyítás. A 27.8. lemma alapján adódik, hogy az eljárás korrekt nem lépjük át a megengedett sávszélességet. A fentiekben bevezetett jelöléseket használva a EXP algoritmus nyeresége az I inputon $\mathrm{EXP}(I) = \sum_{j\in A} b_j$, az OPT algoritmus nyeresége pedig legfeljebb $\sum_{j\in A\cup A^*} b_j$. Következésképpen a 27.9 és 27.10 lemmák alapján azt kapjuk, hogy

$$\mathrm{OPT}(I) \le \sum_{j\in A\cup A^*} b_j \le (2+\log_2\mu)\sum_{j\in A} b_j \le (2+\log_2\mu)\mathrm{EXP}(I) \ ,$$

amely egyenlőtlenség igazolja a tétel állítását. ■

Gyakorlatok

27.3-1. Tekintsük a nyugtázási feladat azon változatát, amelyben a célfüggvény, $k+\sum_{j=1}^{k}\mu_j$, ahol k a nyugták száma és $\mu_j = \max_{t_{j-1}<a_i\le t_j}\{t_j - a_i\}$ a j-edik nyugta által összegyűjtött maximális késedelem. Igazoljuk, hogy az Ébreszt algoritmus ezen célfüggvény esetén is 2-versenyképes.

27.3-2. Reprezentáljuk a lapletöltési feladat azon speciális esetét, amelyben minden lapra $s(g) = c(g) = 1$, a k-szerver feladat speciális eseteként. Milyen metrikus teret használhatunk?

27.3-3. Legyen a lapletöltési feladatban a 8 méretű HA memóriában három lap a, b, c, amelyeknek a mérete és az aktuális kreditértékei $s(a) = 3, s(b) = 2, s(c) = 3, cr(a) = 2, cr(b) = 1/2, cr(c) = 2$. Egy d lapot akarunk letölteni, amelynek a mérete 3, és a letöltési költsége 4. A 6 méretű memóriával rendelkező OPT algoritmus már letöltötte a lapot, a memóriájában d és c szerepel. Adjuk meg, miként helyezi el a HÁZIÚR algoritmus a d lapot és adjuk meg a 27.7. tétel bizonyítása során használt potenciálfüggvény változásait.

27.3-4. Igazoljuk, hogy amennyiben a nyereségmaximalizáló forgalomirányítási modellben az $r(j)/u(e)$ hányadosokra semmiféle kikötést nem teszünk, akkor nem adható meg konstans-versenyképes algoritmus.

27.4. On-line ládapakolási modellek

A ládapakolási feladatnak és különböző változatainak számos gyakorlati alkalmazása van. Ebben az alfejezetben ezzel a feladattal foglalkozunk, az első részben ismertetjük a klasszikus on-line ládapakolásra vonatkozó alapvető eredményeket. Az alfejezet második részében megadjuk a lehetséges többdimenziós általánosításokat és részletesebben vizsgáljuk a sávpakolási feladatot.

27.4.1. On-line ládapakolás

A ládapakolási feladatban bemenetként tárgyak egy sorozatát kapjuk meg, ahol az i-edik tárgyat a mérete határozza meg, ami egy $a_i \in (0, 1]$ érték. Célunk a tárgyak elhelyezése a lehető legkevesebb egység méretű ládába. Formálisabban megfogalmazva a tárgyakat olyan csoportokba akarjuk osztani, hogy minden csoportra a benne levő tárgyakra a $\sum a_i \leq 1$ feltétel teljesüljön. A feladat modellezi a memóriakezelés esetén az állományok optimális elhelyezésének feladatát is, amely kérdéskört ezen könyv első kötetében a memóriagazdálkodásról szóló fejezetben is tárgyaljuk.

Ebben a részben az on-line ládapakolási feladatot vizsgáljuk, amelyben az algoritmusnak az i-edik tárgyat az $a_1, \ldots, a_i$ értékek alapján kell elhelyezni a további tárgyakra vonatkozó információk ismerete nélkül.

Az NF algoritmus, helykorlátos algoritmusok

Érdemes külön megemlítenünk azt a modellt, amelyben az egyidőben nyitott ládák száma korlátozott. Ez azt jelenti, hogy amennyiben a nyitott ládák száma elér egy k-korlátot, akkor ezt követően csak akkor nyithatunk új ládát, ha az eddig nyitott ládák valamelyikét bezárjuk, ami azt jelenti, hogy többet nem használhatjuk. Ha a nyitott ládák száma csak egy lehet, akkor természetesen adódik az algoritmus, amely az aktuálisan nyitott ládába teszi a tárgyat ha az elfér, ellenkező esetben bezárja a nyitott ládát és új ládát nyit. Ezt az algoritmust NF algoritmusnak nevezzük. Az algoritmus pszeudokódját nem ismertetjük, a memóriagazdálkodás fejezetben megtalálható. Az NF algoritmus aszimptotikus versenyképességi hányadosára teljesül a következő állítás.

27.12. tétel. *Az* NF *algoritmus aszimptotikus versenyképességi hányadosa 2.*

Bizonyítás. Vegyünk egy tetszőleges σ tárgysorozatot. Jelölje n az OPT algoritmus által használt ládák számát, jelölje m az NF algoritmus által használt ládák számát. Továbbá legyen S_i, $i = 1, \ldots, m$ az i-edik ládában levő tárgyak méreteinek összege.

Ekkor $S_i + S_{i+1} > 1$, hiszen ellenkező esetben az $(i+1)$-edik láda első eleme elfért volna az i-edik ládában, ami ellentmond az algoritmus definíciójának. Következésképp a tárgyak méreteinek összege több, mint $\lfloor m/2 \rfloor$.

Másrészt az optimális off-line algoritmus sem rakhat 1-nél több összméretű tárgyakat egy ládába, így azt kapjuk, hogy $n > \lfloor m/2 \rfloor$. Ez pedig azt jelenti, hogy $m \leq 2n - 1$, így

$$\frac{\mathrm{NF}(\sigma)}{\mathrm{OPT}(\sigma)} \leq \frac{2n-1}{n} = 2 - 1/n \; .$$

Következésképpen igazoltuk, hogy az algoritmus aszimptotikusan 2-versenyképes.

Most megmutatjuk, hogy a fenti korlát éles. Ehhez tekintsük minden n-re a következő σ_n sorozatot. A sorozat $4n - 2$ tárgyból áll, a $(2i - 1)$-edik tárgy mérete $1/2$, a $2i$-edik tárgy mérete $1/4n$, ahol $i = 1, \ldots, 2n$. Ekkor az NF algoritmus az i-edik ládába a $(2i - 1)$-edik és a $2i$-edik tárgyat teszi, és $\mathrm{NF}(\sigma_n) = 2n - 1$. Az optimális algoritmus az első $n - 1$ ládába $1/2$ méretű tárgyakat párosít, és az n-edik ládába egy $1/2$ méretűt és a kis tárgyakat teszi, így $\mathrm{OPT}(\sigma_n) = n$. Mivel $\mathrm{NF}(\sigma_n)/\mathrm{OPT}(\sigma_n) = 2 - 1/n$ a 2 értékhez konvergál, ha n tart végtelenbe, ezért igazoltuk, hogy az algoritmus aszimptotikus versenyképességi hányadosa legalább 2. ■

Itt érdemes megemlítenünk, hogy amennyiben egynél több (de korlátozott számú) láda lehet nyitva, az NF algoritmusnál jobb algoritmusok is ismertek. A jelenlegi legjobb algoritmusok a ***harmonikus algoritmusok*** családjába tartoznak, ahol az alapötlet az, hogy a $(0, 1]$ intervallumot részintervallumokra osztjuk, és minden tárgynak az az intervallum lesz a típusa, amely intervallumba a mérete esik. A különböző típusú tárgyakat különböző ládákba pakoljuk, az algoritmus párhuzamosan alkalmaz egy-egy NF algoritmust az egyes típusokhoz tartozó tárgyakra.

Az FF algoritmus, a súlyfüggvény technika

Ebben a részben egy olyan módszert mutatunk be, amelyet gyakran használunk a ládapakolási algoritmusok elemzése során. A módszert az FF (First Fit) algoritmuson ismertetjük.

Az FF algoritmus az NF algoritmus továbbfejlesztett változata, arra az esetre, amelyben nincs korlátozva a nyitott ládák száma. Az algoritmus az aktuálisan megérkező tárgyat mindig a legkorábban kinyitott ládába teszi, ahol elfér. Ha nem fér el egyik ládában sem, kinyit egy új ládát és abba rakja. Az algoritmus pszeudokódja megtalálható a memóriagazdálkodásról szóló fejezetben. Az algoritmus versenyképességi hányadosára ad felső korlátot a következő állítás.

27.13. tétel. *Az* FF *algoritmus aszimptotikusan 1.7-versenyképes.*

Bizonyítás. A bizonyítás alapötlete a súlyfüggvény technika, amely azt jelenti, hogy minden tárgyhoz egy súlyt rendelünk, amely azt adja meg valamilyen értelemben, hogy mennyire sok helyet foglalhat el a tárgy egy pakolásban. A tárgyaknak vesszük az összsúlyát, és ezen

érték segítségével becsüljük meg az off-line és az on-line célfüggvények értékét. Definiáljuk a következő súlyfüggvényt:

$$w(x) = \begin{cases} 6x/5, & \text{ha } 0 \le x \le 1/6 \\ 9x/5 - 1/10, & \text{ha } 1/6 \le x \le 1/3 \\ 6x/5 + 1/10, & \text{ha } 1/3 \le x \le 1/2 \\ 6x/5 + 2/5, & \text{ha } 1/2 < x\ . \end{cases}$$

A tárgyak egy tetszőleges H halmazára legyen $w(H) = \sum_{i \in H} w(a_i)$. Ekkor a súlyfüggvényre teljesülnek az alábbi állítások. Mivel mindkét lemma bizonyítása azon alapul, hogy eseteket különböztetünk meg a tárgyak méretétől függően, továbbá a bizonyítások hosszúak és sok technikai részletet tartalmaznak, ezért a bemutatásuktól itt eltekintünk.

27.14. lemma. *Amennyiben tárgyak egy H halmazára teljesül, hogy $\sum_{i \in H} a_i \le 1$, akkor ezen tárgyakra $w(H) \le 17/10$.*

27.15. lemma. *Tárgyak tetszőleges L listájára $w(L) > \mathrm{FF}(L) - 2$.*

A lemmák alapján könnyen igazolható, hogy az algoritmus aszimptotikusan 1.7-versenyképes. Tekintsük tárgyaknak egy tetszőleges L listáját. Mivel az optimális off-line algoritmus el tudja pakolni a lista elemeit $\mathrm{OPT}(L)$ ládába úgy, hogy minden ládába a tárgyak méreteinek összege legfeljebb 1, ezért az első lemma alapján $w(L) \le 1.7\mathrm{OPT}(L)$. Másrészt a második lemma alapján $\mathrm{FF}(L) - 2 \le w(L)$, így azt kapjuk, hogy $FF(L) \le 1.7\mathrm{OPT}(L) + 2$, amiből következik, hogy az algoritmus aszimptotikusan 1.7-versenyképes.

Fontosnak tartjuk megjegyezni, hogy a fentiekben igazolt felső korlát éles, azaz a FF algoritmusra az is igaz, hogy az aszimptotikus versenyképességi hányadosa 17/10. ■

Érdemes megjegyeznünk, hogy számos az FF algoritmusnál kisebb versenyképességi hányadossal rendelkező algoritmus került kifejlesztésre. A jelenleg ismert legjobb algoritmus aszimptotikus versenyképességi hányadosa 1.5888.

Alsó korlátok

Ebben a részben azt vizsgáljuk, miként találhatunk általános alsó korlátokat a lehetséges versenyképességi hányadosokra. Elsőként egy egyszerű alsó korlátot tekintünk, ezt követően megmutatjuk, miként általánosítható a bizonyítás alapgondolata egy általános módszerré.

27.16. tétel. *Nincs olyan on-line algoritmus a ládapakolási feladatra, amelynek az aszimptotikus versenyképességi hányadosa kisebb, mint 4/3.*

Bizonyítás. Legyen A egy tetszőleges on-line algoritmus. Tekintsük tárgyaknak a következő sorozatát. Legyen $\varepsilon < 1/12$ és L_1 egy n darab $1/3 + \varepsilon$ méretű tárgyból álló sorozat, L_2 pedig n darab $1/2 + \varepsilon$ méretű tárgyból álló sorozat. Elsőként az algoritmus megkapja az L_1 listát. Ekkor az algoritmus bizonyos ládákba két tárgyat tesz, bizonyos ládákba egyet. Jelölje k azon ládák számát, amelyek két tárgyat tartalmaznak. Ekkor az algoritmus költsége $A(L_1) = k + n - 2k = n - k$. Másrészt az optimális off-line algoritmus minden ládába két tárgyat tesz, így a költség $OPT(L_1) = n/2$.

Amennyiben ugyanez az algoritmus az L_1L_2 összetett listát kapja, akkor az első részben szintén k ládát használ két tárgynak. (Az on-line algoritmus nem tudja, hogy az L_1 vagy az L_1L_2 lista alapján kapja a tárgyakat.) Következésképp az $1/2 + \varepsilon$ méretű tárgyak közül csak $n - 2k$ darabot párosíthat az előző tárgyakhoz, az összes többihez új ládát kell nyitnia. Tehát $A(L_1L_2) \geq n - k + (n - (n - 2k)) = n + k$. Másrészt az optimális off-line algoritmus minden ládába egy kisebb, $1/3 + \varepsilon$ méretű és egy nagyobb, $1/2 + \varepsilon$ méretű tárgyat tesz, így $OPT(L_1L_2) = n$.

Következésképpen azt kapjuk, hogy az A on-line algoritmusra van olyan L lista, amelyre

$$A(L)/OPT(L) \geq \max\left\{\frac{n-k}{n/2}, \frac{n+k}{n}\right\} \geq 4/3 \ .$$

Másrészt a fenti hányadosokban az $OPT(L)$ érték legalább $n/2$, ami tetszőlegesen nagynak választható. Így a fenti egyenlőtlenségből adódik, hogy az A algoritmus aszimptotikus versenyképességi hányadosa legalább $4/3$, amivel a tétel állítását igazoltuk. ■

A fenti bizonyítás alapötlete, hogy egy hosszabb tárgysorozatot (a fentiekben L_1L_2) veszünk és az algoritmus viselkedésétől függően választjuk ki azt a kezdőszeletét a sorozatnak, amelyre a költségek hányadosa maximális. Természetes gondolat a bizonyításban használt sorozatnál bonyolultabb sorozatot használni. Több alsó korlát született különböző sorozatok felhasználásával. Másrészt a sorozatok elemzéséhez szükséges számítások egyre bonyolultabbak lettek. Az alábbiakban megmutatjuk, miként írható fel a sorozat elemzése vegyes egészértékű programozási feladatként, amely lehetővé teszi, hogy az alsó korlátot számítógép segítségével határozzuk meg.

Tekintsük a következő tárgysorozatot. Legyen $L = L_1L_2 \ldots L_k$, ahol L_i $n_i = \alpha_i n$ egyforma méretű tárgyat tartalmaz, amelyek mérete a_i. Amennyiben egy A algoritmus aszimptotikusan C-versenyképes akkor minden j-re teljesülnie kell a

$$C \geq \limsup_{n\to\infty} \frac{\mathrm{A}(L_1 \ldots L_j)}{\mathrm{OPT}(L_1 \ldots L_j)}$$

feltételnek. A fentiekben tekinthetjük azt az algoritmust, amelyre az általunk adható alsó korlát minimális, így célunk az

$$R = \min_{\mathrm{A}} \max_{j=1,\ldots,k} \limsup_{n\to\infty} \frac{\mathrm{A}(L_1 \ldots L_j)}{\mathrm{OPT}(L_1 \ldots L_j)}$$

érték meghatározása, amely érték egy alsó korlát lesz a versenyképességi hányadosra. Ezen érték meghatározható egy vegyes egészértékű programozási feladat optimumaként. A feladat megadásához szükségünk van a következő fogalmakra.

Egy tetszőleges ládára a láda tartalma leírható a láda pakolási mintájával, amely azt adja meg, hogy az egyes részlistákból hány elemet tartalmaz a láda. A ***pakolási minta*** egy k-dimenziós vektor $(p_1, \ldots, p_k)$, amelynek a p_j koordinátája azt adja meg, hány elemet tartalmaz a láda az L_j részlistából. Pakolási minta olyan nemnegatív egész koordinátájú vektor lehet, amelyre a $\sum_{j=1}^{k} a_j p_j \leq 1$ feltétel teljesül. (Ez a feltétel azt írja le, hogy a minta által leírt tárgyak valóban elférnek egy ládában.)

Osztályozzuk a lehetséges minták T halmazát a következőképpen. Minden j-re legyen T_j azon minták halmaza, amelyeknek az első pozitív együtthatója a j-edik. (Egy p minta a

T_j halmazba kerül, ha $p_i = 0$ minden $i < j$ esetén, és $p_j > 0$.)

Most tekintsük az A algoritmus által kapott pakolást. Az algoritmus minden ládát valamely pakolási minta alapján töltött meg, így az algoritmus által kapott pakolás leírható a pakolási minták segítségével. Jelölje $n(p)$ minden $p \in T$ esetén azon ládák számát, amely ládákat a p mintának megfelelően pakolt az algoritmus.

Vegyük észre, hogy egy láda, amely egy a T_j osztályba eső mintának megfelelően lett megtöltve, az első elemét az L_j részlistából kapja. Következésképpen azt kapjuk, hogy az algoritmus által az $L_1 \dots L_j$ részlista pakolása során kinyitott ládák száma a következőképpen adható meg az $n(p)$ értékekkel:

$$A(L_1 \dots L_j) = \sum_{i=1}^{j} \sum_{p \in T_i} n(p) \ .$$

Tehát egy adott n-re a keresett A értéket a következő vegyes egészértékű programozási feladat megoldásával számíthatjuk ki.

Min R

$$\sum_{p \in T} p_j n(p) = n_j, \qquad 1 \le j \le k$$

$$\sum_{i=1}^{j} \sum_{p \in T_i} n_p \le R \cdot OPT(L_1 \dots L_j), \qquad 1 \le j \le k$$

$$n(p) \in \{0, 1, \dots\}, \qquad p \in T \ .$$

Az első k feltétel azt írja le, hogy az összes tárgyat el kell helyeznünk a ládákban. A második k feltétel az írja le, hogy az R érték valóban nem kisebb, mint az algoritmus költségének és az optimális költségnek a hányadosa a vizsgált részlistákra.

Az $L_1 L_2 \dots L_k$ lista alapján a pakolási minták T halmaza és az optimális $OPT(L_1 \dots L_j)$ értékek meghatározhatók.

A feladatban a változók igen nagy értékeket vehetnek fel és a változók száma is nagy lehet, ezért a feladat helyett a lineáris programozási relaxációt szokás tekinteni. Továbbá a megoldást azon feltétel mellett kell meghatároznunk, hogy n tart a végtelenbe, és ezen feltétel mellett az egészértékű feladat és a relaxáció ugyanazokat a korlátokat adják.

Az eljárást megfelelően választott listákra alkalmazva kapták meg a jelenleg ismert legjobb alsó korlátot, amely azt mondja ki, hogy nincs olyan on-line algoritmus, amelynek kisebb az aszimptotikus versenyképességi hányadosa, mint 1.5401.

27.4.2. Többdimenziós modellek

A ládapakolási feladatnak három különböző többdimenziós általánosítása van, a vektorpakolási, a dobozpakolási és a sávpakolási modellek. Ezen általánosítások közül csak a sávpakolási feladattal foglalkozunk részletesen, a többi általánosításnak csak a modelljét ismertetjük. A ***vektorpakolási feladatban*** a bemenet d dimenziós vektorokból áll, és ezeket a tárgyakat kell minimális számú egységnyi ládában szabályosan elhelyezni. Szabályosnak hívunk egy elhelyezést, ha az egy ládába tett vektorokra minden komponensben az értékek összege legfeljebb egy. Az on-line vektorpakolási feladatban a vektorok egyenként jönnek, és az egyes vektorokat a további vektorokra vonatkozó információ ismerete nélkül kell ládákhoz rendelnünk. A ***dobozpakolási feladatban*** a bemenet d dimenziós téglatestekből áll, és ezeket kell minimális számú d-dimenziós egységkockában elhelyeznünk. Az on-line dobozpakolási feladatban a téglatestek egyenként jönnek, és az egyes téglatesteket a további

téglatestekre vonatkozó információk ismerete nélkül kell ládákhoz rendelnünk.

On-line sávpakolás

A ***sávpakolási feladatban*** téglalapok egy halmaza adott a szélességükkel és magasságukkal, és a célunk az, hogy ezeket a téglalapokat elhelyezzük forgatások nélkül egy függőleges w szélességű sávba úgy, hogy minimalizáljuk a felhasznált rész magasságát. A továbbiakban feltételezzük, hogy a tárgyak magassága legfeljebb 1. Általában az ütemezés megosztott erőforrásokkal két dimenziót eredményez, az erőforrást és az időt. Ebben az esetben tekinthetjük a szélességet a felhasznált erőforrás nagyságának, a magasságot pedig a felhasznált időnek, így célunk a felhasznált idő minimalizálása. A feladat on-line változatát vizsgáljuk, ahol a téglalapok egy listáról érkeznek, és a megérkezett téglalapot el kell helyeznünk a függőleges sávban a további téglalapokra vonatkozó ismeretek nélkül. Az on-line sávpakolási feladatra kidolgozott algoritmusok többsége a polc algoritmusok családjába tartozik. az alábbiakban ezt az algoritmuscsaládot ismertetjük.

Polc algoritmusok

Egy alapvető módszer a téglalapok pakolására az, hogy polcokat definiálunk és a téglalapokat ezekre a polcokra helyezzük el. ***Polcon*** a feltöltendő sávnak egy vízszintes részét értjük. A Polc algoritmus minden téglalapot egy polcra helyez. Miután az algoritmus kiválasztotta azt a polcot, amely a téglalapot tartalmazni fogja, az algoritmus a téglalapot elhelyezi a polcon annyira balra, amennyire lehetséges a már a polcon levő egyéb téglalapok átfedése nélkül. Tehát a téglalap érkezése után az eljárásnak két döntést kell hoznia. Az első döntés az, hogy az eljárás kialakít-e egy új polcot vagy sem. Ha új polcot alakítunk ki, meg kell határoznunk a polc magasságát is. Az újonnan kialakított polcokat mindig az előző polc tetejére helyezzük, az első polc a sáv legalján van. A második döntés, hogy az algoritmusnak ki kell választani azt a polcot, amelyre a téglalapot helyezi. A továbbiakban akkor mondjuk, hogy egy *téglalap elhelyezhető* egy polcon, ha a polc magassága nem kisebb a téglalap magasságánál és a polcon elég hely van ahhoz, hogy a téglalapot elhelyezzük rajta.

Egy eljárást vizsgálunk részletesen a fenti feladat megoldására. Ez az algoritmus a Baker és Schwarz által 1983-ban kifejlesztett NFS_r algoritmus. Az algoritmus egy $r < 1$ paramétertől függ. Az algoritmus minden j-re legfeljebb egy r^j magasságú aktív polcot tart fent és egy tárgy érkezése után a következő szabállyal definiálhatjuk.

A $p_i = (w_i, h_i)$ téglalap érkezése után válasszunk egy olyan k értéket, amelyre teljesül, hogy $r^{k+1} < h_i \leq r^k$. Amennyiben van r^k magasságú aktív polc, és a téglalap elhelyezhető ezen a polcon, akkor helyezzük el. Ellenkező esetben alakítsunk ki egy új r^k magasságú polcot, helyezzük el a téglalapot rajta, és a továbbiakban legyen ez a polc az r^k magasságú aktív polc (ha volt korábbi aktív polc, azt lezárjuk).

27.7. példa. Legyen $r = 1/2$. Legyen az első tárgy mérete $(w/2, 3/4)$. Ez a tárgy 1 magasságú polcra kerül. Ekkor létrehozunk egy 1 magasságú polcot a sáv legalján, ez lesz az 1-magasságú aktív polc, és ennek a polcnak a bal sarkára helyezzük el a tárgyat. Legyen a következő tárgy mérete $(3w/4, 1/4)$. Ez a tárgy 1/4 magasságú polcra kerül. Mivel nincs ilyen aktív polc, ezért létrehozunk egy 1/4 magasságú polcot az előző 1 magasságú polc tetején, ez lesz az 1/4 magasságú aktív polc, és ennek a polcnak a bal sarkára helyezzük el a tárgyat. Legyen a következő tárgy mérete $(3w/4, 5/8)$. Ez a tárgy ismét 1 magasságú polcra kerül. Mivel az 1 magasságú aktív polcon nem fér el, ezért azt lezárjuk, és létrehozunk egy új 1 magasságú polcot az előző 1/4 magasságú polc tetején, ez lesz az 1 magasságú aktív polc, és ennek a polcnak a bal sarkára helyezzük el a tárgyat. Legyen a következő tárgy mérete

$(w/8, 3/16)$. Ez a tárgy 1/4 magasságú polcra kerül. Az 1/4 magasságú aktív polcon még elfér a tárgy, ezért arra polcra rakjuk, annyira balra, amennyire lehetséges a második tárgy mellé.

NFS_r versenyképességére igazak az alábbi állítások.

27.17. tétel. *Az* NFS_r *algoritmus* $(\frac{2}{r} + \frac{1}{r(1-r)})$*-versenyképes. Az* NFS_r *algoritmus aszimptotikusan* $2/r$*-versenyképes.*

Bizonyítás. Tekintsük téglalapoknak egy tetszőleges L listáját, jelölje H a legmagasabb téglalap magasságát. Mivel ekkor $\mathrm{OPT}(L) \geq H$ teljesül, ezért a tétel állításának igazolásához elegendő belátnunk, hogy erre a sorozatra

$$\mathrm{NFS}_r(L) \leq 2/r\mathrm{OPT}(L) + H/r(1-r) \ .$$

Jelölje H_A az L lista végén az aktív polcok összmagasságát, H_Z pedig a többi lezárt polcok összmagasságát. Elsőként vizsgáljuk az aktív polcokat. Jelölje h a legmagasabb aktív polc magasságát. Ekkor az aktív polcok magasságai a hr^i értékeket vehetik fel és minden i-re legfeljebb egy aktív polc van hr^i magassággal. Tehát az aktív polcok összmagasságára

$$H_A \leq h \sum_{i=0}^{\infty} r^i = \frac{h}{1-r} \ .$$

Másrészt a $H > rh$ egyenlőtlenségnek is teljesülnie kell, hiszen különben a legmagasabb téglalap is elfért volna a legfeljebb rh magasságú polcokon és nem nyitottuk volna meg a h magasságú polcot. Következésképpen $H_A \leq H/r(r-1)$.

Most vizsgáljuk a lezárt polcokat. Vegyük egy tetszőleges i-re a hr^i magasságú polcokat, jelölje ezeknek a számát n_i. Minden ilyen lezárt S polcra a következő S' polc elsőként egy olyan elemet tartalmaz, amely már nem volt elhelyezhető, így a két egymást követő polcra az elhelyezett téglalapok teljes szélessége legalább w. Másrészt a hr^i magasságú polcokon minden tárgy magassága legalább hr^{i+1}, hiszen egyébként a tárgyat egy kisebb magasságú polcra helyeznénk. Tehát párosítva a lezárt polcokat és használva az aktív hr^i magasságú polcot is, ha a lezárt polcok száma páratlan, azt kapjuk, hogy az ilyen polcokon elhelyezett tárgyak összterülete legalább $wn_ihr^{i+1}/2$. Következésképpen az összes téglalapnak az összterülete legalább $\sum_{i=0}^{\infty} wn_ihr^{i+1}/2$, így $\mathrm{OPT}(L) \geq \sum_{i=0}^{\infty} n_ihr^{i+1}/2$. Másrészt a lezárt polcok összmagassága $H_Z = \sum_{i=0}^{\infty} n_ihr^i$, így azt kaptuk, hogy $H_Z \leq 2\mathrm{OPT}(L)/r$. Mivel $\mathrm{NFS}_r(L) = H_A + H_Z$, ezért a fentiek alapján adódik a kívánt egyenlőtlenség. ■

A fenti algoritmuson kívül további polc algoritmusokat is vizsgáltak a feladat megoldására. A fenti algoritmus alapgondolata, hogy az egyes polctípusokat ládaként fogjuk fel, és az adott polctípushoz rendelt tárgyakat az NFládapakolási algoritmussal helyezzük el. Természetes gondolat más ládapakolási algoritmusok használata. A jelenlegi legjobb polc algoritmust Csirik és Woeginger fejlesztették ki 1997-ben, amely algoritmus a harmonikus ládapakolási algoritmust használja az adott polctípusokhoz rendelt tárgyak elhelyezésére.

Gyakorlatok

27.4-1. Tegyük fel, hogy egyetlen tárgy mérete sem lehet nagyobb, mint 1/3. Igazoljuk, hogy ezen feltétel mellett NF aszimptotikus versenyképességi hányadosa 3/2.

27.4-2. Igazoljuk, hogy amennyiben egy ládában minden elem mérete legfeljebb 1/3, akkor teljesül a 27.15. lemma.

27.4-3. Tegyük fel, hogy a tárgyak listája az $L_1 L_2 L_3$ lista, ahol L_1 n darab 1/2 méretű tárgyat, L_2 n darab 1/3 méretű tárgyat, L_3 pedig n darab 1/3 méretű tárgyat tartalmaz. Milyen pakolási minták alapján lehet feltölteni a ládákat? Melyik minták esnek a T_2 osztályba?

27.4-4. Tekintsük a sávpakolási feladat azon változatát, amelyben a tárgyak megnyújthatóak oly módon, hogy a tárgyak területe ne változzon meg. Igazoljuk, hogy ebben az esetben NFS_r azon változata, amely a téglalap polcrahelyezése előtt a téglalapot megnyújtja úgy, hogy a polccal megegyező magasságú legyen, $2 + 1/(r(1-r))$-versenyképes.

27.5. On-line ütemezés

Az ütemezési feladatok elméletének igen nagy irodalma van. Az első on-line ütemezési eredmény tulajdonképpen Graham nevéhez kapcsolódik, aki 1966-ban elemezte a Listás ütemezés algoritmust. Mondhatjuk ezt annak ellenére, hogy Graham nem használta az on-line algoritmusok körében később elterjedt fogalomrendszert és az általa vizsgált algoritmust nem on-line algoritmusként vizsgálta, hanem heurisztikus algoritmusnak tekintette.

Speciálisan on-line ütemezési algoritmusokkal az 1990-es években kezdtek el foglalkozni és azóta számos eredmény született. Ebben a részben csak olyan a modellekkel foglalkozunk, ahol a cél a maximális befejezési időpont minimalizálása. Az ütemezési feladatokkal kapcsolatos fogalmakat és eredményeket ezen könyv első kötetében egy fejezet ismerteti, ezért itt csak az általunk az alábbiakban használt fogalmak definícióit elevenítjük fel.

Ütemezési feladatokban munkák végrehajtásait kell megterveznünk. Az általános modellben a munkadarabok több tevékenységből állhatnak és a tevékenységekhez kell meghatároznunk a gépeket, amelyeken és az időintervallumokat, amelyekben az egyes műveletek végrehajtandóak. A továbbiakban azt az egyszerűbb modellt vizsgáljuk, amelyben minden munka egyetlen műveletből áll. Tehát a feladatunk az, hogy a munkákhoz, amelyeknek ismerjük a megmunkálási idejét hozzárendeljük a gépet, amelyen a munkát végrehajtjuk és a megmunkálás kezdési és befejezési időpontját, amely időpontok különbsége szükségképpen a megmunkálási idő.

A gépek tekintetében három különböző modellel foglalkozunk. Amennyiben a munka megmunkálási ideje minden gépen ugyanannyi, akkor azonos párhuzamos gépekről beszélünk. Amennyiben a gépekhez hozzá van rendelve egy s_i sebesség, a munkáknak van egy p_j megmunkálási súlya és a j-edik munka megmunkálási ideje az i gépen p_j/s_i, akkor hasonló párhuzamos gépekről beszélünk. Végül, ha a j-edik munka megmunkálási ideje tetszőleges $P_j = (p_{j1}, \ldots, p_{jm})$ pozitív vektor lehet, ahol a munka megmunkálási ideje az i-edik gépen p_{ji}, akkor általános párhuzamos gépekről beszélünk.

Ütemezési feladatok esetén számos célfüggvényt szokás vizsgálni, ebben a fejezetben azt a modellt vizsgáljuk, amelyben a cél a maximális befejezési idő minimalizálása.

Az alfejezet első részében ismertetjük a két legelterjedtebb on-line ütemezési modellt, és a következő két fejezetben ezen modellekkel foglalkozunk.

27.5.1. On-line ütemezési modellek

Az alábbiakban definiáljuk a két legfontosabb on-line ütemezési modellt.

LISTA modell
Ebben a modellben a munkák egy listáról érkeznek. Amikor egy munkát megkapunk a listáról, akkor ismerjük meg a szükséges megmunkálási időt, ezt követően ütemeznünk kell a munkát, hozzárendelve a kezdési és a befejezési időt, amelyeket később már nem változtathatunk meg, és csak ezt követően kapjuk meg a listáról a következő munkát.

Vegyük észre, hogy amennyiben a célfüggvény a maximális befejezési idő, akkor (miként az off-line esetben is) elegendő olyan algoritmusokkal foglalkoznunk, amelyek nem hagynak üres részeket a gépeken, azaz amelyekben az egyes gépeken a munkák szünet nélkül követik egymást. Ebben az esetben minden gépre a maximális befejezési idő megegyezik a géphez rendelt megmunkálási idők összegével. Minden gépre a gépen levő megmunkálási idők összegét a gépen levő ***töltésnek*** hívjuk.

27.8. példa. Tekintsük a LISTA modellben az azonos párhuzamos gépek esetét, legyen két gép és vegyük a következő munkasorozatot, ahol a munkákat a megmunkálási idők által adjuk meg: $I = (4, 3, 2, 5)$. Ekkor egy on-line algoritmus elsőként a 4 megmunkálási idővel rendelkező munkát kapja csak meg, hozzárendeli valamelyik géphez, tegyük fel, hogy az M_1 géphez. Ezt követően az algoritmus a 3 megmunkálási idővel rendelkező munkát kapja csak meg, és ezt hozzá kell rendelnie valamelyik géphez, tegyük fel, hogy az M_2 géphez. Majd az algoritmus a 2 megmunkálási idővel rendelkező munkát kapja csak meg, és ezt hozzá kell rendelnie valamelyik géphez, tegyük fel, hogy ismét az M_2 géphez. Végül az algoritmus az 5 megmunkálási idővel rendelkező munkát kapja csak meg, és ezt hozzá kell rendelnie valamelyik géphez, tegyük fel, hogy az M_1 géphez. Ekkor a gépeken a töltés $4 + 5$ illetve $3 + 2$ és valóban elérhető, hogy a töltések legyenek a maximális befejezési idők, hiszen az első gépen a munkákat végrehajthatjuk a $(0, 4)$ és $(4, 9)$ időintervallumokban, a második gépen a $(0, 3)$ és $(3, 5)$ időintervallumokban.

IDŐ modell
A második modellben a munkáknak egy r_j ***érkezési ideje*** is van, a munkáról semmit sem tudunk az érkezési ideje előtt, de a munka érkezése után bármikor megkezdhetjük a munka végrehajtását, ha van olyan gépünk, amely nem dolgozik. Amennyiben egy munkát elkezdünk végrehajtani, akkor nem szakíthatjuk félbe.

27.9. példa. Az IDŐ modellre tekintsünk egy két darab hasonló párhuzamos gépekből álló példát. Legyen az M_1 gép sebessége 1, az M_2 gép sebessége 2. Vegyük a következő $I = ((1, 0), (1, 1), (1, 1), (1, 1))$ munkasorozatot, ahol a munkák a (megmunkálási idő, érkezési idő) párokkal vannak megadva. Tehát a 0 időpontban egyetlen munka érkezik 1 megmunkálási idővel, az algoritmus elkezdheti végrehajtani valamely gépen, de várhat is, arra számítva, hogy később nagyobb megmunkálási idővel rendelkező munkák érkeznek. Tegyük fel, hogy az algoritmus az 1/2 időpontig vár és akkor kezdi el végrehajtani a munkát az M_1 gépen. Az 1 időpontban megérkezik három újabb munka, ekkor csak az M_2 gép szabad. Kezdjük el végrehajtani az egyik (1, 1) munkát a gépen. A 3/2 időpontban válik szabaddá mindkét gép, ekkor mindkét géphez hozzárendelhetünk egy-egy munkát, amelyek az M_1 gépen az 5/2 az M_2 gépen a 2 időpontban fejeződnek be, így az algoritmus költsége 5/2. Látszik, hogy amennyiben egyből elkezdjük a 0 időpontban az első munka végrehajtását, akkor az algoritmus kisebb, 2 költséggel is ütemezhette volna a munkákat. Fontos megjegyeznünk, hogy en-

nek ellenére bizonyos esetekben hasznos lehet várakoztatni munkákat, ezzel helyett hagyva az esetleg később érkező nagyobb megmunkálási idővel rendelkező munkák számára.

27.5.2. A LISTA modell

Elsőként vegyük észre, hogy az ütemezésről szóló fejezetben tárgyalt listás ütemezés algoritmus valójában egy on-line algoritmus. Ez a LISTA algoritmus az aktuális tárgyat mindig ahhoz a géphez rendeli hozzá, ahol az aktuális töltés minimális. Az első kötetben igazolást nyert az az állítás, mely szerint az ütemezési feladat tetszőleges I bemenete esetén $\text{LISTA}(I) \leq (2 - 1/m)\text{OPT}(I)$,, amiből következik, hogy ezen algoritmus versenyképességi hányadosa $2 - 1/m$. Első ránézésre nehéz elképzelni más algoritmust az on-line esetre, de több algoritmust fejlesztettek ki, amelyek versenyképességi hányadosa 2-nél kisebb számhoz konvergál, amennyiben a gépek száma tart a végtelenhez. Ezen algoritmusok többsége azon az ötleten alapszik, hogy a gépek többségén igyekszik egyenletesen elosztani a munkákat, de a LISTA algoritmussal ellentétben a gépek egy részén alacsonyan tartja a töltést, biztonsági tartalékként fenntartva ezeket a gépeket az esetleges nagy megmunkálási idővel rendelkező munkáknak.

A továbbiakban az általánosabb eseteket – amelyekben a gépek nem azonosak – vizsgáljuk. Az általános modellben nyilvánvalóan nem elegendő azzal foglalkoznunk, melyik gépen minimális az aktuális töltés, hiszen azon a gépen nagyon nagy lehet a munka megmunkálási ideje. A LISTA algoritmus, amely mohó módon ütemezi a munkákat, a következőképpen általánosítható. A munkák helyét a következő szabály alapján határozzuk meg. Ütemezzük a munkát azon a gépen, ahol a töltés a munka ütemezése után minimális lesz. Ha több ilyen gép is van, akkor ezek közül azt választjuk, ahol a munka megmunkálási ideje a legkisebb és ha ilyen gépből is több van, akkor ezek közül a legkisebb indexű gépet választjuk. Ezt az algoritmust MOHÓ algoritmusnak nevezzük.

27.10. példa. Tekintsük a hasonló párhuzamos gépek esetét, ahol 3 darab gép van és a sebességek $s_1 = s_2 = 1$, $s_2 = 3$. Vegyük a következő $I = (2, 1, 1, 3, 2)$ munkasorozatot, ahol a munkák a megmunkálási idők által vannak megadva. Ekkor az első munka 2/3-kor fejezhető be az M_3 gépen és az 1 időpontban a többi gépen, így M_3-hoz rendeljük. A következő munka az 1 időpontra fejezhető be az összes gépen, így M_3 kapja, mivel ott a legkisebb a megmunkálási idő. A következő munka az 1 időpontra fejezhető be M_1-en és M_2-n, és a 4/3 időpontban az M_3 gépen, így M_1 kapja. A negyedik munka M_1-en a 4, M_2-n a 3 időpontban fejeződne be, az M_3-on a 2 időpontban, így M_3-ra kerül. Végül az utolsó munka M_1-en 3, M_2-n a 2 időpontban fejeződne be, az M_3-on 8/3, így M_2-re kerül.

27.11. példa. Tekintsük az általános párhuzamos gépek esetét, ahol 2 darab gép van. Vegyük a következő $I = ((1, 2), (1, 2), (1, 3), (1, 3))$ munkasorozatot, ahol a munkák a megmunkálási idővektorok által vannak megadva. Ekkor az első munka az 1 időpontban fejezhető be az M_1 gépen és a 2 időpontban a második gépen, így M_1-hez rendeljük. A következő munka a 2 időpontra fejezhető be mindkét gépen, így M_1 kapja. A következő munka a 3 időpontra fejezhető be mindkét gépen, így ismét M_1 kapja. Végül az utolsó munka a 4 időpontra fejezhető be az M_1 gépen és a 3 időpontra fejezhető be az M_2 gépen, így az M_2 géphez rendeljük.

Az algoritmus versenyképességi hányadosát határozzák meg az alábbi tételek.

27.18. tétel. *A* Monó *algoritmus versenyképességi hányadosa m az általános párhuzamos gépek esetében.*

Bizonyítás. Elsőként megmutatjuk, hogy az algoritmus versenyképességi hányadosa nem lehet kisebb, mint m. Tekintsük a következő munkasorozatot. Legyen $\varepsilon > 0$ egy kicsi szám. A sorozat m darab munkát fog tartalmazni. Az első munkára a megmunkálási idő az első gépen 1, az m-edik gépen $1 + \varepsilon$, a többi gépen ∞, $(p_1(1) = 1, p_1(i) = \infty, i = 2, \ldots, m - 1, p_1(m) = 1 + \varepsilon)$, ezt követően a j-edik munkára a megmunkálási idő j a j-edik gépen, $1 + \varepsilon$ a $j - 1$-edik gépen, és ∞ a többi gépen $(p_j(j-1) = 1 + \varepsilon, p_j(j) = j, p_j(i) = \infty$, ha $i \neq j - 1$ és $i \neq j)$.

Ezen munkasorozatra a Monó algoritmus a j-edik munkát a j-edik gépen ütemezi, és a maximális befejezési idő m. Másrészt az optimális off-line algoritmus az első munkát az m-edik gépen, utána a j-edik munkát a $(j-1)$-edik gépen ütemezi, így az optimális maximális befejezési idő $1 + \varepsilon$. A hányados $m/(1 + \varepsilon)$. Ez az érték m-hez tart, ha az ε érték 0-hoz tart, amivel az állítást igazoltuk.

Most megmutatjuk, hogy az algoritmus m-versenyképes. Tekintsünk egy tetszőleges munkasorozatot, legyen az optimális maximális befejezési idő L^*, továbbá legyen $L(k)$ az első k munka Monó algoritmus általi ütemezésében a maximális befejezési idő. Mivel az i-edik munka ütemezéséhez valamely gépen legalább $\min_j p_i(j)$ idő szükséges, és egyik gépen sem használhatunk L^*-nál több időt, ezért $mL^* \geq \sum_{i=1}^{n} \min_j p_i(j)$.

Most igazoljuk teljes indukcióval, hogy $L(k) \leq \sum_{i=1}^{k} \min_j p_i(j)$. Mivel az első munkát ahhoz a géphez rendeljük, ahol a leghamarabb végrehajtható, ezért az állítás $k = 1$-re igaz. Most legyen $1 \leq k < n$ és tegyük fel, hogy az állítás k-ra igaz. Tekintsük a $k + 1$-edik munkát. Legyen az l-edik gép, amelyre a munka megmunkálási ideje minimális. Ezen a gépen végrehajtva a munkát a megmunkálási idő legfeljebb $L(k) + p_{k+1}(l) \leq \sum_{i=1}^{k+1} \min_j p_i(j)$ (az indukciós feltevés alapján).

Mivel a Monó algoritmus által kapott maximális befejezési idő legfeljebb annyi, mint abban az esetben, ha a $k + 1$-edik munkát az l-edik géphez rendeljük, ezért $L(k + 1) \leq \sum_{i=1}^{k+1} \min_j p_i(j)$, azaz az állítást igazoltuk $k+1$-re, így tetszőleges n-nél nem nagyobb egészre.

Következésképpen azt kapjuk, hogy $mL^* \geq \sum_{i=1}^{n} \min_j p_i(j) \geq L(n)$, amivel igazoltuk, hogy az algoritmus m-versenyképes. ■

A hasonló párhuzamos gépek esetének vizsgálatához tekintsünk egy tetszőleges bemenetet. Legyen L az algoritmus által kapott maximális befejezési idő, L^* pedig az optimális off-line algoritmus által kapott maximális befejezési idő. Az elemzés az alábbi lemmákon alapul, amelyek az egyes gépeken levő töltés mennyiségére adnak alsó korlátot.

27.19. lemma. *A töltés a leggyorsabb gépen legalább* $L - L^*$.

Bizonyítás. Vegyük azt a munkát, amelynek a befejezési ideje megegyezik a maximális befejezési idővel. Ha a munka a leggyorsabb gépen van ütemezve, akkor az állítás nyilvánvalóan teljesül. Tegyük fel, hogy nem a leggyorsabb gépen ütemeztük. Mivel az optimális ütemezés költsége L^*, ezért ezen munkának a leggyorsabb gépen történő végrehajtása legfeljebb L^* ideig tarthat. Másrészt ezt a munkát úgy ütemeztük, hogy a befejezési ideje L lett, ami azt jelenti, hogy a munka ütemezésekor a töltésnek a leggyorsabb gépen legalább $(L - L^*)$-nak kellett lennie, hiszen különben a munkát ott ütemeztük volna. ■

27.20. lemma. *Amennyiben a töltés minden legalább v sebességű gépen legalább l, akkor a töltés legalább $l - 4L^*$ minden legalább $v/2$ sebességű gépen.*

Bizonyítás. Amennyiben $l < 4L^*$, az állítás nyilvánvalóan teljesül. Tegyük fel, hogy $l \geq 4L^*$. Tekintsük azon munkák halmazát, amelyeket a legalább v sebességű gépeken ütemezünk az $[l - 2L^*, l]$ intervallumban. Ezen munkák összsúlya nem lehet kevesebb, mint $2L^*$-szor a legalább v sebességű gépek sebességeinek az összege, következésképpen van olyan munka közöttük, amelyet az optimális off-line algoritmus lassabb gépen ütemez (hiszen ellenkező esetben nem lehetne az off-line maximális befejezési idő L^*).

Legyen egy ilyen munka j. Mivel v-nél lassabb gépen ütemezi az off-line algoritmus, ezért a megmunkálási súlya legfeljebb vL^*. Következésképpen ezen munka megmunkálási ideje a legalább $v/2$ sebességű gépeken legfeljebb $2L^*$. Mivel ezen munka befejezési ideje a Monó algoritmus mellett legalább $l - 2L^*$, ezért a munka ütemezésekor minden legalább $v/2$ sebességű gépen a töltés legalább $l - 4L^*$ volt, hiszen ellenkező esetben egy ilyen gépen ütemeztük volna a munkát. ■

A fenti lemmák alapján igazolhatjuk a következő állítást.

27.21. tétel. *Hasonló párhuzamos gépek esetén a* Monó *algoritmus versenyképességi hányadosa* $\Theta(\lg m)$.

Bizonyítás. Elsőként megmutatjuk, hogy az algoritmus $O(\lg m)$-versenyképes. Ehhez vegyünk egy tetszőleges bemenetet. Legyen L az algoritmus által kapott maximális befejezési idő, L^* pedig az optimális off-line algoritmus által kapott maximális befejezési idő.

Legyen $v_{\max}$ a leggyorsabb gép sebessége, ekkor ezen a gépen a 27.19. lemma alapján a töltés legalább $L - L^*$. Ezt követően többször alkalmazva a 27.20. lemmát azt kapjuk, hogy a legalább $v_{\max} 2^{-i}$ sebességű gépeken a töltés legalább $L - L^* - 4iL^*$. Következésképp a legalább $v_{\max}/m$ sebességű gépeken a töltés legalább $L-(1+4\lceil \lg m \rceil)L^*$. Jelölje I a legfeljebb $v_{\max}/m$ sebességű gépek halmazát.

Vizsgáljuk most meg a munkák megmunkálási súlyainak W összegét. Mivel az off-line algoritmus úgy osztja el a munkákat, hogy a maximális töltés L^*, és mivel legfeljebb m gép van, amelyeknek a sebessége kisebb, mint $v_{\max}/m$ ezért

$$W \leq L^* \sum_{i=1}^{m} v_i \leq mL^* v_{\max}/m + L^* \sum_{i \notin I} v_i \leq 2L^* \sum_{i \notin I} v_i \ .$$

Másrészt az on-line algoritmus ezeket a munkákat osztja szét, ezért nem lehetséges, hogy minden I-n kívül eső gépen a töltés $2L^*$-nál nagyobb legyen, hiszen ez a fenti felső korlátnál nagyobb W értéket eredményezne.

Következésképp azt kapjuk, hogy

$$L - (1 + 4\lceil \lg m \rceil)L^* \leq 2L^* \ ,$$

amiből adódik, hogy $L \leq 3 + 4\lceil \lg m \rceil)L^*$, azaz igazoltuk, hogy az algoritmus $O(\lg m)$-versenyképes.

Most megmutatjuk, hogy az algoritmus versenyképességi hányadosa nem lehet kisebb, mint $\Omega(\lg m)$. Tekintsük a következő halmazát a gépeknek. A G_0 halmazban van egy gépünk, amelynek a sebessége 1, a G_1 halmazban van 2 gépünk, amelyek sebessége 1/2, így folytatva a G_i halmazban olyan gépeink vannak, amelyek sebessége 2^{-i}. A halmaz elemszámát úgy definiáljuk, hogy a benne levő gépek annyian legyenek, ahány 2^{-i} súlyú munka végrehajtható a $G_0, G_1, \ldots, G_{i-1}$ halmazban levő gépeken összesen 1 egységnyi idő alatt. Formálisan $|G_i| = \sum_{j=0}^{i-1} |G_j| 2^{i-j}$. Definiáljunk $k+1$ ilyen halmazt. Egyszerű számolás alapján adódik, hogy ekkor $|G_i| = 2^{2i-1}$, ha $i \geq 1$, így a gépek száma $1 + (2/3)(4^k - 1)$.

Tekintsük a következő munkasorozatot. Elsőként az első fázisban érkezzen $|G_k|$ darab munka 2^{-k} súllyal, majd a második fázisban $|G_{k-1}|$ munka $2^{-(k-1)}$ súllyal, és így folytatva az i-edik fázisban $|G_i|$ munka 2^{-i} súllyal, egészen az utolsó, $k+1$-edik fázisig, ahol egy munka érkezik 1 súllyal. Egy off-line algoritmus ütemezheti az i-edik fázis munkáit a G_{k+1-i} géphalmaz gépein, ekkor a maximális befejezési idő 1, így az optimális off-line költség legfeljebb 1.

Vizsgáljuk meg a Mohó algoritmus viselkedését ezen a bemeneten. A G_i halmaz elemszámának definíciója alapján az első fázisban érkező munkák végrehajthatóak a $G_0, \ldots, G_{k-1}$ halmazba eső gépeken 1 idő alatt, és a G_k halmaz gépein is 1 ideig tart végrehajtani a fázisban érkező munkákat, ezért ezeket a munkákat az algoritmus a $G_0, \ldots, G_{k-1}$ halmaz gépein hajtja végre, azokon a gépeken utána 1 lesz a töltés, a G_k halmaz gépein 0. Ezt követően a második fázis munkáit az algoritmus a $G_0, \ldots, G_{k-2}$ halmazok gépein hajtja végre, a harmadik fázis munkáit a $G_0, \ldots, G_{k-3}$ halmazok gépein, végül az utolsó előtti és az utolsó fázis munkáját a G_0 halmazba eső gépen. Tehát a Mohó algoritmus költsége $k+1$, (ez a maximális befejezési idő a G_0-ba eső gépen), és mivel $k = \Omega(\lg m)$, ezért az állítást igazoltuk. ■

27.5.3. Az IDŐ modell

Ebben a modellben egyetlen algoritmust elemzünk, amelynek alapötlete, hogy a munkákat az érkezési idők alapján szétosztja és az egyes munkahalmazokat optimálisan a munkák ismeretében off-line módon ütemezi. Ezt az algoritmust ***intervallumonkénti ütemező algoritmusnak*** nevezzük és INTV algoritmusként jelöljük. Legyen t_0 az első munka érkezési ideje, $i = 0$ és ettől az időponttól kezdve az algoritmus a következő pszeudokóddal írható le:

INTV(I)

```
1  while a munkasorozatnak nincs vége
2        legyen H_i a t_i időpontig megérkezett és ütemezetlen munkák halmaza
3        legyen OFF_i ezen munkákra egy optimális off-line ütemezés
4        rendeljük hozzá a munkákat a gépekhez a kapott ütemezésnek megfelelően
5        legyen q_i a kapott maximális befejezési idő
6        if érkezett a (t_i, q_i] intervallumban új munka vagy vége van a munkasorozatnak
7           then t_{i+1} ← q
7           else  legyen t_{i+1} a következő munka érkezési ideje
8        i ← i + 1
```

27.12. példa. Tekintsük két párhuzamos gép esetét. Legyen a munkák sorozata a következő $I = (1,0),(1/2,0),(1/2,0),(1,3/2),(1,3/2),(2,2)$, ahol a munkákat a (megmunkálási idő, érkezési idő) párral adtuk meg. Az első iterációs részben az első három munkát ütemezzük, egy optimális ütemezés az első munkát az első géphez, a második és harmadik munkákat a második géphez rendeli, és az 1 időpontban befejezi a munkák végrehajtását. Utána, mivel nem érkezett új munka és nincs vége a sorozatnak, várunk egészen a 3/2 időpontig. Ekkor egy új iterációs rész kezdődik, az algoritmus ütemezi a negyedik és ötödik munkákat az első és második gépen, a munkákat az 5/2 időpontra fejezi be. Ekkor elkezdődik a harmadik iterációs rész, és az algoritmus ütemezi a hatodik munkát valamely gépen, az $[5/2, 9/2]$ intervallumra.

A INTV algoritmus versenyképességére vonatkozik a következő állítás.

27.22. tétel. *Az IDŐ modellben az* INTV *algoritmus 2-versenyképes.*

Bizonyítás. Vegyük az algoritmus által kapott ütemezését a munkáknak. Jelölje i az iterációk számát. Legyen $T_3 = t_{i+1} - t_i$, $T_2 = t_i - t_{i-1}$, $T_1 = t_{i-1}$ és T_{OPT} az optimális off-line költség. Ekkor $T_2 \leq T_{OPT}$. Ez az észrevétel a $t_{i+1} \neq q_{i+1}$ esetben nyilvánvaló. Ha $t_{i+1} = q_{i+1}$, akkor azért teljesül, mert az i-edik iterációban ütemezett munkákat az optimális algoritmusnak is ütemezni kell. Másrészt $T_1 + T_3 \leq T_{OPT}$. Ezen észrevétel igazolásához vegyük észre, hogy az i-edik iterációban ütemezett munkák mindegyikének legalább $T_1 = t_{i-1}$ az érkezési ideje, (ellenkező esetben már az $i-1$-edik lépésben ütemeztük volna őket). Következésképp az optimális algoritmus nem ütemezheti ezeket a munkákat a T_1 időpont előtt. Másrészt a munkák végrehajtásához legalább további T_3 idő kell. Mivel az algoritmus által kapott ütemezés költsége $T_1 + T_2 + T_3$, ezért a tétel állítása következik. ■

Később az algoritmusnál kisebb versenyképességi hányadossal rendelkező algoritmusokat is sikerült kifejleszteni, Vestjens igazolta, hogy az ***on-line*** **LPT** algoritmus, amely minden időpontban, amikor szabad használható gép van, a már megérkezett de még nem ütemezett munkák közül a legnagyobb megmunkálási idővel rendelkező munkát kezdi el végrehajtani a gépen, (3/2)-versenyképes. Szintén Vestjens igazolta az alábbi állítást.

27.23. tétel. *Nincs olyan on-line algoritmus az on-line ütemezés IDŐ modelljében a maximális befejezési idő minimalizálására, amelynek a versenyképességi hányadosa kisebb, mint 4/3.*

Bizonyítás. Legyen $\alpha = 0.3473$ az $\alpha^3 - 3\alpha + 1 = 0$ egyenlet $[1/3, 1/2]$ intervallumba eső megoldása. Igazoljuk, hogy egyetlen on-line algoritmusnak se lehet kisebb a versenyképességi hányadosa, mint $1 + \alpha$. Vegyünk egy tetszőleges on-line algoritmust, jelölje ALG. Tekintsük a következő munkasorozatot.

A 0 időpontban egyetlen munka érkezik, amelynek a megmunkálási ideje 1. Legyen S_1 az az időpont, amelyben az algoritmus elkezdi a munkát végrehajtani valamely gépen. Ha $S_1 > \alpha$, akkor erre az egyetlen munkából álló bemenetre $\mathrm{ALG}(I)/\mathrm{OPT}(I) > 1 + \alpha$, amivel az állítást igazoljuk. Tehát feltehetjük, hogy $S_1 \leq \alpha$.

A következő munka érkezzen az S_1 időpontban, a megmunkálási ideje legyen $\alpha/(1-\alpha)$. Legyen a kezdési ideje S_2. Amennyiben $S_2 \leq S_1 + 1 - \alpha/(1-\alpha)$, akkor a munkasorozatot m-1 darab munkával fejezzük be, amelyek mindegyikének az érkezési ideje S_2, a megmunkálási ideje $1+\alpha/(1-\alpha)-S_2$. Ekkor egy optimális off-line algoritmus az első két munkát ugyanazon a gépen ütemezi, a maradék $m-1$ munka mindegyikét pedig egy-egy gépen az S_2 időpontban

elkezdve, így az optimális maximális befejezési idő $1 + \alpha/(1 - \alpha)$. Másrészt az on-line algoritmus esetében az utolsó $m + 1$ munkából legalább egyet csak az első vagy a második munka befejezése után kezdhetünk el, így erre a bemenetre $\mathrm{ALG}(I) \geq 1 + 2\alpha/(1-\alpha)$, amiből adódik, hogy ebben az esetben sem lehet az algoritmus versenyképességi hányadosa kisebb, mint $1 + \alpha$. Következésképpen feltehetjük, hogy $S_2 > S_1 + 1 - \alpha/(1 - \alpha)$.

Ekkor az $S_1 + 1 - \alpha/(1 - \alpha)$ időpontban $m - 2$ darab munka érkezik, amelyeknek a megmunkálási ideje $\alpha/(1 - \alpha)$ és egy további munka, amelynek a megmunkálási ideje $1 - \alpha/(1 - \alpha)$. Az optimális ütemezésben a második és utolsó munka kivételével, minden munkát külön gépen hajtunk végre, a második és az utolsó munkát ugyanazon a gépen és így egy olyan ütemezést kapunk, amelyben a maximális befejezési idő $1 + S_1$. Mivel az $S_1 + 1 - \alpha/(1 - \alpha)$ időpont előtt az utolsó m munka egyikét sem kezdte el az on-line algoritmus, ezért van olyan gép, amelyen ezután az időpont után legalább két munkát kell végrehajtania, és ezen a gépen a maximális befejezési idő legalább $S_1 + 2 - \alpha/(1 - \alpha)$ lesz. Mivel $S_1 \leq \alpha$, ezért az $\mathrm{OPT}(I)/\mathrm{ALG}(I)$ hányados az $S_1\alpha$ érték mellett minimális, és ebben az esetben a hányados $1 + \alpha$, amivel az állítást igazoltuk. ■

Gyakorlatok

27.5-1. Igazoljuk, hogy nincs olyan on-line algoritmus, amelynek két azonos párhuzamos gép esetén kisebb a versenyképességi hányadosa, mint 3/2.

27.5-2. Igazoljuk, hogy a Lista ütemezési algoritmus nem konstans-versenyképes az általános párhuzamos gépek esetében.

27.5-3. Igazoljuk, hogy amennyiben az INTV algoritmusnál nem határozzuk meg minden lépésben az optimális off-line ütemezést, hanem helyette csak egy c-közelítő megoldást használunk, amelynek a költsége legfeljebb c-szer az optimális költség, akkor az így kapott változat $2c$-versenyképes.

Feladatok

27-1. Lapozási feladat

Vegyük a k-szerver feladat azon speciális esetét, ahol bármely két pont távolsága 1. (Ez a feladat ekvivalens az on-line lapozási feladattal.) Tekintsük azt az algoritmust, amely minden esetben, ha a kérés helyén nincs szerver, azt a szervert küldi a kérés kiszolgálására, amely szerver a legrégebben volt használva. (Ez az algoritmus a lapozásnál használt LRU algoritmusnak felel meg.) Igazoljuk, hogy az algoritmus k-versenyképes.

27-2. Az Ébreszt2 algoritmus

Vizsgáljuk meg a nyugtázási feladatra a következő Ébreszt2 algoritmust, ahol az általános algoritmusban szereplő e_j értékekre $e_j = 1/|\sigma_j|$. Igazoljuk, hogy ez az algoritmus nem konstans-versenyképes.

27-3. Ládapakolási alsó korlát

Igazoljuk, hogy nincs olyan on-line ládapakolási algoritmus, amelynek kisebb az aszimptotikus hányadosa, mint 3/2 egy olyan listát használva, amely $1/7 + \varepsilon$, $1/3 + \varepsilon$, $1/2 + \varepsilon$ méretű tárgyakat tartalmaz, ahol ε egy kicsi pozitív szám.

27-4. Sávpakolás nyújtható téglalapokkal
Tekintsük a sávpakolási feladat azon változatát, amelyben a téglalapok megnyújthatók a terület megváltoztatása nélkül. Adjunk 4-versenyképes algoritmust a feladat megoldására.

27-5. On-line* LPT *algoritmus
Tekintsük az IDŐ modellben azt az algoritmust, amely minden időpontban, amikor egy gép szabaddá válik, azt a munkát ütemezi a megérkezett munkák közül, amelynek a legnagyobb a megmunkálási ideje. Ezt az algoritmust on-line LPT algoritmusnak nevezzük. Igazoljuk, hogy az algoritmus 3/2-versenyképes.

Megjegyzések a fejezethez

Az on-line algoritmusok területéhez kapcsolódó eredmények részletes összefoglalásai találhatóak a [51, 106] könyvekben.

A k-szerver feladatra vonatkozó első eredményeket, a 27.1. és 27.2. tételeket Manasse, McGeoch és Sleator publikálták a [240] cikkben. Az egyenesre, mint speciális metrikus térre vonatkozó eredmény a 27.3. tétel Chrobak, Karloff, Payne és Viswanathan [67] cikkéből származik, később Chrobak és Larmore általánosították az algoritmust fák esetére is [65]. Az első konstans-versenyképes algoritmust az általános feladatra Fiat, Rabani és Ravid [105] fejlesztették ki, a legjobb jelenleg ismert eljárás a munkafüggvényen alapuló $(2k-1)$-versenyképes algoritmus, amelyet Chrobak és Larmore fejlesztett ki az [66] cikkben és Koutsoupias és Papadimitriou elemeztek a [210] cikkben.

A nyugtázás modelljét Dooly, Goldman, és Scott vetették fel a [94] cikkben innen származnak a 27.5. és 27.6. tételek. A feladat egy további modelljét vizsgálja Albers és Bals [15], a feladatra kidolgozott véletlenített algoritmusokat vizsgálja Karlin Kenyon és Randall [187]. A lapletöltési feladat megoldására a Házıúr algoritmust Young [359] dolgozta ki. Az on-line forgalomirányítás területének részletes tárgyalása megtalálható Leonardi [226] összefoglaló cikkében. Az exponenciális algoritmust töltéselosztási modellre Aspnes, Azar, Fiat, Plotkin és Waarts [23] vizsgálja, az itt ismertetett exponenciális algoritmust Awerbuch, Azar és Plotkin alkalmazták a nyereségmaximalizáló modellre a [28] cikkben.

A ládapakolás elméletét tekinti át Csirik és Woeginger [74] cikke. Az NF és FF algoritmusokat a versenyképességi elemzés alapján Johnson, Demers, Ullman, Garey és Graham elemezték a [181, 182] cikkekben, részletesebb elemzés található Johnson [180] doktori értekezésében. Az on-line ládapakolási algoritmusok lehetséges versenyképességi hányadosára vonatkozó alsó korlátokat meghatározó pakolási minták módszerét van Vliet [346, 345] dolgozta ki. Az on-line sávpakolási feladatra az NFS_r algoritmust Baker és Schwarz [34] fejlesztették ki és elemezték. Később egyéb polcok meghatározásán alapuló algoritmusok is kifejlesztésre kerültek, a legkisebb versenyképességi hányadossal rendelkező polcokon alapuló algoritmust az on-line sávpakolási feladatra Csirik és Woeginger [75] fejlesztették ki.

Az on-line ütemezés területén elért eredmények egy részletes összefoglalója található Sgall [308] cikkében. Az első on-line eredmény a LISTA algoritmus elemzése Graham [143] cikkében szerepel, azóta számos eredmény született az azonos gépek esetére, jelenleg a legkisebb versenyképességi hányadossal a Fleischer és Wahl által kifejlesztett, [111] cikkben publikált algoritmus rendelkezik, melynek versenyképességi hányadosa az 1.9201 szám-

hoz tart, ahogy a gépek száma tart a végtelenbe. A hasonló párhuzamos gépek esetén a Monó algoritmus versenyképességi hányadosára alsó korlátot Cho és Sahni [64] adott, a felső korlátot, az általános párhuzamos gépek elemzését és a forgalomirányításnál is használt exponenciális függvényt használó algoritmusok elemzését tartalmazza Aspnes, Azar, Fiat, Plotkin és Waarts [23] cikke. Később további fejlesztésekkel sikerült javítani az exponenciális függvényt használó algoritmusok versenyképességi hányadosán, a kapcsolódó eredmények megtalálhatók Azar [29] cikkében. Az IDŐ modellben ismertetett 27.22. tétel Shmoys, Wein és Williamson [311] cikkén alapul. Az LPT algoritmus elemzése, az IDŐ modellre vonatkozó alsó korlátok és a modellre vonatkozó eredmények részletes tárgyalása megtalálhatóak Vestjens [366] doktori értekezésében. Ebben a fejezetben csak a legalapvetőbb on-line ütemezési modellt ismertettük, ezen modellnek egy érdekes általánosítása, amelyben a gépek száma nem adott, hanem meg kell őket vásárolni, ezt a modellt a [172] és [95] cikkekben vizsgálták.

A 27-1. feladat a [316], a 27-2. feladat a [94], a 27-3. feladat a [355], a 27-4. feladat a [171] és végül a 27-5. feladat a [366] cikken alapul.

A fejezet megírását támogatta az OTKA F048587 számú pályázata.

X. ADATBÁZISKEZELÉS

Bevezetés

Az első kötetben három fejezet (*Adattömörítés, Memóriagazdálkodás* és *Relációs adatmodellek tervezése*) foglalkozott az adatbáziskezelési témával.

Ezeket a fejezeteket most négy további követi.

A 28. fejezet az adatbányászat egyik aktuális területéről, a *gyakori minták kinyeréséről* szól.

Ezt követi az adatbányászat másik aktuális kérdésének, a *klaszterezésnek* a vizsgálata.

A 30. fejezet a *relációs adatbázisok lekérdező algoritmusait* foglalja össze.

Végül a rész utolsó fejezete az utóbbi évtizedben gyorsan terjedő *részben strukturált adatbázisok* algoritmusait mutatja be.

28. Gyakori elemhalmazok keresése

Nagy adatbázisokból a hasznos, rejtett információ feltárása az adatbányászat fő célja. A hagyományos elemzések, mint az alapvető statisztikai értékek meghatározása, vagy a függetlenségvizsgálat, osztályozás, klaszterelemzés mellett egy új igény is megjelent, melynek során az adatbázisban gyakran előforduló elemeket, mintákat kell megkeresni. Egyszerű minták esetében a feladat általában könnyen és gyorsan megoldható megfelelő SQL lekérdezések segítségével. Bonyolult minták esetben a feladat nehezebb, mert vagy nem fogalmazható meg megfelelő lekérdezés, vagy a lekérdezés megválaszolása túl sok erőforrást igényel.

A legkutatottabb gyakori minta kinyerési terület a gyakori elemhalmazok meghatározása. Algoritmusai, fogalmai, ötletei többségét felhasználhatjuk más típusú gyakori minták keresésénél is. Ebben a fejezetben a gyakori elemhalmazok kinyerésének alapjait mutatjuk be.

A kutatási területet egy marketinges igény keltette életre. Óriási áruházakban a gyakran együtt vásárolt termékeket kellett meghatározni. Ezen információ alapján hatékonyabban lehet megszervezni a leárazásokat, akciókat, illetve kialakítani az áruház terméktérképét. A vásárlók viselkedése mellett vizsgálhatjuk a gyakori balesetet okozó helyzeteket, a számítógépes hálózatban gyakran előforduló, riasztással végződő eseménysorozatokat, vagy például azt, hogy az egyes nyomtatott médiumoknak milyen az olvasói összetétele. Amennyiben több magazinnak, újságnak hasonló a célcsoportja, érdemes üzenetünket több helyen is elhelyezni, hogy hatékonyabban ösztönözzük meglevő és potenciális vásárlóinkat. Oldalakon keresztül lehetne sorolni azon példákat, amikor a gyakran előforduló „dolgok” értékes információt rejtenek magukban. A szakirodalomban a dolgokat mintáknak nevezzük, és ***gyakori minták kinyeréséről*** beszélünk.

Vásárlói szokások felderítésénél gyakori *elemhalmazokat* keresünk, ahol a termékek felelnek meg a halmazok elemeinek. Utazásokkal kapcsolatos szokásoknál a gyakran igénybe vett, költséges szolgáltatások sorrendje is fontos, így itt gyakori *sorozatokat* keresünk. Telekommunikációs hálózatokban olyan feltételek (predikátumok) gyakori fennállását kutatjuk, amelyek gyakran eredményeznek riasztást. Ezeket a gyakori *Boole-formulákat* megvizsgálva kaphatjuk meg például a téves riasztások gyakori okait. A böngészési szokások alapján fejleszthetjük oldalaink struktúráját, linkjeit, így a látogatók még gyorsabban és hatékonyabban találják meg a keresett információkat. A böngészés során egy weboldal bejárását *címkézett gyökeres fákkal* jellemezhetjük. Gyakori mintákat kinyerő algoritmusokat a rákkutatásban is alkalmaznak. Azt vizsgálják, hogy a rákkeltő anyagokban vannak-e gyakran előforduló molekula-struktúrák. Ezeket a struktúrákat *címkézett gráfokkal* írjuk le.

A példák arra utalnak, hogy a minta típusa sokféle lehet. Sejthetjük, hogy más technikákat kell majd alkalmazni pl. címkézett gráfok keresésénél, mint amikor csak egyszerű elemhalmazokat kell megtalálnunk. Terjedelmi korlátok miatt a jelen írásban csak elemhalmaz típusú mintákkal foglalkozunk. Bemutatunk három algoritmust, amelyek nemcsak azért tűnnek ki, mert a gyakori elemhalmazok kinyerésnek legmeghatározóbb „szereplői", hanem azért is, mert alapelveik alkalmazhatók, bármilyen típusú mintával van dolgunk. Végül ismertetünk egy sztochasztikus algoritmust, amely mintavételezésen alapszik, így nem biztos, hogy minden gyakori elemhalmazt megtalál, viszont rendkívül gyors.

28.1. Gyakori elemhalmazok keresése

Legyen $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek halmaza és $\mathcal{T} = \langle t_1, \ldots, t_n \rangle$ az $\mathcal{I}$ hatványhalmaza felett értelmezett sorozat, azaz $t_j \subseteq \mathcal{I}$. A $\mathcal{T}$ sorozatot ***bemeneti sorozatnak*** hívjuk, amelynek t_j elemei a ***tranzakciók***. Az $I \subseteq \mathcal{I}$ elemhalmaz ***támogatottsága*** (jelölésben $supp(I)$) megegyezik azon tranzakciók számával, amelyeknek részhalmaza az I. Az I ***gyakori,*** amennyiben támogatottsága nem kisebb egy előre megadott konstansnál, amelyet hagyományosan ***min-supp***-pal jelölünk, és ***támogatottsági küszöbnek*** hívunk. A gyakori elemhalmazok keresése során adott egy $\mathcal{I}$ elemhalmaz, $\mathcal{T}$ bemeneti sorozat, *min-supp* támogatottsági küszöb, feladatunk meghatározni a gyakori elemhalmazokat és azok támogatottságát.

Elterjedt, hogy a támogatottság helyett ***gyakoriságot,*** a támogatottsági küszöb helyett ***gyakorisági küszöböt*** használnak, melyeket $freq(I)$-vel, illetve ***min-freq***-kel jelölnek. Az I elemhalmaz gyakoriságán a $supp(I)/|\mathcal{T}|$ hányadost értjük.

A gyakorlatban előforduló adatbázisokban nem ritka, hogy az elemek száma $10^5 - 10^6$, a tranzakcióké pedig $10^9 - 10^{10}$. Elméletileg már az eredmény kiírása is az $\mathcal{I}$ elemszámában exponenciális lehet, hiszen előfordulhat, hogy $\mathcal{I}$ minden részhalmaza gyakori. A gyakorlatban a maximális méretű gyakori elemhalmaz mérete $|\mathcal{I}|$-nél jóval kisebb (legfeljebb 20-30). Ezen kívül minden tranzakció viszonylag kicsi, azaz $|t_j| \ll |\mathcal{I}|$. A keresési tér tehát nagy, ami azt jelenti, hogy az egyszerű nyers erő módszerek (határozzuk meg minden elemhalmaz támogatottságát, majd válogassuk ki a gyakoriakat) elfogadhatatlanul lassan futnának.

A példákban az elemeket $A, B, C, \ldots$ betűkkel fogjuk jelölni. Az egyszerűség kedvéért a halmazt jelölő kapcsos zárójeleket és az elemeket határoló vesszőket elhagyjuk, tehát például az $\langle\{A, C, D\}, \{B, C, E\}, \{A, B, C, E\}, \{B, E\}, \{A, B, C, E\}\rangle$ sorozatot $\langle ACD, BCE, ABCE, BE, ABCE\rangle$ formában írjuk.

28.1. példa. Legyen $\mathcal{T} = \langle ACD, BCE, ABCE, BE, ABCE\rangle$ bemeneti sorozat. Ekkor például $supp(\emptyset) = 5$, $supp(AC) = 3$, $supp(BE) = 4$. Amennyiben a támogatottsági küszöb 4, akkor a gyakori elemhalmazok a következők: $\emptyset$, B, C, E, BE.

A bemenetet illetően négy adattárolási módot szokás megkülönböztetni. Tegyük fel, hogy minden tranzakcióhoz azonosítót rendelünk, például a bemenetben elfoglalt pozícióját. ***Horizontális adattárolásról*** beszélünk, ha minden tranzakció azonosítóhoz tároljuk a tranzakcióban található elemeket. ***Vertikális adattárolásnál*** minden elemhez tároljuk az elemet tartalmazó tranzakciók azonosítóit. A vertikális tárolás nagy előnye, hogy gyorsan megkaphatjuk egy elemhalmaz támogatottságát.

tranzakció	elemhalmaz
1	C
2	A, B, C

(a) horizontális

elem	tranzakcióhalmaz
A	2
B	2
C	1,2

(b) vertikális

	A	B	C
1	0	0	1
2	1	1	1

(c) mátrixos

tranzakció	elem
1	C
2	A
2	B
2	C

(d) relációs

28.1. ábra. A négyféle tárolási mód.

Ábrázolhatjuk az adatbázist egy $|\mathcal{I}|\times|\mathcal{T}|$ méretű ***bináris mátrixszal*** is. A k-adik sor l-edik eleme 1, amennyiben a k-adik tranzakció tartalmazza az l-edik elemet, ellenkező esetben 0. A $|t_j| \ll |\mathcal{I}|$ feltevés miatt a mátrix ritka, így ez az adattárolási mód az esetek többségében pazarló.

Tudjuk, hogy egy tranzakcióban változó számú elem lehet (és fordítva: egy elem változó számú tranzakcióban szerepelhet). A legtöbb mai adatbázis ***relációs táblákat*** tárol, amelyekben csak rögzített számú attribútum szerepelhet. A valóságban ezért a tranzakciók két attribútummal rendelkező relációs tábla formájában találhatók, ahol az első attribútum a tranzakciót, a második pedig az elemet adja meg (pontosabban a tranzakciók és az elemek azonosítóit). A $\langle C, ABC\rangle$ sorozat négyféle tárolására mutatnak példát a 28.1. táblázatok.

28.1.1. Asszociációs szabályok

A gyakori elemhalmazok kinyerése először az érvényes asszociációs szabályok meghatározásánál került elő. Adott bemeneti sorozat esetén és I, I' nem üres, diszjunkt elemhalmazok esetén az $R : I \xrightarrow{c,s,f} I'$ kifejezést c bizonyosságú, s támogatottságú, f függetlenségi mutatójú ***asszociációs szabálynak*** nevezzük, ahol

$$c = \frac{supp(I \cup I')}{supp(I)} ,$$

$$s = supp(I \cup I') ,$$

$$f = \frac{freq(I \cup I')}{freq(I) \cdot freq(I')} = \frac{c}{freq(I)} .$$

Adott $\mathcal{T}$ bemeneti sorozat, *min-supp* támogatottsági, $min - conf$ bizonyossági, *min-lift* függetlenségi küszöb mellett az $I \xrightarrow{c,s,f} I'$ asszociációs szabály ***érvényes,*** ha $s \geq min\text{-}supp$, $c \geq min - conf$ és $f \geq min\text{-}lift$. Az asszociációs szabályok első alkalmazási területe a vásárlói szokások felderítése volt. Például az „azok akik sört vásárolnak, általában pelenkát is vesznek” állítást kifejezhetjük egy asszociációs szabállyal.

Az asszociációs szabályok kinyerésének feladatában adott egy bemeneti sorozat és három küszöbszám (*min-supp*, $min-conf$, *min-lift*). Feladatunk megtalálni az érvényes asszociációs szabályokat. A feladat hasonlít a gyakori elemhalmazok kinyeréséhez. Az $I \to I'$ szabály érvényességének feltétele, hogy az $I \cup I'$ gyakori elemhalmaz legyen, ahol a gyakorisághoz használt küszöb *min-supp*. Ha ismerjük a gyakori elemhalmazokat, akkor az asszociációs szabályokat könnyen megkaphatjuk: minden legalább kételemű gyakori elemhalmazt kettéosztunk feltétel- és következményrészre, majd ellenőrizzük, hogy az így kapott asszociációs szabály bizonyossága és függetlenségi mutatója nagyobb-e a megadott küszöböknél.

Egy asszociációs szabály azt fejezi ki, hogy azon tranzakciók c-szeresében, amelyben I megtalálható, az I' is megtalálható. A támogatottság adja meg, hogy ez a szabály mennyi tranzakcióban áll fenn. Az f értéke a függetlenséget próbálja megragadni. Amennyiben $f = 1$, akkor ez az jelenti, hogy ugyanolyan gyakran fordul elő I' az I-t tartalmazó tranzakciókban, mint általában a többi tranzakcióban (hiszen $f = 1$ ekvivalens azzal, hogy $freq(I') = freq(I \cup I')/freq(I)$), azaz I nincs hatással (független) az I' előfordulására. Az $f > 1$ egyfajta pozitív korrelációt jelent (I' gyakrabban fordul elő az I-t tartalmazó tranzakciókban, mint általában), $f < 1$ pedig negatívat.

Valójában a függetlenségi mutató nem ragadja meg kellőképpen a két esemény (I és I' előfordulása) statisztikai függetlenségét. Tudjuk, hogy az I, I' események függetlenek, ha $p(I)p(I') = p(I, I')$, amelyet átírhatunk $1 = p(I'|I)/p(I)$ alakra. A jobb oldal annyiban tér el a függetlenségi mutatótól, hogy abban a valószínűségek helyén relatív gyakoriságok szerepelnek. Pusztán a relatív gyakoriságok hányadosa nem elég jó mérték a függetlenség mérésére. Nézzünk például a következő két esetet. Első esetben négy tranzakció van, $supp(I) = 2$, $c = 0.5$, amiből $f = 1$. A másodikban a tranzakciók száma négyezer, $supp(I) = 1992$, $c = 0.504$, amiből $f = 1.012$. Ha csak a függetlenségi mutatókat ismernénk, akkor azt a téves következtetést vonhatnánk le, hogy az első esetben a két esemény függetlenebb, mint a második esetben. Holott érezzük, hogy az első esetben olyan kevés a tranzakció, hogy abból nem tudunk függetlenségre vonatkozó következtetéseket levonni. Minél több tranzakció alapján állítjuk, hogy két elemhalmaz előfordulása összefüggésben van, annál jobban kizárjuk ezen állításunk véletlenségének (esetlegességének) esélyét.

A függetlenség mérése alkalmasabb az ún. χ^2 próbastatisztika. Az $A_1, A_2, \ldots, A_r$ és $B_1, B_2, \ldots, B_s$ két teljes eseményrendszer χ^2 próbastatisztikáját az alábbi képlet adja meg:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{s} \frac{\left(k_{ij} - \dfrac{k_{i.}k_{.j}}{n}\right)^2}{\dfrac{k_{i.}k_{.j}}{n}},$$

ahol k_{ij} az $A_i \cap B_j$ esemény, $k_{i.} = \sum_{j=1}^{s} k_{ij}$ az A_i esemény és $k_{.j} = \sum_{i=1}^{r} k_{ij}$ a B_j esemény bekövetkezésének számát jelöli. Minél kisebb a próbastatisztika, annál inkább függetlenek az események.

A mi esetünkben az egyik eseményrendszer az I elemhalmaz a másik az I' elemhalmaz előfordulásához tartozik, és mindkét eseményrendszernek két eseménye van[1] (előfordul az

[1] Amennyiben mindkét eseményrendszer két eseményből áll, akkor az eredeti képletet módosítani szokás a Yates-féle korrekciós együtthatóval, azaz $\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{2} \left(\left| k_{ij} - \dfrac{k_{i.}k_{.j}}{n} \right| - \frac{1}{2} \right)^2 / \dfrac{k_{i.}k_{.j}}{n}$.

elemhalmaz az adott tranzakcióban, vagy sem). A következő táblázat mutatja, hogy a χ^2 próbastatisztika kiszámításához szükséges értékek közül melyek állnak rendelkezésünkre támogatottság formájában.

	I	nem I	$\sum$
I'	$supp(I \cup I')$		$supp(\mathrm{I}')$
nem I'			
$\sum$	$supp(\mathrm{I})$		$\lvert\mathcal{T}\rvert$

A hiányzó értékeket a táblázat ismert értékei alapján könnyű pótolni lehet, hiszen például $k_{2,1} = supp(I) - supp(I \cup I')$.

A gyakori elemhalmazok első alkalmazásának bemutatása után térjünk rá arra, hogy miként tudjuk a gyakori elemhalmazokat hatékonyan meghatározni.

Gyakorlatok

28.1-1. Legyen $I = I' \cup i'_1 \cup i'_2 \cdots \cup i'_j$, ahol minden i'_k $(1 \le k \le j)$ az I elemhalmaz egy eleme. Mutassuk meg, hogy

$$supp(I) \ge supp(I' \cup i'_1) + supp(I' \cup i'_2) + \cdots + supp(I' \cup i'_j) - (j-1)supp(I') \; .$$

28.1-2. Mennyi az érvényes asszociációs szabályok maximális száma, ha az elemek száma n?

28.2. Gyakori elemhalmazokat kinyerő algoritmusok

A keresési teret úgy képzelhetjük el, mint egy irányított gráfot, amelynek csúcsai az elemhalmazok, és az I_1-ből él indul I_2-be, amennyiben $I_1 \subset I_2$, és $|I_1| + 1 = |I_2|$. A keresési tér bejárásán mindig ezen gráf egy részének bejárását fogjuk érteni. Tehát például a keresési tér szélességi bejárása ezen gráf szélességi bejárását jelenti.

A következőkben bemutatjuk a három leghíresebb gyakori elemhalmazokat kinyerő (GYEK) algoritmust. Mindhárman az üres mintából indulnak ki. Az algoritmusok egy adott fázisában ***jelöltnek*** hívjuk azokat az elemhalmazokat, amelyek támogatottságát meg akarjuk határozni. Az algoritmus akkor ***teljes,*** ha minden gyakori elemhalmazt megtalál és ***helyes,*** ha csak a gyakoriakat találja meg.

Mindhárom algoritmus három lépést ismétel. Először jelölteket állítanak elő, majd meghatározzák a jelöltek támogatottságát, végül kiválogatják a jelöltek közül a gyakoriakat. Természetesen az egyes algoritmusok különböző módon járják be a keresési teret (az összes lehetséges elemhalmazt), állítják elő a jelölteket, és különböző módon határozzák meg a támogatottságokat.

Az általánosság megsértése nélkül feltehetjük, hogy az $\mathcal{I}$ elemein tudunk definiálni egy teljes rendezést, és a jelöltek, illetve a tranzakciók elemeit ezen rendezés szerint tároljuk. Más szóval az elemhalmazokat sorozatokká alakítjuk. Egy sorozat ℓ-elemű *prefixén* a sorozat első ℓ eleméből képzett részsorozatát értjük. A példákban majd, amennyiben a rendezésre nem térünk ki külön, az ábécé szerinti sorrendet használjuk. A GYEK algoritmusok általában érzékenyek a használt rendezésre. Ezért minden algoritmusnál megvizsgáljuk, hogy milyen rendezést célszerű használni annak érdekében, hogy a futási idő, vagy a memóriaszükséglet a lehető legkisebb legyen.

A jelölt-előállítás ***ismétlés nélküli,*** amennyiben nem állítja elő ugyanazt a jelöltet többféle módon. Ez a hatékonyság miatt fontos, ugyanis ismétléses jelölt-előállítás esetében minden jelölt előállítása után ellenőrizni kellene, hogy nem állítottuk-e elő már korábban. Ha ezt nem tesszük, akkor feleslegesen kötünk le erőforrásokat a támogatottság ismételt meghatározásánál. Mindhárom ismertetett algoritmusban a jelöltek előállítása ismétlés nélküli lesz, amit a rendezéssel tudunk garantálni.

Az algoritmusok pszeudokódjaiban GY-vel jelöljük a gyakori elemhalmazok halmazát, J-vel jelöltekét és *j.számláló*-val a j jelölt számlálóját. Az olvashatóbb kódok érdekében feltesszük, hogy minden számláló kezdeti értéke nulla, és az olyan halmazok, amelyeknek nem adunk kezdeti értéket (például GY), nem tartalmaznak kezdetben egyetlen elemet sem.

28.2.1. Az Apriori algoritmus

Az Apriori algoritmus az egyik legelső GYEK algoritmus. Szélességi bejárást valósít meg, ami azt jelenti, hogy a legkisebb mintából (ami az üres halmaz) kiindulva szintenként halad előre a nagyobb méretű gyakori elemhalmazok meghatározásához. A következő szinten (iterációban) az eggyel nagyobb méretű elemhalmazokkal foglalkozik. Az iterációk száma legfeljebb eggyel több, mint a legnagyobb gyakori elemhalmaz mérete.

A jelöltek definiálásánál a következő egyszerű tényt használja fel: *Gyakori elemhalmaz minden részhalmaza gyakori.* Az állítást indirekten nézve elmondhatjuk, hogy egy elemhalmaz biztosan nem gyakori, ha van ritka részhalmaza. Ennek alapján ne legyen jelölt azon elemhalmaz, amelynek van ritka részhalmaza. Az Apriori algoritmus ezért építkezik lentről. Egy adott iterációban csak olyan jelöltet veszünk fel, amelynek összes valódi részhalmazáról tudjuk, hogy gyakori. Az algoritmus onnan kapta a nevét, hogy az ℓ-elemű jelölteket a bemeneti sorozat ℓ-edik átolvasásának megkezdése előtt (a priori) állítja elő. Az ℓ-elemű jelöltek halmazát J_ℓ-lel, az ℓ-elemű gyakori elemhalmazokat pedig GY_ℓ-lel jelöljük.

Apriori($\mathcal{T}$, *min-supp*)

```
1  ℓ ← 0
2  J_ℓ ← {∅}
3  while |J_ℓ| ≠ 0
4        do for minden t ∈ T                          ▷ Támogatottságok meghatározása.
5              do for minden j ∈ J_ℓ, j ⊆ t
6                    do j.számláló ← j.számláló +1
7           for minden j ∈ J_ℓ                        ▷ Gyakori jelöltek kiválogatása.
8              do if j.számláló ≥ min-supp
9                    then GY_ℓ ← GY_ℓ ∪ {j}
10          GY ← GY ∪ GY_ℓ
11          J_{ℓ+1} ← Jelölt-előállítás(GY_ℓ)
12          ℓ ← ℓ + 1
13 return GY
```

A kezdeti értékek beállítása után egy ciklus következik, amely akkor ér véget, ha nincsen egyetlen ℓ-elemű jelölt sem. A cikluson belül először meghatározzuk a jelöltek támogatottságát. Ehhez egyesével vesszük a tranzakciókat (4. sor), és azon jelöltek számlálóját növeljük eggyel, amelyeket tartalmaz a vizsgált tranzakció (5–6. sorok). Ha rendelkezésre

állnak a támogatottságok, akkor a jelöltek közül kiválogathatjuk a gyakoriakat (7–9. sorok). A JELÖLT-ELŐÁLLÍTÁS függvény az ℓ-elemű gyakori elemhalmazokból $(\ell+1)$-elemű jelölteket állít elő. Azok és csak azok az elemhalmazok lesznek jelöltek, amelyek minden részhalmaza gyakori.

A jelöltek előállítása során olyan ℓ-elemű, gyakori I_1, I_2 elemhalmaz párokat keresünk, amelyekre igaz, hogy

- I_1 lexikografikusan megelőzi I_2-t,
- I_1-ből a legnagyobb elem törlésével ugyanazt az elemhalmazt kapjuk, mintha az I_2-ből törölnénk a legnagyobb elemet.

Ha a feltételeknek megfelelő párt találunk, akkor képezzük a pár unióját, majd ellenőrizzük, hogy a kapott elemhalmaznak minden valódi részhalmaza gyakori-e. A támogatottság antimonotonitása miatt szükségtelen az összes valódi részhalmazt megvizsgálni; ha mind az $\ell+1$ darab ℓ-elemű részhalmaz gyakori, akkor az összes valódi részhalmaz is gyakori. Az I_1, I_2 halmazokat a jelölt ***generátorainak*** szokás hívni.

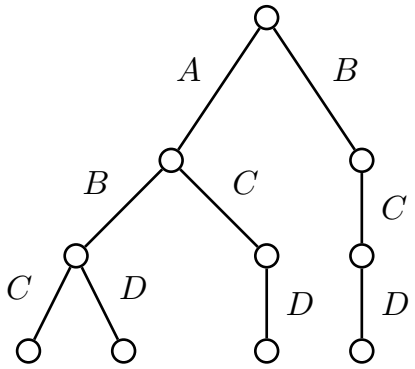
28.2. példa. Legyenek a 3-elemű gyakori elemhalmazok a következők: $GY_3 = \{ABC, ABD, ACD, ACE, BCD\}$. Az ABC és ABD elemhalmazok megfelelnek a feltételnek, ezért képezzük az unió-jukat. Mivel $ABCD$ minden háromelemű részhalmaza a GY_3-nak is eleme, az $ABCD$ jelölt lesz. Az ACD, ACE pár is megfelel a két feltételnek, de uniójuknak van olyan részhalmaza (ADE), amely nem gyakori. Az APRIORI a következő iterációban tehát már csak egyetlen jelölt támogatottságát határozza meg.

A fenti módszer csak akkor alkalmazható, ha $\ell > 0$. Az egyelemű jelöltek előállítása egyszerű: minden egyelemű halmaz jelölt, amennyiben az üres elemhalmaz gyakori ($|\mathcal{T}| \geq$ *min-supp*). Ez összhangban áll azzal, hogy akkor lehet egy elemhalmaz jelölt, ha minden részhalmaza gyakori.

Vizsgáljuk meg részletesebben az 5. sort. Adott egy tranzakció és ℓ-méretű jelöltek egy halmaza. Feladatunk meghatározni azon jelölteket, amelyek a tranzakció részhalmazai. Megoldhatjuk ezt egyszerűen úgy, hogy a jelölteket egyesével vesszük, és eldöntjük, hogy tartalmazza-e őket a tranzakció. Rendezett halmazban rendezett részhalmaz keresése elemi feladat. Ennek az egyszerű módszernek a hátránya, hogy sok jelölt esetén lassú, hiszen annyiszor kell a tranzakció elemein végighaladni, amennyi a jelöltek száma. A gyorsabb működés érdekében a jelölteket szófában célszerű tárolni.

A ***szófa*** éleinek címkéi elemek lesznek. Minden csúcs egy elemhalmazt reprezentál, amelynek elemei a gyökérből a csúcsig vezető út éleinek címkéivel egyeznek meg. Feltehetjük, hogy az egy csúcsból induló élek, továbbá az egy úton található élek címkék szerint rendezve vannak (pl. legnagyobb elem az első helyen). A jelöltek számlálóit a jelöltet reprezentáló levélhez rendeljük. A 28.2.1. ábrán egy szófát láthatunk.

A t tranzakcióban az ℓ-elemű jelölteket úgy találjuk meg, hogy a jelölteket leíró fa gyökeréből kiindulva, rekurzív módon bejárunk bizonyos részfákat. Ha egy d szintű belső csúcshoz a tranzakció j-edik elemén keresztül jutunk, akkor azon élein keresztül lépünk eggyel mélyebb szintre, amelyeknek címkéje megegyezik a tranzakció j'-edik elemével, ahol $j < j' \leq |t| - \ell + d$ (ugyanis $\ell - d$ elemre még szükség van ahhoz, hogy levélbe érjünk). Ha ily módon eljutunk egy ℓ szintű csúcshoz, az azt jelenti, hogy a csúcs által reprezentált elemhalmazt tartalmazza t, így ennek a levélnek a számlálóját kell növelnünk eggyel.



28.2. ábra. Az ABC, ABD, ACD, BCD jelölteket tároló szófa.

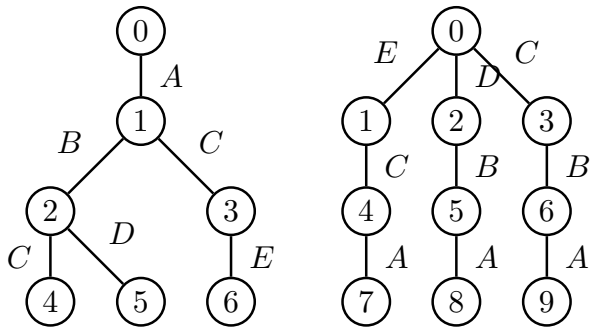
A szófát prefix fának is szokták hívni, ami arra utal, hogy a közös prefixeket csak egyszer tárolja. Ettől lesz gyorsabb a szófás támogatottság-meghatározás a naiv módszernél. A közös prefixeket összevonjuk, és csak egyszer foglalkozunk velük.

A szófa nagy előnye a gyors támogatottság-meghatározás mellett, hogy a jelölt-előállítást is támogatja. Tudjuk, hogy két gyakori elemhalmaz akkor lesz generátor, ha a legnagyobb sorszámú elemük elhagyásával ugyanazt az elemhalmazt kapjuk, vagy más szavakkal, a két gyakori elemhalmaz $\ell - 1$ hosszú prefixei megegyeznek. A támogatottság-meghatározásában használt szófát felhasználhatjuk a következő iterációs lépés jelöltjeinek az előállítására, hiszen a szófa tárolja a jelölt-előállításhoz szükséges gyakori elemhalmazokat.

Az egész algoritmus alatt tehát egyetlen szófát tartunk karban, amely az algoritmus kezdetekor csak egy csúcsból áll (ez reprezentálja az üres halmazt). A támogatottság-meghatározás után töröljük azon leveleket, amelyek számlálója kisebb *min-supp*-nál. Az iterációs lépés végére kialakuló szófa alapján előállítjuk a jelölteket, amely során a szófa új, eggyel mélyebb szinten lévő levelekkel bővül. A jelölt-előállítás során arra is lehetőségünk van, hogy az előző iterációban gyakorinak talált elemhalmazokat és azok számlálóit kiírjuk (a kimenetre vagy a háttértárolóra).

Szükségtelen tárolni azon csúcsokat, amelyekből az összes elérhető levelet töröltük. Ezek ugyanis lassítják a támogatottságok meghatározását (miközben szerepet nem játszanak benne) és feleslegesen foglalják a memóriát.

Gyorsíthatjuk az algoritmust, ha a bemeneti sorozatot nem az eredeti formájában dolgozzuk fel. Általában az adatbázis speciálisan formázott fájlban a háttértáron található. Az operációs rendszer a fájl egy blokkját bemásolja a memóriába és amikor használni akarjuk a tranzakciót, akkor át kell alakítani a feldolgozáshoz megfelelő formátumba (például egész értékeket tartalmazó vektorrá). Ha van elég hely a memóriában, akkor a tranzakció bent



28.3. ábra. Példa: különböző rendezést használó, ugyanazokat az elemhalmazokat tároló szófák.

marad a memóriában, I/O művelet nem történik a következő iterációban, de az átalakítást ismét végre kell hajtani. Ezt az átalakítást megtakaríthatjuk a bemenet explicit tárolásával.

Sajnos a bemenet általában olyan nagy, hogy azt az eredeti formájában nem tudjuk tárolni. Erre nincs is szükség. Csökkenthetjük a memóriaigényt, ha csak a tranzakciók gyakori elemeit tároljuk (megszűrjük a tranzakciót), továbbá, az azonos tranzakciókat egyszer tároljuk, és egy számlálót rendelünk hozzájuk, amely megadja a tranzakció multiplicitását.

A szűrt tranzakciókat célszerű olyan adatstruktúrában tárolni, amelyet gyorsan fel lehet építeni (azaz gyorsan tudjuk beszúrni a szűrt tranzakciókat) és gyorsan végig tudunk menni a beszúrt elemeken. Alkalmazhatunk erre a célra egy szófát, vagy egy piros-fekete fát, amelynek csúcsaiban egy-egy szűrt tranzakció található.

Vizsgáljuk meg, hogy az Apriori mennyire érzékeny az elemeken definiált rendezésre. A jelöltek előállításánál a rendezést csak azért használtuk fel, hogy a jelölt-előállítás ismétlés nélküli legyen. Itt tetszőleges rendezést használhatunk, a kimenet (a jelöltek halmaza) független a rendezéstől. A jelölteket tároló szófa szerkezete azonban már nagyban függ a rendezéstől. Ezt a 28.2.1. ábrával is szemléltethetjük, ahol két olyan szófát láthatunk, amelyek a ABC, ABD, ACE elemhalmazokat tárolják. Az első szófa az ABC szerint csökkenő sorrendet használja ($C \prec B \prec A$), míg a második ennek ellenkezőjét.

Memóriaigény szempontjából az lenne a legkedvezőbb, ha azt a rendezést használnánk, amelyik a legkevesebb csúcsú szófát adná, ugyanis a szófa mérete egyenesen arányos a csúcsainak számával. Sajnos ennek meghatározása nehéz feladat.

28.1. tétel. *Legyen $\mathcal{I}$ egy elemhalmaz, $J \subseteq 2^{\mathcal{I}}$ és k pozitív egész szám. NP-teljes annak eldöntése, hogy van-e olyan rendezése $\mathcal{I}$-nek, amely esetén a J-t tartalmazó szófa csúcsainak száma nem több k-nál.*

A fentiek szerint a minimális méretű szófa meghatározása NP-nehéz feladat. Ennek ellenére egy nagyon egyszerű heurisztika a gyakorlatban rendkívül jól működik. Tudjuk, hogy a szófa az azonos prefixeket egyszer tárolja, azaz minél több a közös prefix, annál kisebb a szófa mérete. Akkor van a legnagyobb esélye annak, hogy két halmaznak közös a prefixe, ha a halmazt úgy alakítjuk át sorozattá, hogy a sorozat elején a leggyakoribb elemek szerepelnek, vagy más szavakkal a választott rendezés a gyakoriság szerint csökkenő

rendezéssel egyezik meg. Tulajdonképpen ennek a heurisztikának a helyességét sejteti az előző ábra is.

Amennyiben nem a kis memóriaigény a fontos, hanem a gyors futás, akkor a fenti heurisztika ellenkezőjét célszerű használni, azaz a teljes rendezés a gyakoriság szerint növekvő rendezéssel egyezik meg. Ebben az esetben a gyökérhez közeli éleken lesznek a ritkább elemek és a levelekhez közel a leggyakoribbak. Ez azt jelenti, hogy a támogatottságmeghatározás során először azt ellenőrizzük, hogy a tranzakció tartalmazza-e a jelölt ritkább elemeit. Ezen a teszten jóval kevesebb jelölt fog átmenni, mintha a gyakori elemek vizsgálatával kezdenénk. Jól szemlélteti ezt az előző ábra, amennyiben a vizsgált tranzakció az $\{ABFGH\}$ halmaz. Gyakoriság szerint csökkenő esetben a 0,1,2 sorszámú csomópontokat fogjuk bejárni a támogatott jelöltek keresés során, míg ellenkező esetben a támogatottság meghatározása már a gyökérben véget ér. A gyors futás érdekében több memóriát használunk.

Futási idő és memóriaigény

A GYEK feladat megadásakor elmondtuk, hogy már az eredmény kiírása – ami a futási időnek a része – az $|I|$-ben exponenciális lehet. A memóriaigényről is hasonló mondható el. Az $(\ell+1)$-elemű jelöltek előállításához szükségünk van az összes ℓ-elemű jelöltre, amelyek száma akár $\binom{|I|}{|I|/2}$ is lehet. Ezek a felső korlátok élesek is, hiszen $min-supp = 0$-nál minden elemhalmaz gyakori.

Az algoritmus indítása előtt tehát nem sokat tudunk mondani a futási időről. A futás során, azonban egyre több információt gyűjtünk, így felmerül a kérdés, hogy ezt fel tudjuke használni az algoritmus maradék futási idejének jóslására. Például, ha a gyakori elemek száma négy, akkor tudjuk, hogy a legnagyobb gyakori elemhalmaz mérete legfeljebb négy (azaz még legfeljebb háromszor olvassuk végig az adatbázist), az összes jelölt maximális száma pedig $\binom{4}{2}+\binom{4}{3}+\binom{4}{4} = 11$. A következőkben megvizsgáljuk, hogy mit tudunk elmondani a jelöltek számáról és a maximális jelöltek méretéről, ha adottak az ℓ-elemű gyakori elemhalmazok (GY_ℓ).

A következő rész fontos fogalma a kanonikus reprezentáció lesz.

28.2. lemma. *Adott n és ℓ pozitív egészek esetében a következő felírás egyértelmű:*

$$n = \binom{m_\ell}{\ell} + \binom{m_{\ell-1}}{\ell-1} + \cdots + \binom{m_r}{r},$$

ahol $r \geq 1$, $m_\ell > m_{\ell-1} > \cdots > m_r$ és $m_j \geq j$ minden $j = r, r+1, \ldots, \ell$ számra.

Ezt a reprezentációt hívják ℓ-kanonikus reprezentációnak. Meghatározása nagyon egyszerű: m_ℓ-nek ki kell elégítenie a $\binom{m_\ell}{\ell} \leq n < \binom{m_\ell+1}{\ell}$ feltételt, $m_{\ell-1}$-nek a $\binom{m_{\ell-1}}{\ell-1} \leq n - \binom{m_\ell}{\ell} < \binom{m_{\ell-1}+1}{\ell-1}$ feltételt, és így tovább, amíg $n - \binom{m_\ell}{\ell} - \binom{m_{\ell-1}}{\ell-1} - \cdots - \binom{m_r}{r}$ nulla nem lesz.

Legyen $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek halmaza és GY_ℓ egy olyan $\mathcal{I}$ feletti halmazcsalád,[2] amelynek minden eleme ℓ-elemű. Az ℓ-nél nagyobb méretű $I \subseteq \mathcal{I}$ halmaz ***fedi*** a GY_ℓ-et, ha I minden ℓ-elemű részhalmaza eleme GY_ℓ-nek. Az összes lehetséges $(\ell+p)$-méretű GY_ℓ-et fedő halmazokból alkotott halmazcsaládot $J_{\ell+p}(GY_\ell)$-lel jelöljük. Nem véletlen, hogy ezt a halmazt ugyanúgy jelöltük, mint az Apriori algoritmus jelöltjeit, ugyanis az $(\ell+p)$-méretű

[2] A H-t az $\mathcal{I}$ feletti ***halmazcsaládnak*** nevezzük, amennyiben $H \subseteq 2^{\mathcal{I}}$.

jelöltek ezen halmazcsaládnak az elemei, és ha az algoritmus során minden jelölt gyakori, akkor az $(\ell + p)$-méretű jelöltek halmaza megegyezik $J_{\ell+p}(GY_\ell)$-lel.

A következő tétel megadja, hogy adott GY_ℓ esetén legfeljebb mennyi lehet a $J_{\ell+p}(GY_\ell)$ elemeinek száma.

28.3. tétel. *Ha*

$$|GY_\ell| = \binom{m_\ell}{\ell} + \binom{m_{\ell-1}}{\ell-1} + \cdots + \binom{m_r}{r}$$

ℓ-kanonikus reprezentáció, akkor

$$|J_{\ell+p}(GY_\ell)| \leq \binom{m_\ell}{\ell+p} + \binom{m_{\ell-1}}{\ell-1+p} + \cdots + \binom{m_s}{s+p},$$

ahol s a legkisebb olyan egész, amelyre $m_s < s + p$. Ha nincs ilyen egész szám, akkor $s = r - 1$.

A fenti tétel a Kruskal–Katona tétel következménye, ezért a tételben szereplő felső korlátot a továbbiakban $KK_\ell^{\ell+p}(|GY_\ell|)$-el jelöljük.

28.4. tétel. *A 28.3. tételben szereplő felső korlát éles, azaz adott n, ℓ, p számokhoz mindig létezik GY_ℓ, amelyre $|GY_\ell| = n$, és $|J_{\ell+p}(GY_\ell)| = KK_\ell^{\ell+p}(|GY_\ell|)$.*

A kanonikus reprezentáció segítségével egyszerű éles felső becslést tudunk adni a legnagyobb jelölt méretére (jelölésben $maxsize(GY_\ell)$) is. Tudjuk, hogy $|GY_\ell| < \binom{m_\ell+1}{\ell}$, ami azt jelenti, hogy nem létezhet olyan jelölt, amelynek mérete nagyobb m_ℓ-nél.

28.5. következmény. *Amennyiben a $|GY_\ell|$ számnak az ℓ-kanonikus reprezentációjában szereplő első tag $\binom{m_\ell}{\ell}$, akkor $maxsize(GY_\ell) \leq m_\ell$.*

Az m_ℓ számot a továbbiakban $\mu_\ell(|GY_\ell|)$-el jelöljük. Ez az érték azt is megmondja, hogy mekkora jelöltméretnél válik nullává a felső korlát, azaz:

28.6. következmény. $\mu_\ell(|GY_\ell|) = \ell + \min\{p \mid KK_\ell^{\ell+p}(|GY_\ell|) = 0\} - 1$

A maradék futási idő jóslására a következő állítás nyújt segítséget.

28.7. következmény. *Az összes lehetséges ℓ-nél nagyobb méretű jelölt száma legfeljebb*

$$KK_\ell^{összes}(|GY_\ell|) = \sum_{p=1}^{\mu_\ell(|GY_\ell|)} KK_\ell^{\ell+p}(|GY_\ell|)\,.$$

A fenti korlátok szépek és egyszerűek, mivel csak két paramétert használnak: az ℓ aktuális méretet és az ℓ-elemű gyakori elemhalmazok számát ($|GY_\ell|$). Ennél jóval többet tudunk. Nem csak a gyakori elemhalmazok számát ismerjük, hanem már pontosan meghatároztuk őket magukat is! Az új információ segítségével számos esetben jobb felső korlátot adhatunk. Például, ha a GY_ℓ-ben csak páronként diszjunkt elemhalmazok vannak, akkor nem állítunk elő jelölteket. A 28.3. tételben szereplő felső korlát azonban jóval nagyobb lehet nullánál. A következőkben bemutatjuk, hogyan lehet a meglévő felső korlátot az ℓ méretű gyakori elemhalmazok *struktúrájára* rekurzívan alkalmazni. Ehhez feltesszük, hogy egy teljes rendezést tudunk definiálni az $\mathcal{I}$ elemein, ami alapján tetszőleges elemhalmaznak meg tudjuk

határozni a legkisebb elemét. Vezessük be a következő két jelölést:

$$GY_\ell^i = \{I - \{i\} \mid I \in GY_\ell, i = \min I\} ,$$

A GY_ℓ^i halmazt úgy kapjuk GY_ℓ-ből, hogy vesszük azon halmazokat, amelyek legkisebb eleme i, majd töröljük ezekből az i elemet.

Ezek után definiálhatjuk a következő rekurzív függvényt tetszőleges $p > 0$-ra:

$$KK^*_{\ell,p}(GY_\ell) = \begin{cases} \binom{|GY_\ell|}{p+1}, & \text{ha } \ell = 1, \\ \min\{KK_\ell^{\ell+p}(|GY_\ell|), \sum_{i\in\mathcal{I}} KK^*_{\ell-1,p}(GY_\ell^i)\}, & \text{ha } \ell > 1 . \end{cases}$$

A definícióból következik, hogy $KK^*_{\ell,p}(GY_\ell) \leq KK_\ell^{\ell+p}(|GY_\ell|)$, továbbá

28.8. tétel. $|J_{\ell+p}(GY_\ell)| \leq KK^*_{\ell,p}(GY_\ell)$.

Bizonyítás. A bizonyítás teljes indukción alapul, az $\ell = 1$ eset triviális. Tulajdonképpen csak az kell belátni, hogy

$$|J_{\ell+p}(GY_\ell)| \leq \sum_{i\in\mathcal{I}} KK^*_{\ell-1,p}(GY_\ell^i) .$$

Az egyszerűség kedvéért vezessük be a következő jelölést: $H \cup i = \{I \cup \{i\} \mid I \in H\}$, ahol H egy $\mathcal{I}$ feletti halmazcsalád. Vegyük észre, hogy $GY_\ell = \sum_{i\in\mathcal{I}} GY_\ell^i \cup i$ és $GY_\ell^i \cap GY_\ell^j = \emptyset$ minden $i \neq j$ elempárra. Azaz a GY_ℓ halmazcsalád egy partícióját képeztük.

Amennyiben $I \in J_{\ell+p}(GY_\ell)$, és I-nek legkisebb eleme i, akkor $I \setminus \{i\} \in J_{\ell-1+p}(GY_\ell^i)$, hiszen $I \setminus \{i\}$ minden $(\ell-1)$-elemű részhalmaza GY_ℓ^i-beli. Ebből következik, hogy

$$J_{\ell+p}(GY_\ell) \subseteq \bigcup_{i\in\mathcal{I}} J_{\ell-1+p}(GY_\ell^i) \cup i .$$

Abból, hogy az GY_ℓ^i halmazcsaládok páronként diszjunktak következik, hogy $J_{\ell-1+p}(GY_\ell^i) \cup i$ is páronként diszjunkt halmazcsaládok. Ebből következik az állítás, hiszen:

$$\begin{aligned} |J_{\ell+p}(GY_\ell)| &\leq |\bigcup_{i\in\mathcal{I}} J_{\ell-1+p}(GY_\ell^i) \cup i| \\ &= \sum_{i\in\mathcal{I}} |J_{\ell-1+p}(GY_\ell^i) \cup i| \\ &= \sum_{i\in\mathcal{I}} |J_{\ell-1+p}(GY_\ell^i)| \\ &\leq \sum_{i\in\mathcal{I}} KK^*_{\ell-1,p}(GY_\ell^i) , \end{aligned}$$

ahol az utolsó egyenlőtlenségnél az indukciós feltevést használtuk. ■

A páronként diszjunkt halmazok esete jó példa arra, hogy a minimum kifejezésben szereplő második tag kisebb lehet az elsőnél. Előfordulhat azonban az ellenkező eset is.

Például legyen $GY_2 = \{AB, AC\}$. Könnyű ellenőrizni, hogy $KK_2^3(|GY_2|) = 0$, ugyanakkor a második tagban szereplő összeg 1-et ad. Nem tudhatjuk, hogy melyik érték a kisebb, így jogos a két érték minimumát venni.

Javíthatjuk a legnagyobb jelölt méretére, illetve az összes jelölt számára vonatkozó felső korlátokat is. Legyen $\mu_\ell^*(GY_\ell) = \ell + \min\{p | KK_{\ell+p}^*(GY_\ell) = 0\} - 1$ és

$$KK_{\text{összes}}^*(GY_\ell) = \sum_{p=1}^{\mu_\ell^*(GY_\ell)} KK_{\ell+p}^*(GY_\ell) \ .$$

28.9. következmény. *maxsize*$(GY_\ell) \leq \mu_\ell^*(GY_\ell) \leq \mu_\ell(|GY_\ell|)$.

28.10. következmény. *Az összes lehetséges ℓ-nél nagyobb méretű jelölt száma legfeljebb $KK_{összes}^*(GY_\ell)$ lehet, és $KK_{összes}^*(GY_\ell) \leq KK_\ell^{összes}(|GY_\ell|)$.*

A KK^* érték függ a rendezéstől. Például a $KK_{2,1}^*(\{AB, AC\})$ értéke 1, amennyiben a rendezés szerinti legkisebb elem A, és 0 bármely más esetben. Elméletileg meghatározhatjuk az összes rendezés szerinti felső korlátot, és kiválaszthatjuk azt, amelyik a legkisebb értéket adja. Ez a megoldás azonban túl sok időbe telne. A szófa által használt rendezés szerinti felső korlátot viszonylag könnyen meghatározhatjuk. Ehhez azt kell látnunk, hogy a gyökér i címkéjű éléhez tartozó részfa levelei reprezentálják a GY_ℓ^i elemeit. A szófa egyetlen bejárásával egy egyszerű rekurzív módszer segítségével minden csúcshoz kiszámíthatjuk a $KK_{\ell-d,p}^*(GY_{\ell-d}^I)$ és $KK_{\ell-d}^{\ell-d+p}(|GY_{\ell-d}^I|)$ értékeket, ahol d a csúcs mélységét jelöli, $GY_{\ell-d}^I$ pedig az adott csúcshoz tartozó részfa által reprezentált elemhalmazokat. A gyökérhez kiszámított két érték adja meg a KK és KK^* korlátokat.

Ha a maradék futási idő becslésére kívánjuk használni a fenti felső korlátot, akkor tudnunk kell, hogy a jelöltek támogatottságának meghatározása függ az Apriori algoritmusban felhasznált adatstruktúrától. Szófa esetében például egy jelölt előfordulásának meghatározásához el kell jutnunk a jelöltet reprezentáló levélhez, ami a jelölt méretével arányos lépésszámú műveletet igényel. A maradék futási idő pontosabb felső becsléséhez a $KK_{\ell+p}^*(GY_\ell)$ értékeket súlyozni kell $(\ell + p)$-vel.

28.2.2. Az Eclat algoritmus

Az Eclat az üres mintából indulva egy rekurzív, mélységi jellegű bejárást valósít meg. A rekurzió mélysége legfeljebb eggyel több, mint a legnagyobb gyakori elemhalmaz mérete. Az Apriori-val szemben mindig egyetlen jelöltet állít elő, majd ennek azonnal meghatározza a támogatottságát. Az $(\ell + 1)$-elemű, P prefixű jelölteket, ahol $|P| = \ell - 1$ az ℓ-elemű, P prefixű gyakori elemhalmazokból állítja elő egyszerű páronkénti unióképzéssel.

Az algoritmus központi fogalma az ún. TID-halmaz. Egy elemhalmaz ***TID-halmazának*** (Transaction IDentifier) elemei azon bemeneti sorozatok azonosítói (sorszámai), amelyek tartalmazzák az adott elemhalmazt. Más szóval egy TID-halmaz a vertikális adatbázis egy megfelelő sora. Például $\langle AD, AC, ABCD, B, AD, ABD, D\rangle$ bemenet esetén az $\{A, C\}$ elemhalmaz TID-halmaza $\{1, 2\}$, amennyiben egy tranzakció azonosítója megegyezik a bemeneti sorozatban elfoglalt helyével, és a helyek számozását nullától kezdjük.

A TID-halmaz két fontos tulajdonsággal bír:

1. Az I elemhalmaz TID-halmazának mérete megadja az I támogatottságát.

2. Egy jelölt TID-halmazát megkaphatjuk a generátorainak TID-halmazaiból egy egyszerű metszetképzéssel.

Az Eclat pszeudokódja az alábbi.

Eclat($\mathcal{T}, min\text{-}supp$)

```
1  for minden t ∈ T                          ▷ Elemek támogatottságának meghatározása.
2      do for minden i ∈ t
3             do J_1 ← J_1 ∪ {i}
4                i.számláló ← i.számláló + 1
5  for minden j ∈ J_1                        ▷ Gyakori elemek meghatározása.
6      do if j.számláló ≥ min-supp
7           then GY_1 ← GY_1 ∪ {j}
8  for i ← 1 to |T|                          ▷ Gyakori elemek TID-halmazainak felépítése.
9      do for minden j ∈ t_i ∩ GY_1
10            do j.TID ← j.TID ∪ {i}
11 return GY_1 ∪ Eclat-segéd(GY_1, ∅, min-supp)
```

Először meghatározzuk a gyakori elemeket, majd felépítjük a gyakori elemek TID-halmazait. A későbbiekben nem használjuk a bemenetet, csak a TID-halmazokat. Az algoritmus lényege a Eclat-segéd rekurziós eljárás. Jelöljük a P prefixű, P-nél eggyel nagyobb méretű gyakori elemhalmazokból alkotott halmazcsaládot GY^P-vel. Nyilvánvaló, hogy $GY^{\emptyset} = GY_1$.

Eclat-segéd($GY^P, P, min\text{-}supp$)

```
1  for minden gy ∈ GY^P
2      do for minden gy' ∈ GY^P, gy ≺ gy'
3             do j ← gy ∪ gy'
4                j.TID ← gy.TID ∩ gy'.TID
6                if |j.TID| ≥ min-supp
7                  then GY^gy ← GY^gy ∪ {j}
8        if |GY^gy| ≥ 2
9          then GY ← GY ∪ GY^gy ∪ Eclat-segéd(GY^gy, gy, min-supp)
10         else GY ← GY ∪ GY^gy
11 return GY
```

Az Eclat jelölt-előállítása megegyezik az Apriori jelölt-előállításával, azzal a különbséggel, hogy nem ellenőrizzük az unióképzéssel kapott halmaznak minden részhalmazára, hogy gyakori-e (a mélységi bejárás miatt ez az információ nem is áll rendelkezésünkre). Látható, hogy az Eclat abban is különbözik az Apriori-tól, hogy egy jelölt előállítása után azonnal meghatározza a támogatottságát, mielőtt újabb jelöltet állítana elő. Nézzünk egy példát a keresési tér bejárására.

28.3. példa. Legyen $\mathcal{T} = \langle ACDE, ACG, AFGM, DM \rangle$ és *min-supp* $= 2$. Első lépésben meghatározzuk a gyakori elemeket: A, C, D, G, M, ami nem más, mint GY^0. Ezután előállítjuk és azonnal meg is határozzuk az (A, C), (A, D), (A, G), (A, M) párok unióját. Ezek közül csak az AC, AG halmazok gyakoriak. A következő rekurziós lépésben ennek a két halmaznak vesszük az unióját, állítjuk elő a TID-halmazát, amely alapján kiderül, hogy az ACG ritka, és a rekurzió ezen ága véget ér. Ezután a C elemnek vesszük az unióját a sorban utána következő elemekkel egyesével és így tovább.

Látnunk kell, hogy az Eclat legalább annyi jelöltet állít elő, mint az Apriori. A mélységi bejárás miatt ugyanis egy jelölt előállításánál nem áll rendelkezésünkre az összes részhalmaz. Az előző példa esetében például az {A,C,G} támogatottságát hamarabb vizsgálja, mint a {C,G} halmazét, holott ez utóbbi akár ritka is lehet. Ebben a tekintetben tehát az Eclat rosszabb az Apriori-nál, ugyanis több lesz a ritka jelölt.

Az Eclat igazi ereje a jelöltek támogatottságának meghatározásában van. A jelöltek TID-halmazainak előállítása egy rendkívül egyszerű és nagyon gyors művelet lesz. Emellett ahogy haladunk egyre mélyebbre a mélységi bejárás során, úgy csökken a TID-halmazok mérete, és ezzel a támogatottság meghatározásának ideje is. Ezzel szemben az Apriori-nál ahogy haladunk az egyre nagyobb méretű jelöltek felé, úgy nő a szófa mélysége, és lesz egyre lassabb minden egyes jelölt támogatottságának meghatározása.

A keresési tér bejárása függ a prefix definíciójától, amit az elemeken definiált rendezés határoz meg. Melyek lesznek azok a jelöltek, amelyek az Apriori-ban nem lennének jelöltek (tehát biztosan ritkák), illetve várhatóan melyik az a rendezés, amely a legkevesebb ilyen tulajdonságú halmazt adja? Ha egy elemhalmaz jelölt az Eclat algoritmusban, de az Apriori-ban nem, akkor van olyan részhalmaza, amely ritka. Amennyiben feltételezzük, hogy az elemek függetlenek, akkor azon részhalmaz előfordulásának lesz legkisebb a valószínűsége (és ezzel együtt az esélye annak, hogy ritka), amely a leggyakoribb elemet nem tartalmazza. A jelölt prefixe generátor, tehát gyakori, így akkor lesz a legnagyobb esélye annak, hogy minden részhalmaz gyakori, ha a prefix a leggyakoribb elemet nem tartalmazza. Az Eclat algoritmusnál a legkevesebb ritka jelöltet és így a legjobb futási időt tehát a gyakoriság szerint csökkenő rendezéstől várhatjuk.

28.4. példa. Ennek a gondolatmenetnek az illusztrálására nézzük a következő példát. Legyenek gyakori halmazok a következők: A, B, C, D, AB, AC, BC, AD, ABC, továbbá $supp(A) < supp(B) < supp(C) < supp(D)$. Amennyiben az Eclat algoritmus a gyakoriság szerint növekvő sorrendet használja, akkor az előállítás sorrendjében a következő halmazok lesznek jelöltek: A, B, C, D, AB, AC, AD, ABC, ABD, ACD, BC, BD, CD. Ugyanez gyakoriság szerint csökkenő sorrendnél D, C, B, A, DC, DB, DA, CB, CA, CBA, BA. Az utóbbi esetben tehát négy ritka jelölt helyett (ABD, ACD, BD, CD) csak kettő lesz (CD, BD). Megjegyezzük, hogy ez a két elemhalmaz az Apriori esetében is jelölt lesz. A gyakoriság szerint csökkenő esetben egyszer állítunk elő olyan háromelemű jelöltet, amelynek van olyan kételemű részhalmaza, amelyet nem vizsgáltunk. Ez a jelölt a CBA és a nem megvizsgált részhalmaz a BA. Mivel a részhalmaz éppen a leggyakoribb elemeket tárolja, ezért van nagy esélye annak, hogy gyakori (főleg ha hozzávesszük, hogy a jelölt két generátora, CB és CA is gyakori).

28.2.3. Az FP-GROWTH algoritmus

Az FP-GROWTH algoritmus[3] egy mélységi jellegű, rekurzív algoritmus, a keresési tér bejárása tekintetében megegyezik az ECLAT-tal. A támogatottságok meghatározását az egyelemű gyakori halmazok meghatározásával, majd a bemenet ***szűrésével*** és ***vetítésével*** valósítja meg rekurzív módon. A bemenet szűrése azt jelenti, hogy az egyes tranzakciókból töröljük a bennük előforduló ritka elemeket. A $\mathcal{T}$ elemhalmaz P elemhalmazra vetítését (jelölésben $\mathcal{T}|P$) pedig úgy kapjuk, hogy vesszük a P-t tartalmazó tranzakciókat, majd töröljük belőlük a P-t. Például $\langle ACD, BCE, ABCE, BE, ABCE\rangle|B = \langle CE, ACE, E, ACE\rangle$. Az algoritmus pszeudokódja a következőkben olvasható.

FP-GROWTH($\mathcal{T}$, *min-supp*)

```
1  FP-GROWTH-SEGÉD(T, min-supp, ∅)
```

A segédeljárás harmadik paramétere (P) egy prefix elemhalmaz, az első paraméter pedig az eredeti bemenet P-re vetítése. Az eredeti bemenet $\emptyset$-ra vetítése megegyezik önmagával.

FP-GROWTH-SEGÉD(T, *min-supp*, P)

```
 1  for minden t ∈ T                                  ▷ A tranzakciókban előforduló
 2      do for minden i ∈ t                           ▷ elemek támogatottságának meghatározása.
 3          do J_1 ← {i}
 4             i.számláló ← i.számláló + 1
 5  for minden j ∈ J_1                                ▷ Gyakori elemek kiválogatása.
 6      do if j.számláló ≥ min-supp
 7             then GY_1 ← GY_1 ∪ {j}
 8  T* ← SZŰRÉS(T, GY_1)
 9  for minden gy ∈ GY_1
10      do T*|gy ← VETÍTÉS(T*, gy)
11         GY ← GY ∪ {P ∪ {gy}}
12         GY ← GY ∪ FP-GROWTH(T*|gy, min-supp, P ∪ {gy})
13         T* ← TÖRLÉS(T*, gy)
14  return GY
```

Egy rekurziós lépés három fő lépésből áll. Először meghatározzuk azon elemek támogatottságát, amelyek előfordulnak valamelyik tranzakcióban (1–4. sorok). Ezekből kiválasztjuk a gyakoriakat (5–7. sorok). Ezután minden gy gyakori elemet egyesével veszünk (9. sor). Meghatározzuk a gy-hez tartozó vetített bemenetet, majd meghívjuk az algoritmust rekurzívan a $\mathcal{T}|gy$ bemenetre. Törölnünk kell a gy elemet a $\mathcal{T}^*$-beli tranzakciók elemei közül (13. sor) annak érdekében, hogy egy jelöltet csak egyszer állítsunk elő.

A jelöltek előállításának tekintetében az FP-GROWTH algoritmus a legegyszerűbb. Ha az I elemhalmaz gyakori, akkor a következő rekurziós szinten azon $I \cup j$ halmazok lesznek a

[3] Az FP a **F**requent **P**attern rövidítése, ami miatt az algoritmust ***mintanövelő*** algoritmusnak is hívják. Ez az elnevezés azonban félrevezető, ugyanis szinte az összes GYEK algoritmus mintanövelő abban az értelemben, hogy egy új jelölt a generátorainak egyelemű bővítése, vagy más szóval növelése. Az FP-GROWTH sajátsága nem a jelöltek előállítása, hanem a jelöltek támogatottság-meghatározásának módja.

jelöltek, ahol j az I-re vetített bemenetben előforduló elem és $I \cup j$ nem volt jelölt korábban. Tulajdonképpen az FP-GROWTH a nagy elemszámú jelöltek támogatottságának meghatározását visszavezeti három egyszerű műveletre: egyelemű gyakori elemhalmazok kiválogatása, szűrés és vetített bemenet előállítása.

A 9. sorban egyesével vesszük a gyakori elemeket. Ezt valamilyen rendezés szerint kell tennünk és ez a rendezés határozza meg, hogy milyen sorban járjuk be a keresési teret, milyen vetített bemeneteket állítunk elő és mely elemhalmazok lesznek a hamis jelöltek. Az ECLAT-nál elmondottak itt is élnek; várhatóan abban az esetben lesz a hamis jelöltek száma minimális, amennyiben a prefixben a legritkább elemek vannak, azaz a 9. sorban gyakoriság szerint növekvő sorban vesszük az elemeket.

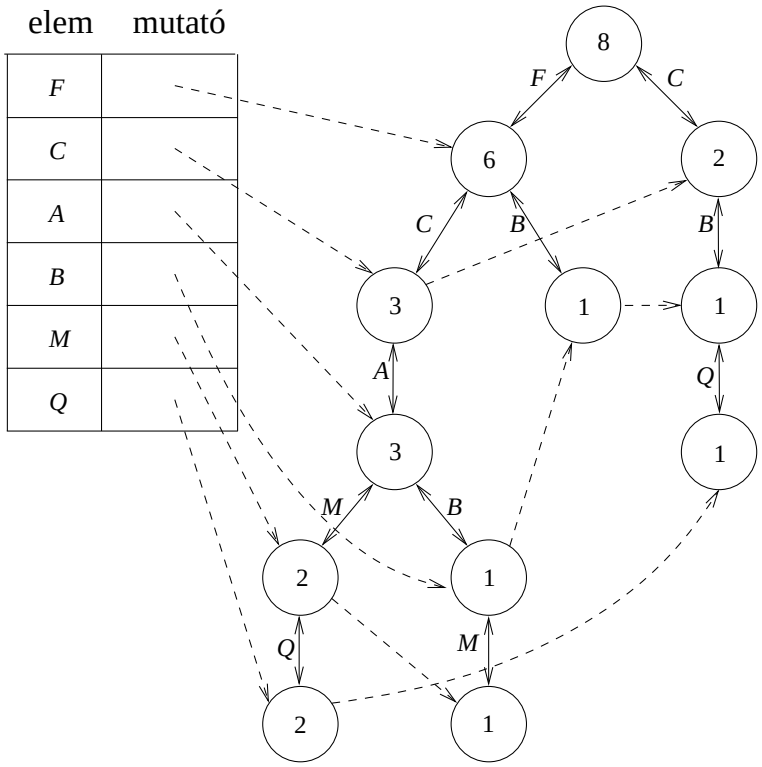
Az FP-GROWTH algoritmus szerves része az ***FP-fa,*** amelyben a szűrt bemenetet tároljuk. Az FP-fa segítségével könnyen előállíthatjuk a vetített bemeneteket, azokban könnyen meghatározhatjuk az elemek támogatottságát, amiből előállíthatjuk a vetített, majd szűrt bemenetet. Ezt a vetített és szűrt bemenetet szintén egy FP-fában tároljuk, amelyet ***vetített FP-fának*** hívunk.

Az FP-fa egy keresztélekkel és egy fejléc táblával kibővített szófa. Az élek címkéi gyakori elemek. Az egyszerűbb leírás kedvéért egy (nemgyökér) csúcs címkéjén a csúcsba mutató él címkéjét értjük. Minden csúcs egy elemhalmazt reprezentál, amelynek elemei a gyökérből a csúcsig vezető út csúcsainak címkéivel egyeznek meg. Minden csúcshoz egy számlálót rendelünk. Ez a számláló adja meg, hogy a csúcs által reprezentált halmaz mennyi bemeneti (vagy vetített) elemhalmaznak a prefixe. Az azonos címkéjű csúcsok láncolt listaszerűen össze vannak kötve keresztirányú élekkel. A lánc legelső elemére mutat a fejléctáblának az adott eleméhez tartozó mutatója.

28.5. példa. Tegyük fel, hogy bemenetként a $\langle ACDFMQ, ABCFMO, BFO, BCKSQ, ACFMQ, CS, DFJ, FHI\rangle$ sorozat van adva, és $min\text{-}supp = 3$. A gyakori elemek: A, B, C, F, M, Q, amelyek támogatottsága rendre 3, 3, 5, 6, 3, 3. Ekkor a szűrt bemenetet ($\langle ACFMQ, ABCFM, BF, BCQ, ACFMQ, C, F, F\rangle$) reprezentáló FP-fa, amely gyakoriság szerint csökkenő sorrendet ($Q \prec M \prec B \prec A \prec C \prec F$) használ, a 28.4. ábrán látható

Egy FP-fát hasonló módon építünk fel, mint egy szófát. Különbség, hogy egy I elemhalmaz beszúrásánál nem csak az I-t reprezentáló levélnek a számlálóját növeljük eggyel, hanem minden olyan csúcsét, amelyet érintünk a beszúrás során (hiszen ezen csúcsokat reprezentáló halmazok az I prefixei). A keresztirányú éleket és a fejléctáblát is egyszerűen megkaphatjuk. Legyen a fejléctábla mutatóinak kezdeti értéke NIL. Amikor beszúrunk egy új, i címkéjű csúcsot, akkor két dolgot kell tennünk. Az új csúcs keresztél mutatója felveszi a fejléctábla i-hez tartozó bejegyzését, majd ezt a bejegyzést az új csúcs címére cseréljük. Ezzel tulajdonképpen olyan láncot készítünk, amelyben a csúcsok a beszúrási idejük szerint csökkenően vannak rendezve (az először beszúrt elem van leghátul) és a lista a fejléctáblában kezdődik.

Az FP-fa mérete – hasonlóan a szófa méretéhez – függ az elemeken definiált rendezéstől. Az FP-GROWTH algoritmus akkor lesz hatékony, ha a fa elfér a memóriában, ezért fontos lenne azt a rendezést használni, ami várhatóan a legkisebb fát eredményezi. Az APRIORI esetében már elmondtuk, hogy az a heurisztika, amely az elemek gyakoriság szerint csökkenő rendezését használja, általában kis méretű fát eredményez.



28.4. ábra. Az $\langle ACFMQ, ABCFM, BF, BCQ, ACFMQ, C, F, F\rangle$ szűrt bemenetet tároló FP-fa.

Egyszerű lesz a vetített bemenet előállítása és a szűrt bemenetből egy elem törlése, amennyiben a legritkább gyakori elemet (gy_r) vesszük először. Ez összhangban áll azzal, hogy a pszeudokód 9. sorában az elemeket gyakoriság szerint növekvő sorrendben vesszük. A gy_r csak levél címkéje lehet. Mivel a fából törölni fogjuk a gy_r címkéjű csúcsokat a rekurziós művelet után (13. sor), a következő elem is csak levél címkéje lesz.

Nézzük most meg, hogy amennyiben a szűrt bemenet egy FP-fában van tárolva, akkor hogyan kaphatjuk meg a gy_r elemre vett vetítésben az elemek támogatottságát. A fejléctábla gy_r eleméhez tartozó mutatóból kiindulva a keresztélek alkotta láncban pontosan azok a csúcsok vannak, amelyek gy_r-t tartalmazó bemeneti elemet reprezentálnak. Az egyes elemhalmazok előfordulását a gy_r címkéjű csúcsokhoz rendelt számláló adja meg, az elemeiket pedig a gyökérig felsétálva kaphatjuk. A lista utolsó csúcsának feldolgozása után rendelkezésünkre állnak a gy_r elemhez tartozó vetített bemenetben valahol előforduló elemek támogatottságai, amely alapján kiválogathatjuk a vetített bemenetben gyakori elemeket.

Ugyanilyen bejárással kaphatjuk meg a vetített, majd szűrt bemenetet tartalmazó FP-fát. A fejléctáblából kiindulva végigmegyünk a láncolt lista elemein. A csúcs által reprezentált elemhalmazból töröljük a ritka elemeket, majd a kapott elemhalmazt beszúrjuk az új FP-fába. A kis memóriaigény érdekében a gyakoriság szerint csökkenő sorrendet használjuk.

Ezt a sorrendet a vetített bemenet alapján állítjuk fel (lévén az új fa a vetített és szűrt bemenetet fogja tárolni), ami különbözhet az eredeti FP-fában alkalmazott rendezéstől.

28.6. példa. Folytassuk az előző példát és állítsuk elő a legritkább gyakori elemhez (Q) tartozó vetített és szűrt bemenetet. A fejléctábla Q eleméhez tartozó mutatóból kiindulva mindössze két csúcsot látogatunk meg, ami azt jelenti, hogy a vetített bemenet két különböző elemhalmazt tartalmaz: az $FCAM$-et kétszer, a CB-t egyszer. Ez alapján a vetített bemenetben egyetlen gyakori elem van, C. Ez a rekurziós ág nem folytatódik, hanem visszatér a QC gyakori elemhalmazzal. Az FP-fából törölhetjük a fejléctábla Q bejegyzéséhez tartozó mutatóból, keresztirányú élek segítségével elérhető csúcsokat. A következő vizsgált elem az M. Az M vetített bemenetében három gyakori elem van, és a vetített szűrt bemenet az FCA elemhalmazt tartalmazza háromszor. Ezt a vetített, szűrt bemenetet egy egyetlen útból álló FP-fa fogja reprezentálni. A többi FP-fa ugyanilyen egyszerűen megkapható.

Hatékonysági szempontból rendkívül fontos, hogy a rekurziót ne folytassuk, ha a vizsgált FP-fa egyetlen útból áll. A rekurzió helyett képezzük inkább az út által reprezentált elemhalmaz minden részhalmazát. A részhalmaz támogatottságát annak a csúcsnak a számlálója adja meg, amely a legmélyebben van a részhalmazt meghatározó csúcsok között.

Hasonlítsuk össze a három algoritmust. A ritka jelöltek számát vizsgálva az Apriori a legjobb, vagy más szavakkal, az Apriori tölti a legkevesebb idő felesleges elemhalmazok támogatottságának meghatározásával. A támogatottságok meghatározásának tekintetében az Apriori az esetek többségében nem tud labdába rúgni a két riválisával szemben. Ennek oka az , hogy az Apriori semmilyen már kinyert tudást nem használ fel az új jelöltek támogatottságának meghatározása során. Ezzel szemben a másik két algoritmus ügyesen kihasználja a prefixek előfordulásáról megszerzett tudást. Az Eclat-nál rendelkezésre állnak a jelölt generátorainak TID-halmazai, az FP-growth esetében pedig a prefixre vetített bemenet.

28.2.4. Toivonen mintavételező algoritmusa

Az adatbányászati alkalmazásokban a feldolgozandó adathalmaz mérete óriási lehet, akár több terabájt nagyságú is. Habár a rendelkezésre álló erőforrások (memória, processzor teljesítmény) is egyre nagyobbak, az algoritmusok nem mindig adnak elfogadható időn belül megoldást. Sokszor a gyors válaszidő fontosabb, mint hogy minden gyakori elemhalmazt megtaláljunk. Ilyen esetekben mintavételező algoritmusokat célszerű használni.

Tegyük fel, hogy rendelkezésünkre áll a bemeneti sorozatnak egy akkora – nem feltétlenül összefüggő – részsorozata ($\mathcal{T}'$), amelyet könnyen fel tudunk dolgozni valamely, az előzőekben bemutatott algoritmusnak a segítségével. Jelöljük a $\mathcal{T}'$-ben gyakori elemhalmazok halmazát GY'-vel.

A GY' elemei nem feltétlenül egyeznek meg az eredeti sorozatban gyakori elemhalmazokkal. Ennek két oka van. Egyfelől lehet, hogy a $\mathcal{T}'$-ben gyakori elemhalmaz valójában ritka. Másfelől a $\mathcal{T}'$-ben ritka elemhalmaz lehet, hogy valójában gyakori.

Az első típusú hibát könnyű kiküszöbölni: tekintsük a GY' elemeit jelölteknek, és határozzuk meg a támogatottságokat most már a $\mathcal{T}$-ben. A hatékonyság miatt célszerű az elemhalmazokat az Apriori algoritmushoz hasonlóan egy szófában tárolni. Különbség lesz, hogy ekkor a szófában egyszerre vannak jelen különböző méretű elemhalmazok, így számlálót nem csak a levelekhez kell rendelni, hanem a belső pontokhoz is.

A második típusú hibát nem lehet ilyen egyszerűen kiküszöbölni. A hiba elkövetésének esélyét csökkenthetjük, ha csökkentjük *min-freq*-et. Ez azt a veszélyt rejti magában, hogy túl

sok ritka elemhalmaz lesz gyakori a részsorozatban. A másik, talán még nagyobb probléma, hogy nem lehetünk biztosak abban, hogy minden gyakori elemhalmazt megtaláltunk.

Toivonen ezért a *min-freq* csökkentése helyett a következőt javasolta. Ne csak a GY' elemeinek támogatottságát határozzuk meg $\mathcal{T}$-ben, hanem az $NB(GY')$ elemeiét is. Egy elemhalmaz eleme $NB(GY')$-nek, amennyiben nem eleme GY'-nek, de minden valódi részhalmaza benne van GY'-ben.[4]

28.7. példa. Legyen $\mathcal{I} = \{A, B, C, D, E, F\}$ és $GY' = \{A, B, C, F, AB, AC, AF, CF, ACF\}$. Ekkor $NB(GY') = \{BC, BF, D, E\}$.

A következő állítás szerint bizonyos esetekben biztosak lehetünk abban, hogy nem veszítünk el jelöltet.

28.11. állítás. *Legyen $\mathcal{T}'$ a $\mathcal{T}$ bemeneti sorozat egy része. Jelöljük GY-vel a $\mathcal{T}$-ben, GY'-vel az $\mathcal{T}'$-ben gyakori elemhalmazokat és GY^*-gal azokat az $\mathcal{T}$-ben gyakori elemhalmazokat, amelyek benne vannak $GY' \cup NB(GY')$-ben ($GY^* = GY \cap (GY' \cup NB(GY'))$). Amennyiben*

$$NB(GY^*) \subseteq GY' \cup NB(GY')$$

teljesül, akkor $\mathcal{T}$-ben a gyakori elemhalmazok halmaza pontosan a GY^*, tehát $GY^* \equiv GY$.

Bizonyítás. Indirekt tegyük fel, hogy létezik $I \in GY$, de $I \notin GY^*$, és a feltétel teljesül. Tekintsünk egy legkisebb méretű $I' \subseteq I$-t, amire $I' \in GY$ és $I' \notin GY^*$ (ilyen I'-nek kell lennie, ha más nem, ez maga az I halmaz). A GY^* definíciója miatt ekkor $I' \notin GY' \cup NB(GY')$. Az I' minimalitásából következik, hogy minden valódi részhalmaza eleme $GY' \cup NB(GY')$-nek és gyakori $\mathcal{T}$-ben. Ebből következik, hogy I' minden részhalmaza eleme GY^*-nak, amiből kapjuk, hogy $I' \in NB(GY^*)$. Ez ellentmond az $NB(GY^*) \subseteq GY' \cup NB(GY')$ feltételnek, hiszen van olyan elem, amely eleme a bal oldalnak, de nem eleme a jobb oldalnak. ■

Amennyiben a $\mathcal{T}'$-ben gyakori elemhalmazok meghatározásához az Apriori algoritmust használjuk, akkor az $NB(GY')$ elemeit nem kell külön előállítanunk. A ritka jelöltek halmaza ugyanis pontosan $NB(GY')$ lesz (ez a tulajdonság sem az Eclat, sem az FP-growth algoritmusra nem teljesül).

28.8. példa. Legyen $\mathcal{I} = \{A, B, C, D\}$ és $GY' = \{A, B, C\}$. Ekkor az $NB(GY')$ elemei az $\{AB, AC, BC, D\}$ halmazok. Tehát ennek a 7 elemhalmaznak fogjuk a támogatottságát meghatározni a teljes bemenetben. Ha például az A, B, C, AB halmazokat találjuk gyakorinak, akkor a tételbeli tartalmazási reláció fennáll, hiszen az $NB(\{A, B, C, AB\})$ összes eleme szerepel a 7 jelölt között. Nem mondhatjuk el ugyanezt, ha D-ről derül ki, hogy gyakori. Ekkor Toivonen algoritmusa jelenti, hogy nem biztos, hogy minden gyakori elemhalmazt megtalált.

Gyakorlatok

28.2-1. Mutassuk meg, hogy már a kételemű halmazokat tároló minimális méretű szófa megtalálása is NP-nehéz. *Útmutatás.* Használjuk fel azt, hogy egy gráf minimális lefogó élhalmazának meghatározása NP-nehéz feladat.

[4] Az NB jelölést a „negative border” rövidítéséből kapjuk.

28.2-2. Mutassunk példát arra, amikor nem a gyakoriság szerint csökkenő sorrend adja a minimális méretű szófát.
28.2-3. Adjunk olyan módszert az egy- és kételemű jelöltek támogatottságának meghatározására, amely a szófát használó módszernél gyorsabb és kevesebb memóriát használ.
28.2-4. Legyen $GY_2 = \{AB, AC, CD, DE, EF, FG\}$. Határozzuk meg $KK_2^{2+1}(|GY_2|)$ és $KK_{2,1}^*(GY_2)$ értékeket.
28.2-5. Melyik rendezéstől várhatjuk a legkisebb KK^* felső korlátot abban az esetben, ha az $\mathcal{I}$ elemeinek előfordulása függetlenek egymástól.
28.2-6. Adjuk olyan bemeneti sorozatot és *min-supp* küszöböt, amelyek esetében az FP-GROWTH és az ECLAT különböző módon járja be a keresési teret.

Feladatok

28-1. Nem bővíthető gyakori elemhalmazok
Az I elemhalmaz ***nem bővíthető, gyakori*** elemhalmaz, mennyiben nem létezik olyan gyakori elemhalmaz, amelynek I valódi részhalmaza. A nem bővíthető, gyakori elemhalmazokat egy utószűréssel megkaphatjuk a gyakori elemhalmazokból. Amennyiben csak a nem bővíthető gyakori minták kinyerése a cél, akkor hatékonyabban megoldhatjuk a feladatot olyan algoritmussal, amely eleve csak ezeket a halmazokat határozza meg. Hogyan kell módosítani az APRIORI, ECLAT és FP-GROWTH algoritmusokat, hogy csak a nem bővíthető, gyakori mintákat nyerje ki? *Útmutatás.* Gondoljuk meg, hogy melyek azok a gyakori bővíthető elemhalmazok, amelyek nem szabad törölnünk, mert még szükség van rájuk az algoritmusnak egy későbbi lépésében.)

28-2. Zárt elemhalmazok
Legyen az I elemhalmaz ***lezártja*** az a legnagyobb elemhalmaz, amelynek támogatottsága megegyezik I támogatottságával, és tartalmazza I-t részhalmazként. Az I lezártját jelölje $h(I)$.

a. Mutassuk meg, hogy a lezárás valóban egy lezárási operátor, tehát $I \subseteq h(I)$, idempotens, azaz $h(h(I)) = h(I)$ és monoton, más szóval $I \subseteq J$ esetén $h(I) \subseteq h(J)$.

b. Mutassuk meg, hogy minden maximális méretű gyakori elemhalmaz zárt is egyben.

c. A zártságot felhasználva hogyan tudjuk csökkenteni a jelöltek számát az APRIORI, ECLAT és FP-GROWTH algoritmusokban?

d. Hogyan kell módosítani az APRIORI, ECLAT és FP-GROWTH algoritmusokat, hogy eleve csak a gyakori, zárt mintákat nyerjék ki?

Megjegyzések a fejezethez

A gyakori elemhalmazok keresése először az asszociációs szabályok kinyerésének [7] részfeladataként merült fel 1993-ban. Egy évvel később Agrawal és Srikant publikálták az APRIORI algoritmust [6], illetve tőlük függetlenül Mannila, Toivonen és Verkamo [244]. Az öt

szerző végül egyesítette a két írást [4] (az optimális szófa NP-teljességét Comer és Sethi bizonyította a 3SAT problémára való visszavezetéssel [68]). A jelöltekre vonatkozó felső korlátokat Geerts, Goethals és Bussche adták [115]. A probléma felbukkanása után az Apriori algoritmus és különböző módosulatai voltak jellemzőek. A 90-es évek közepén még viszonylag kis memóriával rendelkeztek a számítógépek, így a legfőbb törekvés az algoritmusok I/O műveletei számának csökkentése volt. Emellett a jelöltek számát próbálták csökkenteni, mondván, hogy annál gyorsabb egy algoritmus, minél kevesebb időt tölt a hamis jelöltek támogatottságának meghatározásával.

Az Eclat [364] és FP-growth [156] algoritmusokat Zaki és Han által vezetett csoportok mutatták be. A két módszer rendkívül hatékonynak bizonyult és megcáfolták azt a korábbi hipotézist, hogy a sok ritka jelöltet előállító algoritmusok csak rosszak lehetnek. Ezzel, a mélységi bejárást megvalósító algoritmusok kiütötték a kutatások középpontjából a szélességi bejárást megvalósító algoritmusokat.

A GYEK algoritmusok nagy száma miatt 2003-ban megrendezték a GYEK implementációk első versenyét [136]. A rendezvény rengeteg tanulsággal és meglepetéssel szolgált. Sok új, gyorsnak beharangozott algoritmus meglehetősen rosszul szerepelt a teszteken, jócskán elmaradva még a legegyszerűbb algoritmustól, az Apriori-tól is. A versenynek nem volt abszolút győztese, a különböző karakterisztikával rendelkező adatbázisokon más-más implementáció győzött. Az FP-growth [293] és Eclat [338] algoritmusok gyorsítása és egy új algoritmus, az LCM [330] azonban mindig az elsők között szerepelt. Az Apriori [47, 48, 50] az alacsony memóriaigényével tűnt ki sok esetben. A versenyzők folyamatosan frissíthetik programjaikat, amelyek fél évenként összemérhetik erejüket. A legfrissebb eredmények és implementációk megtalálhatók a `http://fimi.cs.helsinki.fi` oldalon.

Nemcsak egyre gyorsabb algoritmusok születtek, hanem a minták típusa is egyre bonyolultabb lett. Elemhalmazokon túl kerestek gyakori sorozatokat, elemhalmazokat tartalmazó sorozatokat [5], epizódokat [243], Boole-formulákat [242], címkézett rendezett/nem rendezett gyökeres fákat [361] és gyakori részgráfokat [214], illetve feszített részgráfokat [173].

A nem bővíthető gyakori elemhalmazok fogalmát a 28-1. feladatban ismertettük. A nem bővíthető, gyakori elemhalmazok alapján tetszőleges elemhalmazról el tudjuk dönteni, hogy gyakori vagy sem, igaz, a pontos támogatottságokat nem tudjuk kikövetkeztetni. A nem bővíthető, gyakori elemhalmazok száma jóval kisebb lehet az összes gyakori elemhalmaz számánál, ezért sokszor csak ezeket nyerik ki. A legismertebb ilyen algoritmusok az FP-growth* [144], Mafia [57] és az afopt [229].

A gyakori elemhalmazoknak másik fontos részhalmaza a *zárt* gyakori elemhalmazok (pontos definíciót lásd 28-2.). A zárt elemhalmazokból meg tudjuk határozni a gyakori elemhalmazokat és azok pontos támogatottságát is. A zártság fogalmát Pasquier [263, 264, 265] és Zaki [363] vezette be egymástól függetlenül, és később számos algoritmus született (Apriori-Close [265], Charm [362], Closet [268], Closet+ [350], Mafia [57]), amelyek csak a gyakori zárt elemhalmazokat nyerik ki.

Két adatbányászatról szóló könyvet fordítottak magyarra [3, 155]. Mindkét könyv bevezető jellegű, elsősorban a nagyközönség számára készült. Ugyancsak bevezető jellegű, de sok témával fogalalkozó mű a Khosrow-Pour által szerkesztett enciklopédia [197], amely elektronikusan is elérhető. Gyakori elemhalmazok kereséséről a társítási szabályokról szóló részekben olvashatunk. A [3] könyvet röviddel a GYEK feladat születése után adták ki, így nem csoda, hogy csak az Apriori algoritmus szerepel benne. A [155]-ban az Apriori mellett

részletesen írnak az FP-GROWTH-ról is. A [46]-ban olvashatunk legtöbbet a gyakori minták kereséséről. Szerepel az írásban elemhalmazok keresésén kívül sorozatok, epizódok, címkézett fák és gráfok keresése is. Szó van még a munkában a GYEK feladat változatairól is: például zárt és nem bővíthető minták keresése, dinamikus GYEK, kényszerek kezelése, illetve GYEK változó támogatottsági küszöb esetén.

A szerző munkáját részben támogatta a T 042706 számú OTKA-szerződés.

29. Klaszterezés

A ***klaszterezés*** egy adathalmaz pontjainak, rekordjainak hasonlóság alapján való csoportosítását jelenti. A klaszterezés szinte minden nagyméretű adathalmaz ***leíró modellezésére*** alkalmas. A teljesség igénye nélkül néhány példa: csoportosíthatunk weboldalakat tartalmuk, webfelhasználókat böngészési szokásaik, kommunikációs és szociális hálózatok pontjait közösségeik, kémiai vegyületeket szerkezetük, géneket funkcióik, betegségeket tüneteik szerint.

A klaszterezés feladatával eleinte a statisztikában kezdtek foglalkozni, majd az adatbányászat keretében az igen nagy méretű adathalmazok klaszterezésének kérdései kerültek az előtérbe. Így a klaszterezés az adatbányászat egyik legrégebbi és talán leggyakrabban alkalmazott része. A klaszterezés, illetve osztályozás során az adatpontokat diszjunkt csoportokba, a későbbiekben használt nevükön klaszterekbe, illetve osztályokba soroljuk, azaz particionáljuk az adathalmaz elemeit. A klaszterezés célja, hogy az elemeknek egy olyan partícióját adjuk meg, amelyben a közös osztályba kerülő elempárok lényegesen hasonlóbbak egymáshoz, mint azok a pontpárok, melyek két különböző osztályba sorolódtak.

Klaszterezés során a megfelelő csoportok kialakítása nem egyértelmű feladat, hiszen a különböző adatok eltérő jelentése és felhasználása miatt adathalmazonként más és más szempontok szerint kell csoportosítanunk. További nehézséget jelent, hogy egy n darab adatpontot tartalmazó adathalmaznak Bell-számnyi, $O(e^{n \lg n})$ darab klaszterezése lehet, vagyis az adatpontok számában exponenciálisnál nagyobb méretű keresési térben kívánunk egy – több szempontból is – optimális klaszterezést megtalálni.

A klaszterezéshez hasonló feladat az osztályozás. A klaszterezéstől az osztályozás feladatát az különbözteti meg, hogy az osztályozás *felügyelt* (supervised), a klaszterezés pedig *felügyelet nélküli* (unsupervised) csoportosítás. Osztályozás esetén a választható osztályok valamilyen ábrázolással, mintával vagy modellel leírva előre adottak. Klaszterezés esetén előzetesen megadott osztályok nincsenek, az adatok maguk alakítják ki a klasztereket, azok határait.

Egy klaszterezési feladat megoldásához ismernünk kell a fellelhető algoritmusok lényeges tulajdonságait, illetve szükségünk van az eredményként kapott klaszterezés kiértékelésére, jóságának mérésére. Ezeket az alapvető elemeket tárgyalja a 29.2. fejezet.

Mivel egy klaszterezés az adatpontok hasonlósági viszonyaiból indul ki, ezért az első fontos lépés az adatpontok páronkénti hasonlóságát lehető legjobban megragadó hasonlósági függvény kiválasztása. A különféle adattípusokra leggyakrabban használt távolság és hasonlóság függvények leírása a 29.3. fejezetben található meg.

Nagy adathalmazok klaszterezése során gyakran találkozunk azzal a problémával, hogy az adatpontok távolságának, illetve hasonlóságának kiszámítása nem oldható meg elég gyorsan ahhoz, hogy az adott klaszterező algoritmus elfogadható futási időben véget érjen. Számos esetben e problémának az adatok sok (tízes nagyságrendnél nagyobb számú) attribútummal való jellemzése az oka. Ebben az esetben szoktuk az adatok dimenzióját nagynak tekinteni. A hasonlóság vagy távolság hatékony kiszámításának problémáját megoldhatja az adatok dimenzió-csökkentése, amely során az adathalmazt – az adatpontok közötti hasonlósági viszonyokat a legkevésbé torzítva – olyan formába alakítjuk, amelyen már hatékonyan tudjuk a kívánt klaszterező algoritmust futtatni. A dimenzió-csökkentés két módszeréről olvashatunk a 29.4. fejezetben.

A klaszterező módszerek tárgyalása során bemutatunk három klaszterezési elvet és azok legjellemzőbb algoritmusait. A particionáló algoritmusok (29.5. fejezet) és a hierarchikus módszerek (29.6. fejezet) a főleg a régebb óta ismert, de ma is gyakran használt módszereket képviselik, míg a sűrűség alapú eljárások (29.7. fejezet) az elmúlt évtized új elveire, eljárásaira mutatnak példát.

29.1. Alapok

29.1.1. A hasonlóság és távolság tulajdonságai

A klaszterezés általunk használt modelljeiben minden egyes elempár *hasonlósága* vagy *távolsága* ismert, illetve kiszámítható.

29.1. definíció. *Tetszőleges két x,y klaszterezendő elem esetén azok* ***hasonlóságát*** *$s(x,y)$ jelöli. Feltesszük, hogy $0 \leq s(x,y) \leq 1$ és $s(x,y) = s(y,x)$ minden x,y pontpárra, valamint $s(x,x) = 1$ minden x adatpontra.*

Hasonlóság helyett többször az x,y pontok $d(x,y)$-vel jelölt távolságát fogjuk használni.

29.2. definíció. *Egy $d(x,y)$ függvény* ***távolság*** *ha teljesülnek rá a következő tulajdonságok:*

1. *minden x,y esetén $0 \leq d(x,y) < \infty$,*
2. *minden x adatpontra $d(x,x) = 0$,*
3. *szimmetrikus, azaz minden x,y esetén $d(x,y) = d(y,x)$.*

Időnként feltesszük, hogy d-re teljesül a *háromszög-egyenlőtlenség*.

29.3. definíció. *Egy d távolságra akkor teljesül a* ***háromszög-egyenlőtlenség****, ha minden x,y,z ponthármasra $d(x,z) \leq d(x,y) + d(y,z)$. Ha egy távolságra a háromszög-egyenlőtlenség teljesül, akkor azt* ***metrikának*** *hívjuk.*

A későbbiekben, hacsak külön nem említjük, nem követeljük meg a háromszög-egyenlőtlenség teljesülését. Számos esetben az előforduló hasonlóság és távolság összekapcsolható a $d(x,y) = 1 - s(x,y)$ vagy az $s(x,y) = 1/(1 + d(x,y))$ összefüggésekkel.

29.1.2. Mátrixábrázolások

Egy klaszterező eljárás bemenetét a szimmetrikus ***távolságmátrixszal*** is reprezentálhatjuk:

$$\begin{bmatrix} 0 & d(1,2) & d(1,3) & \cdots & d(1,n) \\ d(2,1) & 0 & d(2,3) & \cdots & d(2,n) \\ d(3,1) & d(3,2) & 0 & \cdots & d(3,n) \\ & \vdots & \vdots & \ddots & \vdots \\ d(n,1) & d(n,2) & d(n,3) & \cdots & 0 \end{bmatrix},$$

ahol $d(i, j)$ adja meg az i-edik és a j-edik elem távolságát és n az adatpontok száma. Ennek mintájára értelmezhető a ***hasonlóságmátrix*** is.

A gyakorlatban legtöbbször az n adatpont attribútumokkal van leírva, és a hasonlóságokat az attribútumok értékeiből valamilyen függvény segítségével számolhatjuk ki. Ha megadjuk a függvényt, akkor elvben felírhatjuk a megfelelő hasonlóságmátrixot.

Elképzelhető, hogy a mátrix elemeinek kiszámolása megoldható, de elemeinek száma olyan nagy, hogy a hasonlóságmátrix nem fér el a belső tárban. Ha a hasonlóságmátrix ritka, azaz ha csak lineárisan sok nullától különböző elem szerepel benne, akkor segítséget jelent a ***ritka mátrixos tárolási forma*** használata. Ekkor a mátrixnak csak a nullától különböző elemeit tároljuk el.

29.2. A klaszterező algoritmusok jóságának kérdései

Egy „jó” klaszterező algoritmussal szemben sokféle követelményt támaszthatunk. Ebben a fejezetben először arra mutatunk példát, hogy egy klaszterező eljárás kimenetének *jóságát* hogyan lehet numerikusan mérni. A fejezet további részében megadjuk azokat a legfontosabb tulajdonságokat, amelyeket egy klaszterező eljárástól meg szeretnénk követelni, majd a *kívülálló* adatpontok problémáját tárgyaljuk. Végül a 29.4. tétel a klaszterezés elvi határaira mutat példát: nem létezik túl sok intuitív szempontnak *pontosan* megfelelő klaszterező algoritmust.

Ha egy klaszterező eljárás jóságát, eredményének pontosságát szeretnénk mérni, és rendelkezésünkre áll egy előre adott, pontosnak tartott minta klaszterezés, akkor a vizsgálandó algoritmus adta klaszterezést össze tudjuk hasonlítani a minta klaszterezéssel. Egy ilyen összehasonlítás alapja a ***tévesztésmátrix.*** Legyen a minta klaszterezés t darab osztálya $M_1, M_2, \ldots, M_t$, és az eredményként kapott klaszterezés s darab osztálya $K_1, K_2, \ldots, K_s$. A $t \times s$ méretű C tévesztésmátrix elemei legyenek $c_{i,j} = |M_i \cap K_j|$. Látható, hogy ha a klaszterező algoritmus pontosan a minta klaszterezést adja eredményképpen, akkor C (feltéve a klaszterek megfelelő címkézését) egy diagonális mátrix lesz.

Minden M_i, K_j klaszterpárra értelmezhetjük az

$$r_{i,j} = \frac{|M_i \cap K_j|}{|K_j|}$$

felidézés (recall) és a

$$p_{i,j} = \frac{|M_i \cap K_j|}{|M_i|}$$

pontosság (precision) mennyiségeket. Az így kapott $t \cdot s$ mutatószámot többféleképpen aggregálhatjuk. Az egyik elterjedt lehetőség, hogy a koordinátánkénti két érték harmonikus közepeinek összegzése,

$$2 \cdot \sum_{i,j} \frac{|M_i \cap K_j|}{|M_i| + |K_j|} ,$$

adja a két klaszterezés eltérését.

Alább felsorolunk olyan általános tulajdonságokat, amelyek döntően befolyásolják egy klaszterező algoritmus alkalmazhatóságát.

Skálázhatóság: az algoritmus tár és idő szükséglete agy adathalmazon, illetve magas dimenziós adatokon alkalmazva is kezelhető maradjon.

Adattípusok: többfajta adattípusra is működjön (numerikus, bináris, kategorikus és ezek keverékei).

Klaszter-geometria: ne preferáljon a kialakítható klaszterek között azok geometriai tulajdonságai alapján.

Robusztusság: ne legyen zajos adatokra érzékeny;

Sorrend-függetlenség: ne függjön az adatpontok beolvasási sorrendjétől.

Hasznos tulajdonsága lehet egy klaszterező algoritmusnak, hogy ha előre megadhatunk számára klaszterhatárokat, azaz a kimenete egy előre adott (kényszer) klaszterezés finomítása lesz.

Gyakorlati példák esetében gyakran találkozhatunk olyan adatpontokkal, amelyekhez kevés más pont hasonló. Az olyan adatpontokat, amelyek lényegesen különböznek szinte minden más adatponttól, ***kívülállóknak*** (outlier) nevezzük. A gyakorlatban előforduló kívülálló adatpontok egyik forrása mérési hiba, zaj. Ebben az esetben a kívülálló pontok elhagyása kívánatos. Ezzel szemben ha pontos adatok alapján bizonyul egy adatpont lényegesen különbözőnek az adathalmaztól, akkor ez az információ fontos lehet. Mindkét esetben előnyben részesítjük a kívülállókat elkülönítő klaszterező algoritmusokat. Végül célszerű lehet a klaszterezést a várhatóan homogénebb maradék adatpontokon újra elvégezni, mert a kívülállók kihagyásával a maradék adathalmazt leíró modell, klaszterezés minősége ugrásszerűen javulhat.

Egy klaszterező algoritmustól számos intuitíven többé-kevésbé indokolt formális tulajdonságot követelhetünk meg. Ilyen tulajdonságok például az alábbiak:

Skála-invariancia: ha minden elempár távolsága helyett annak az α-szorosát vesszük alapul (ahol $\alpha > 0$), akkor a klaszterező eljárás eredménye ne változzon meg.

Gazdagság: tetszőleges előre megadott klaszterezéshez létezzen olyan távolságmátrix, hogy a klaszterező eljárás az adott előre megadott partícióra vezessen.

Konzisztencia: tekintsük egy adott adathalmazon alkalmazott klaszterező algoritmus eredményét. Ha ezután a pontok közötti távolságokat úgy változtatjuk meg, hogy tetszőleges, azonos csoportban lévő elempárok között a távolság nem nő, illetve különböző csoportban lévő elempárok közötti távolság nem csökken, akkor az újonnan kapott távolságokra alkalmazott algoritmus az eredetivel megegyező csoportosítást adja.

Meglepő negatív eredmény a következő, legalább kételemű adathalmazokat klaszterező algoritmusokról szóló Kleinberg-tétel, amelyet bizonyítás nélkül közlünk.

29.4. tétel. *Nem létezik olyan klaszterező eljárás, ami egyszerre rendelkezik a skálainvariancia, gazdagság és konzisztencia tulajdonságokkal.*

29.3. Adattípusok és távolságfüggvények

Egy adathalmaz klaszterezésének sikere nagymértékben függ a megfelelő, az adatok hasonlóságát jól modellező hasonlósági, illetve távolság függvények megválasztásától. Adatpontokat alapvetően kétfajta attribútum típussal szokták leírni: *numerikus* és *kategorikus értékekkel.* Numerikus érték esetén az attribútumok valós számok, amelyeken aritmetikai műveleteket és rendezést alkalmazhatunk. Kategorikus adatok esetén csak az adatpontok megfelelő attribútumainak egyenlőségét vagy különbözőségét tudjuk felhasználni az adatpontok közötti távolságok definiálására.

29.3.1. Numerikus adatok

Numerikus attribútumok esetén a kapott vektorok távolságának mérésére a legfontosabb példa az L_p vagy más néven ***Minkowski-metrika.*** Legyen u, v két adatpont (illetve az őket leíró vektorok) és u_i, v_i az attribútumok (koordináták) értékei, ahol $i = 1, \ldots, d$. Ekkor

$$d_p(u, v) = \left(\sum_{i=1}^{d} |u_i - v_i|^p \right)^{1/p}$$

adja a két pont távolságát. A p paraméter értéke elvileg tetszőleges pozitív szám lehet. A legnépszerűbb a $p = 2$ értékhez tartozó, geometriailag és statisztikailag könnyen interpretálható ***euklideszi metrika.*** A $p = 1$ értékhez tartozó metrikát ***Manhattan-távolságnak*** nevezik. A $p \to \infty$ esetben kapjuk a

$$d_{\max}(u, v) = \max_{i \in \{1,\ldots,d\}} |u_i - v_i|$$

maximum távolságot.

Ha az adatpontokat leíró attribútumok nagysága helyett csak azok aránya lényeges, akkor az adatpontok távolságának mérésére a két attribútumvektor által bezárt szög koszinuszát megadó

$$d_{\cos}(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2} = \frac{\sum_{i=1}^{d} u_i v_i}{\sqrt{\sum_{i=1}^{d} u_i^2 \sum_{i=1}^{d} v_i^2}}$$

koszinusz távolságot alkalmazhatjuk.

29.3.2. Bináris és kategorikus adatok

Kategorikus attribútumok esetén az egyező értéket felvett attribútum számát osztva az összes attribútum számával kapjuk a

$$s_R(u, v) = \frac{|\{i \mid u_i = v_i\}|}{d}$$

Rand-hasonlóságot (másnéven egyszerű illeszkedési hasonlóságot).

Bináris (0-1) értékű attribútumok esetén, amikor a hasonlóság szempontjából csak az 1 értéket felvett attribútumok megegyezése lényeges, a

$$s_J(u,v) = \frac{|\{i \mid u_i = 1 \text{ és } v_i = 1\}|}{\left|\{i \mid u_i = 1 \text{ vagy } v_i = 1\}\right|}$$

Jaccard-hasonlóságot használhatjuk.

Kettőnél több értékű kategorikus attribútumok átírhatók bináris attribútumokká úgy, hogy az attribútum minden lehetséges értékéhez hozzárendelünk egy bináris változót, amely 1 lesz, ha az attribútum felveszi az adott értéket, és 0 lesz különben.

Gyakorlatok

29.3-1. Legyen $G = (V, E)$ egy egyszerű gráf. Mutassuk meg, hogy a gráf pontjain értelmezett

$$d(u,v) = \begin{cases} 1, & \text{ha } (u,v) \in E; \\ 2, & \text{különben} \end{cases}$$

függvény metrika.

29.3-2. Legyen d egy metrika és t egy tetszőleges pozitív szám. Mutassuk meg, hogy a

$$d'(u,v) = \frac{d(u,v)}{d(u,v) + t}$$

függvény is metrika.

29.3-3. Mutassuk meg, hogy ha az s a

a. Rand-hasonlóságot,

b. Jaccard-hasonlóságot

jelöli, akkor az $1 - s$ függvény metrika.

29.4. Dimenzió-csökkentés

A klaszterező algoritmusok alapvető építőeleme egy hatékonyan kiértékelhető, az adatpontok közötti viszonyokat hűen tükröző hasonlóság- vagy távolságfüggvény. A *dimenzió-csökkentés* módszere, melynek során az adatpontok túlságosan sok attribútummal való leírását kevesebb attribútumot használó új leírással helyettesítjük, a korábban bevezetett hasonlósági függvényeken több szempontból is javíthat. A továbbiakban az eredeti adathalmazt az $m \times n$-es M mátrixszal (m az adatpont, n az attribútumok száma), az új leírást pedig az $m \times k$ méretű $\widehat{M}$ mátrixszal (k darab új attribútum, lásd a 29.1. ábrát) reprezentáljuk.

A dimenzió-csökkentésnek eltérő céljai lehetnek. Egyik nyilvánvaló cél az adatbázis tömörítése, a tárolandó adatok mennyiségének csökkentése. Másik lehetséges cél, hogy a csökkentés utáni $\widehat{m}$ mátrix sorai között hatékonyan tudjuk kiértékelni két adatpont hasonlóságát, illetve hogy megfelelő adatszerkezettel támogassuk az olyan lekérdezéseket, amelyek egy adott ponthoz kikeresik a hozzá leghasonlóbbakat. Végül a dimenzió-csökkentés céljai között megemlíthetjük a zajszűrést is. E mögött az a feltételezés áll, hogy az adatpontok egy alacsony dimenziójú térből származnak, és a dimenzió megnövekedését valamilyen véletlen zaj okozza. Az utóbbi esetben a dimenzió-csökkentési eljárás után számított hasonlóság értékek pontosabbak lesznek, mint ha az eredeti M mátrixból számítanánk őket.

29.1. ábra. A dimenziócsökkentés célja, hogy az M mátrix mérete jelentősen csökkenjen, azaz $k \ll n$ legyen.

Az alábbiakban két példát mutatunk dimenzió-csökkentésre; mindkettő jelentősen csökkentheti egy adatbázis méretét. Az első példánál egy lineáris algebrai szempontból optimális dimenzió-csökkentést ismertetünk, melynek zajszűrési tulajdonságait több tapasztalati tény is mutatja. A második ismertetett módszer algoritmikus szempontból érdekes, segítségével nagyban növelhető a hasonlósági lekérdezések hatékonysága.

29.4.1. Szinguláris felbontás

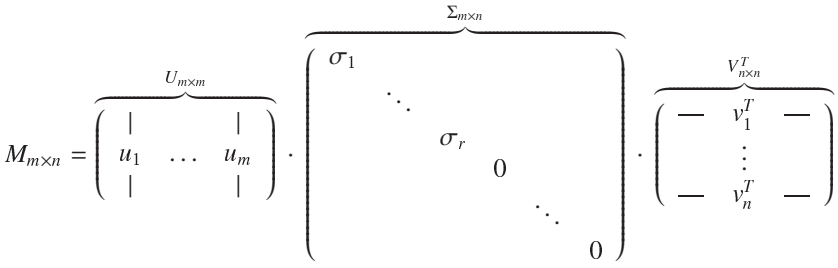
A szinguláris felbontás az elméleti szempontból egyik legtöbbet vizsgált, klasszikus lineáris algebrai eszközöket használó dimenzió-csökkentési eljárás. Ennek alkalmazása után nyert $\widehat{M}$ mátrix soraiból jól közelíthető az euklideszi távolság, illetve az attribútumok vektoraiból számított skaláris szorzattal mért hasonlóság. Utóbbi megegyezik a koszinusz mértékkel, ha a mátrix sorai normáltak. Ebben a pontban néhány jelölés és alapvető fogalom után definiáljuk a szinguláris felbontást, igazoljuk a felbontás létezését, majd megmutatjuk, hogy miként használható a felbontás dimenzió-csökkentésre. Megjegyezzük, hogy a szakasz nem mutat a gyakorlatban numerikus szempontból jól alkalmazható módszert a felbontás kiszámítására. Kisebb adathalmaz esetén általános lineáris algebrai programcsomag (Matlab, Octave, Maple) használata javasolt, míg nagyobb adatbázisoknál az adatok sajátosságát kihasználó szinguláris felbontó program (SVDPack) használata ajánlott.

Egy $U \in \mathbb{R}^{n \times n}$ mátrixot ***ortogonálisnak*** nevezünk, ha oszlopai ortogonális rendszert alkotnak, azaz $U^T U = I_n$, ahol I_n az $n \times n$ méretű egységmátrixot, és U^T az U transzponáltját jelöli. Másképpen mondva U invertálható és U^{-1} inverzére $U^{-1} = U^T$ teljesül. Mátrix ortogonalitásának szemléletes tárgyalásához szükségünk lesz a vektorok hosszának általánosítására, a norma fogalmára. Egy $v \in \mathbb{R}^n$ vektor $\|v\|_2$-vel jelölt ***2-normáját*** a $\|v\|_2 = \sqrt{\sum_i v_i^2}$ egyenlőséggel definiáljuk. Egyszerűen látható, hogy $\|v\|_2{}^2 = v^T v$ teljesül. A 2-norma általánosítása a tetszőleges $M \in \mathbb{R}^{m \times n}$ mátrix esetén értelmezett $\|M\|_F$ ***Frobenius-norma***, amelynek definíciója $\|M\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} M_{i,j}^2}$.

Visszatérve az ortogonalitás szemléletes jelentésére, egy ortogonális mátrix által reprezentált lineáris transzformációra úgy gondolhatunk, mint egy forgatásra, amely a vektorok hosszát nem változtatja. A szemlélet alapja, hogy tetszőleges $U \in \mathbb{R}^{n \times n}$ ortogonális mátrix és $x \in \mathbb{R}^n$ vektor esetén

$$\|Ux\|_2 = \|x\|_2$$

teljesül. Az azonosság az alábbi elemi lépésekből következik: $\|Ux\|_2^2 = (Ux)^T(Ux) = x^T(U^T U)x = x^T x = \|x\|_2^2$. Hasonlóan belátható, hogy tetszőleges $X \in \mathbb{R}^{m \times n}$ mátrix ese-



29.2. ábra. A szinguláris felbontás sematikus vázlata.

tén és $U \in \mathbb{R}^{m\times m}$ illetve $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok esetén igaz, hogy

$$\left\|UXV^T\right\|_F = \|X\|_F \ .$$

A rövid bevezető után rátérünk a szinguláris felbontás definíciójára. Egy nem szükségszerűen négyzetes $M \in \mathbb{R}^{m\times n}$ mátrix ***szinguláris érték felbontásán*** (singular value decomposition, SVD) az olyan

$$M = U\Sigma V^T$$

szorzattá bontást értjük, ahol $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok, továbbá a Σ mátrix M-mel megegyező méretű és a bal felső sarokból 45°-ban lefele elhelyezkedő $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ pozitív számokat csupa 0 követ és a többi elem szintén 0. A σ_i számokat ***szinguláris értékeknek*** nevezzük, és a $\sigma_i = 0$ választással terjesztjük ki az $i > r$ esetre. A felbontásból látható, hogy $\operatorname{rang}(M) = \operatorname{rang}(\Sigma) = r$.

Az U és a V oszlopait ***bal-, illetve jobboldali szinguláris vektoroknak*** mondjuk. A jelölések áttekintése a 29.2. ábrán látható.

29.5. tétel. *Tetszőleges $M \in \mathbb{R}^{m\times n}$ mátrixnak létezik szinguláris érték felbontása, azaz léteznek $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok, melyekkel*

$$M = U\Sigma V^T,$$

ahol

$$\Sigma \in \mathbb{R}^{m\times n}, \qquad \Sigma = \begin{pmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{pmatrix},$$

továbbá Σ^+ egy $r \times r$ méretű diagonális mátrix, amelynek főátlójában a $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ számok helyezkednek el sorrendben.

Bizonyítás. Az $M^T M$ mátrix szimmetrikus, ezért ortogonális transzformációval diagonalizálható és sajátértékei valósak. Továbbá pozitív szemidefinit, mert tetszőleges $x \in \mathbb{R}^{n\times n}$ vektor esetén $x^T M^T Mx = (Mx)^T(Mx) = \|Mx\|_2^2 \geq 0$, ezért a sajátértékek nem negatívak. A sajátértékek legyenek $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_r^2 > 0$. Az ezekhez tartozó sajátvektorokból alkotott ortogonális mátrixot jelölje V, ekkor

$$V^T M^T MV = \begin{pmatrix} \Sigma_+^2 & 0 \\ 0 & 0 \end{pmatrix}.$$

A mátrixot két részre osztva $V = (V_r\, V_2)$, ahol $V_r \in \mathbb{R}^{n\times r}$ a pozitív sajátértékhez tartozó sajátvektorokat tartalmazza. Vagyis

$$V_r^T M^T M V_r = \Sigma_+^2 .$$

Vezessük be az

$$U_r = M V_r \Sigma_+^{-1}$$

jelölést, ekkor

$$M = U_r \Sigma_+ V_r^T .$$

Az U_r vektorai ortogonális vektorrendszert alkotnak, ezt tetszőlegesen kiegészítve $U = (U_r\, U_2)$ ortogonális mátrixszá a

$$M = U \begin{pmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{pmatrix} V^T$$

eredményt kapjuk. ■

Most megmutatjuk, hogy szinguláris felbontás segítségével hogyan lehet dimenzió-csökkentést végrehajtani. Emlékeztetünk rá, hogy az M mátrix n-dimenziós sorvektorai adatpontokat jellemeznek. Dimenzió-csökkentéskor az n attribútumot szeretnénk $k < n$ dimenziójú vektorokkal jellemezni úgy, hogy közben az adatpontok euklideszi távolsága vagy skaláris szorzattal mért hasonlósága csak kis mértékben változzon. A mátrixszorzás elemi tulajdonsága, hogy a szinguláris felbontás az alábbi formában is írható.

$$M = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T ,$$

ahol $u_i v_i^T$ a bal- illetve a jobboldali szinguláris vektorokból képzett diádszorzat, azaz egy oszlop- és egy sorvektor szorzataként felírt $m \times n$ méretű 1-rangú mátrix. Látható, hogy az $u_i v_i^T$ diádok monoton csökkenő σ_i súllyal szerepelnek az összegben. Innen adódik az ötlet, hogy $k < r$ esetén csak az első k legnagyobb súlyú diád összegével közelítsük az M mátrixot. Azaz

$$M_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T = U_k \Sigma_k V_k^T ,$$

ahol $U_k = (u_1\, u_2\, \ldots\, u_k)$ és $V_k = (v_1\, v_2\, \ldots\, v_k)$, valamit Σ_k egy $k\times k$ méretű diagonális mátrix, melynek főátlójában a $\sigma_1, \sigma_2, \ldots, \sigma_k$ értékek vannak. Könnyen látható, hogy M_k sorai egy k-dimenziós altérben helyezkednek el, hiszen $\operatorname{rang}(M_k) = \operatorname{rang}(\Sigma_k) = k$. Sokkal mélyebb eredmény a következő, melynek bizonyítását mellőzzük.

29.6. tétel. *Legyen M egy legalább k rangú mátrix és legyen M_k a fenti módon számított közelítése. Ha a közelítés hibáját Frobenius-normával mérjük, akkor a k-rangú mátrixok közül az M_k mátrix a lehető legjobban közelíti M-et, azaz*

$$\|M - M_k\|_F = \min_{N:\, \operatorname{rang}(N)=k} \|M - N\|_F .$$

Továbbá a közelítés hibája a σ_i szinguláris értékekkel kifejezhető:

$$\|M - M_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2} .$$

Az M_k mátrix sorai az M-éhez hasonlóan n méretűek, de most már egy k-dimenziós altérnek az elemei. Ennek az altérnek egy bázisát alkotják a V_k^T sorai, és az

$$M' = U_k \Sigma_k$$

mátrix k-dimenziós sorvektorai e bázisban fejezik ki az M_k sorait. Tehát a dimenzió-csökkentés eredménye, hogy az M mátrix n-dimenziós sorait a vetítés után az M' mátrix k-dimenziós soraival közelítjük. A V_k^T sorainak ortogonalitásából könnyen belátható, hogy az M_k, illetve az M' soraiból számított euklideszi távolságok és skaláris szorzatok is megegyeznek. Tehát a közelítés alatt torzítás kizárólag az M-ből M_k-ba történő vetítés során történik, melynek mértéke a 29.6. tétel alapján felülről becsülhető.

Gyakorlatok

29.4-1.

a. Igazoljuk, hogy ha az u és v vektorokra $\|u\|_2 = \|v\|_2 = 1$ teljesül, akkor az uv^T diádszorzatra is fennáll, hogy $\left\|uv^T\right\|_F = 1$.

b. Bizonyítsuk be, hogy tetszőleges M mátrix esetén, ahol M szinguláris értékeit σ_i jelöli,

$$\|M\|_F = \sqrt{\sum_{i=1}^{r} \sigma_i^2} .$$

c. Lássuk be a 29.6. tétel második egyenlőségét.

29.4-2. Az M mátrix sorai által reprezentált adatpontok hasonlóságát skaláris szorzattal számítjuk. Az $M \cdot M^T$ mátrix *hasonlóságmátrix* lesz, mert elemei az egyes adatpontok közti hasonlóságok értékeivel egyeznek meg. Fejezzük ki az M' mátrixból számított hasonlóságmátrix Frobenius-normában mért hibáját a σ_i értékekből és k-ból, ahol a jelölések megegyeznek a 29.6. tételben használtakkal.

29.4-3. Legyen M egy $m \times n$ méretű mátrix, jelölje A az alábbi $(m+n) \times (m+n)$ méretű mátrixot.

$$A = \begin{pmatrix} 0 & M \\ M^T & 0 \end{pmatrix}$$

Az A mátrix szimmetrikus, ezért tudjuk, hogy sajátértékei valósak, és sajátvektorai ortogonális rendszert alkotnak. Azaz létezik az $A = WDW^T$ szorzattá bontás, ahol W ortogonális és D diagonális mátrix, melynek főátlójában valós számok szerepelnek.

a. Igazoljuk, hogy A pozitív sajátértékei megegyeznek M szinguláris értékeivel.

b. Lássuk be, hogy ha $\lambda > 0$ sajátértéke A-nak, akkor $-\lambda$ is az.

c. Adjunk új bizonyítást a 29.5. tételre felhasználva, hogy az A-nak létezik az $A = WDW^T$ sajátérték-felbontása.

29.4.2. Ujjlenyomat alapú dimenzió-csökkentés

Ebben a pontban egy véletlen mintavételezésen alapuló, elemi eszközöket használó dimenzió-csökkentési eljárást tárgyalunk, melynek segítségével bináris attribútumok ál-

tal jellemzett adatpontok Jaccard-hasonlóságát lehet közelíteni. Először ismertetjük, hogy miként lehet a dimenziót csökkenteni, majd a rövidebb vektorokból hasonlóságot számítani. Utána megmutatjuk a módszer igazi erejét, egy olyan hasonlóság-keresési eljárást, amely külső táron elhelyezett adatbázison is működik, lekérdezési időben korlátozott számú adatbázis-hozzáférést igényel, ezért különösen nagy adatbázisokon is alkalmazható.

Tegyük fel, hogy az $M \in \{0,1\}^{m \times n}$ mátrix m sora m különböző adatpontot jellemez egyenként n darab bináris attribútummal. Jelöljük $X(i)$-vel az i-edik sorban elhelyezkedő 1-ek indexeinek halmazát, azaz

$$X(i) = \{\ j\ :\ M_{ij} = 1\ \}.$$

A továbbiakban feltesszük, hogy $X(i) \neq \emptyset$. Az i_1 és i_2 adatpontok $s_J(i_1, i_2)$ Jaccard-hasonlóságát a korábbi definíció alapján az alábbi formában is írhatjuk:

$$s_J(i_1, i_2) = \frac{|X(i_1) \cap X(i_2)|}{|X(i_1) \cup X(i_2)|}\ .$$

Most egy olyan eljárást ismertetünk, amely minden adatponthoz egy véletlen ujjlenyomatot rendel úgy, hogy az i_1 és i_2 ujjlenyomata pont $s_J(i_1, i_2)$ valószínűséggel egyezzen meg. Legyen σ az attribútumok $1, 2, \cdots, n$ indexeinek egy véletlen permutációja, amelyet az $n!$ darab permutáció közül egyenletes eloszlás szerint választunk ki. Az i-edik adatponthoz tartozó $u(i)$ *ujjlenyomatot* az alábbiak szerint választjuk meg:

$$u(i) = \operatorname*{argmin}_{j \in X(i)} \sigma(j)\ \ ,$$

azaz $u(i) = j$, ahol j az $M_{ij} = 1$ egyenlőséget kielégítő indexek közül a σ permutáció alkalmazása után a legkisebb értéket kapja.

29.7. tétel. *A tetszőleges i_1 és i_2 adatpontok ujjlenyomatai $s_J(i_1, i_2)$ valószínűséggel egyeznek meg, azaz*

$$\Pr\{u(i_1) = u(i_2)\} = s_J(i_1, i_2)\ .$$

Bizonyítás. Az $u(i_1) = u(i_2)$ egyenlőség pontosan akkor teljesül, ha

$$\operatorname*{argmin}_{j \in X(i_1) \cup X(i_2)} \sigma(j) \in X(i_1) \cap X(i_2).$$

Az utóbbi esemény valószínűsége $|X(i_1) \cap X(i_2)|\ /\ |X(i_1) \cup X(i_2)| = s_J(i_1, i_2)$, mivel bármelyik $X(i_1) \cup X(i_2)$-beli elem azonos valószínűséggel lesz a permutáció alkalmazása után minimális. ■

Az előző állítás alkalmazásával egy olyan algoritmust mutatunk, amely független minták átlagaként becsüli a hasonlóság-értékeket.

Először a dimenzió-csökkentő lépés során minden adatponthoz k darab ujjlenyomatot generálunk, ezzel az $M \in \{0,1\}^{m \times n}$ mátrixból egy $M' \in \{1, 2, \cdots, n\}^{m \times k}$ mátrixot készítünk az alábbi pszeudokód szerint. Fontos, hogy a $\sigma_1, \sigma_2, \cdots, \sigma_k$ véletlen permutációkat egymástól függetlenül válasszuk meg.

UJJLENYOMAT(M,k)

1 **for** $\ell \leftarrow 1$ **to** k
2 **do** σ_ℓ = az $1, 2, \cdots, n$ indexek egy véletlen permutációja
3 $M'_{i\ell} = \text{argmin}_{j\,:\,M_{ij}=1}\, \sigma_\ell(j)$
4 **return** M'

Ha az ujjlenyomatokat már elkészítettük, és az i_1 és i_2 adatpontok hasonlóságát szeretnénk kiértékelni, akkor összeszámoljuk, hogy hány pozícióban egyezik meg az M' mátrix i_1-ik és i_2-ik sora, majd az egyezések számát osztjuk k-val, azaz a

$$\frac{\left|\{\, \ell \,:\, M'_{i_1\ell} = M'_{i_2\ell} \,\}\right|}{k}$$

mennyiséggel becsüljük az $s_J(i_1, i_2)$ hasonlóságot. Bizonyítás nélkül megemlítjük, hogy bármely rögzített $\delta > 0$ esetén annak a valószínűsége, hogy a becsült hasonlóság a valóditól legalább δ értékkel eltér, k növelésével exponenciálisan tart 0-hoz.

Megjegyezzük, hogy n elem véletlen permutációjának kiszámítása, tárolása és a permutáció által meghatározott rendezés hatékony lekérdezése nem könnyű feladat nagy n esetén. A gyakorlatban nem az összes lehetséges $n!$-féle permutáció közül választunk, hanem permutációk egy kisebb, tömören reprezentálható családjából. Egy lehetséges megoldás, hogy választunk egy $p \gg n$ prímet. Ekkor a véletlen permutációt úgy kapjuk, hogy az a, b számokat egyenletesen véletlenül generáljuk az $[1, p-1]$ intervallumból, és az $j \in \{1, 2, \cdots, n\}$ indexek σ permutációbeli sorrendjét az $a \cdot j + b \pmod p$ értékek sorrendjével határozzuk meg. Rögzített p esetén a permutációk családjának mérete $(p-1)^2$, és egy permutáció tárolásához elég az a és b számokat tárolni. A módszer előnye, hogy a permutáció tömören reprezentálható; hátránya, hogy nem csak egy kisebb családból választ permutációt, továbbá a nagy egészekkel való aritmetika lassítja az ujjlenyomatok kiszámítását.

A szakasz befejezéseként egy egyszerű eljárást mutatunk a *hasonlóság-keresési* feladatra, melynek bemenete egy $i_q \in \{1, 2, \cdots, m\}$ adatpontként adott lekérdezés és egy $\alpha > 0$ küszöb, a kimenet pedig az olyan adatpontok halmaza, melyek az α küszöbnél hasonlóbbak i_q-hoz. Formálisan megfogalmazva tehát az $\{\, i \,:\, s_J(i_q, i) > \alpha \,\}$ elemeit keressük. Kézenfekvő, de nem hatékony megoldás, hogy az i_q adatpontot összehasonlítjuk az összes többivel az M' mátrix alapján. Ez a megoldás különösen sokáig tart, ha az M' mátrix csak a háttértáron fér el.

A következő algoritmusban nem követeljük meg, hogy M' mátrix a memóriában legyen. Az M' elemeit most egy adatbázisban tároljuk, amely a háttértárról az $M'[i]$ lekérdezés hatására betölti az i adatponthoz tartozó k darab ujjlenyomatot a memóriába. A hasonlósági lekérdezések előtt elkészítjük az I adatbázist is, amely az $I[\ell][j]$ lekérdezés hatására azon i indexeket adja vissza, amelyeknek ℓ-ik ujjlenyomata j. Az I adatbázis M'-ből úgy számítható, hogy M' elemeit ℓ és j szerint lexikografikusan rendezzük. Háttértáras rendezés alkalmazásával ez akkor is megtehető, ha M' nem fér el a memóriában.

A következő algoritmus egyszer fér hozzá M'-höz, k-szor az I-hez, és eközben megszámolja, hogy a többi adatpontnak az adott i_q-val hány közös ujjlenyomata van. Az egyes adatpontokkal közös ujjlenyomatok számát a H hash-táblában tároljuk azon adatpontok esetén, amelyekkel már találtunk közös ujjlenyomatot. Az alábbi pszeudokódban $H[i] = 0$, ha i nincs benne még a hash-táblában.

HASONLÓ(i_q, α)

```
1  u ← M'[i_q]
2  for ℓ ← 1 to k
3      do for i ∈ I[ℓ][u_ℓ]
4          do H[i] ← H[i] + 1
5  return { i : H[i]/k > α }
```

A dimenzió-csökkentés eredményeként egyrészt sikerült M helyett a k által szabályozható méretű M' mátrixba tömöríteni az adatbázist. Másrészt a hasonlóság keresési problémára is hatékonyan, mindössze $k + 1$ tömbelem hozzáféréssel tudunk válaszolni.

Gyakorlatok

29.4-1. Javasoljunk olyan módszert véletlen ujjlenyomat generálására, hogy az i_1 és i_2 adatpontok esetén az ujjlenyomatok egybeesésének valószínűsége az

$$\frac{|X(i_1) \cap X(i_2)|}{|X(i_1)| \cdot |X(i_2)|}$$

érték legyen.

29.5. Particionáló klaszterező algoritmusok

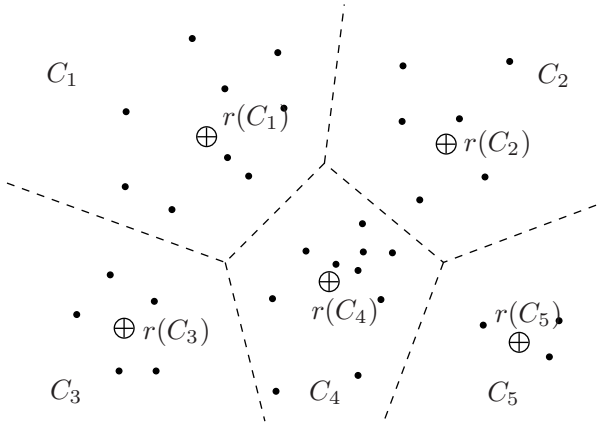
A *particionáló algoritmusok* alapgondolata, hogy a megfelelő klaszterezést a pillanatnyi eredményként kapott klaszterezés folyamatos pontosításával iterálva érjük el. Az iteráció egy fázisában a klasztereket tömören reprezentáljuk úgy, hogy a reprezentáció segítségével a reprezentált klaszterek és az adatpontok közötti távolságokat értelmezni és hatékonyan számolni tudjuk. Ezek után az adatpontokat újból particionáljuk, minden adatpontot hozzárendelünk a hozzá legközelebb eső klaszterhez. Az algoritmus akkor ér véget, ha az iterációs lépés során nem (vagy csak alig) változik meg a partíció, illetve a klaszterek reprezentánsai.

Egy particionáló klaszterező algoritmus elvileg tetszőleges reprezentációból, illetve a hozzátartozó partícióból kiindulhat, de általában célszerű valamilyen egyszerű (klaszterező) heurisztikát alkalmazni a kezdeti klaszter-reprezentánsok meghatározására vagy véletlenszerűen választani azokat. A kezdeti partíció megadásával az osztályok k száma is adott lesz. Ez az szám az itt bemutatandó particionáló klaszterező algoritmusok futása során nem változik.

29.5.1. k-KÖZÉP

A k-KÖZÉP (k-means) algoritmus az egyik legrégebbi és legegyszerűbb klaszterező algoritmus. Feltesszük, hogy az adatpontok egy vektortérben helyezkednek el. Ekkor a klasztereket súlypontjukkal, középpontjukkal (innen az algoritmus elnevezése) reprezentáljuk, azaz az adott klasztert az adatpontjaihoz tartozó vektorok átlagával reprezentáljuk. Az algoritmus olyan C klaszterezést keres, ahol az adatpontoknak a klaszterük $r(C_i)$ reprezentánsától mért

$$E(C) = \sum_{i=1}^{k} \sum_{u \in C_i} d(u, r(C_i))^2$$



29.3. ábra. A k-KÖZÉP klaszterező algoritmus egy fázisa: a C_i klasztereket az $r(C_i)$ súlypontjaik reprezentálják.

négyzetes távolságösszege minimális. Egy adott C klaszterezés esetén az $E(C)$ mennyiségre a későbbiekben a klaszterezés ***hibájaként*** fogunk hivatkozni.

Az algoritmus menete a következő. Legyen k a felhasználó által előre megadott klaszterszám. Ezután k darab tetszőleges elemet választunk az adatpontok vektorterében (célszerűen azok konvex burkában). Az iteráció első fázisában ezek a pontok reprezentálják a kezdeti k darab klasztert. Ezután minden adatpontot a legközelebbi reprezentánssal rendelkező klaszterhez sorolunk be. A besorolás során kialakult klaszterek új reprezentáns pontjai az új klaszterek középpontjai lesznek (lásd a 29.3. ábrát). A besorolás, középpont választás lépéseket addig iteráljuk, amíg a reprezentánsok rendszere változik. Akkor állunk meg, amikor a klaszterek elemei, illetve középpontjai már nem változnak meg az iterációs lépéstől.

Jelölje D az adatpontok halmazát egy euklideszi vektortérben. Legyen k az előre adott klaszterszám. A k-KÖZÉP algoritmus pszeudokódja a következő:

k-KÖZÉP(D, k)

1 $\{r(C_1), r(C_2), \ldots, r(C_k)\} \leftarrow$ reprezentánsok tetszőleges k elemű kezdeti halmaza
2 **while** az $r(C_i)$ reprezentánsok rendszere változik
3 **do for** $i \leftarrow 1$ **to** k
4 **do** $r(C_i) \leftarrow \dfrac{1}{|C_i|} \displaystyle\sum_{u \in C_i} u$
5 **for** minden $u \in D$
6 **do** u legyen az $\operatorname{argmin}_i d(u, r(C_i))$ indexű klaszterben
7 $C \leftarrow$ az új klaszterezés
8 **return** C

Az algoritmus jól alkalmazható, ha a „valódi” klaszterek konvex burkai diszjunktak. Fontos előnye, hogy egyszerűen megvalósítható. Futási ideje $O(nkt)$ darab távolságszámítás, ahol n az adatpontok, k a klaszterek és t az iterációk számát jelöli. Látható, hogy az algoritmus futási ideje nagyban függ az alkalmazott távolságszámítástól. A gyakorlatban az

algoritmus fenti alapváltozata kis (tízes nagyságrendű) k és t értékeket igényel. Az algoritmus további előnye, hogy nem érzékeny az adatpontok sorrendjére és alkalmazható L_p terekben is.

A k-KÖZÉP algoritmus egyik hátránya, hogy az algoritmus a hiba *lokális minimumában* is megállhat, azaz előfordulhat, hogy egy újabb iteráció során nem változik meg a partíció, mégis létezik olyan klaszterezés, aminek hibája kisebb, mint az algoritmus által adott partícióé. Gyakorlatban a lokális optimumba kerülés esélyének csökkentése érdekében érdemes az algoritmust többször futtatni különböző kezdeti középpontokkal. Azt a klaszterezést fogadjuk el a többszöri futások végeredményei közül, amelyik hibája a legkisebb hibát adja.

Megmutatjuk, hogy a k-KÖZÉP algoritmus érzékeny a kívülálló pontokra. Ebből a célból bevezetjük a Voronoi-tartomány fogalmát. Egy ponthalmaz u pontjának ***Voronoi-tartománya*** álljon azon pontokból, amelyeknek u-tól vett távolsága nem nagyobb, mint bármely más ponthalmazbeli ponttól vett távolsága. A k-KÖZÉP algoritmus során a klaszterek a reprezentáló középpontok által kialakított konvex *Voronoi-tartományokba* esnek, ahogyan ez a 29.3. ábrán is látható. Mivel a klaszterek a Voronoi-tartományok belsejében alakulnak ki, ezért a k-KÖZÉP eredményében a klaszterek konvex burkai diszjunktak lesznek. Az algoritmus tehát a nyilvánvalóan kívülálló pontokat is besorolja valamelyik klaszterbe, és ezek a pontok a klaszter-középpont meghatározásakor indokolatlan torzító hatást okoznak.

További problémák az algoritmussal, hogy a megfelelő k klaszterszám megtalálása többszöri futtatást igényelhet, az iterációk során szélsőséges klaszterméretek alakulhatnak ki és a végeredmény nagyban függ a kezdeti partíciótól.

A k-KÖZÉP egyik, a gyakorlatban jobb minőségű eredményt adó módosítása a következő. A pontokat egy rögzített sorrend szerint ciklikusan vizsgálva meghatározzuk a vizsgált ponthoz legközelebbi klaszter-középpontot, majd ha szükséges, átsoroljuk az adatpontot a hozzá közelebbi középpontú klaszterbe. Átsorolás esetén kiszámoljuk a két érintett klaszter új középpontjait. Az alapváltozathoz hasonlóan akkor állunk meg az algoritmussal, ha már nem változnak a klaszterek. Az eljárás pszeudokódja a következő.

k-KÖZÉP VÁLTOZAT(D, k)

1 $\{r(C_1), r(C_2), \ldots, r(C_k)\} \leftarrow$ reprezentánsok tetszőleges k elemű kezdeti halmaza
2 **while** $r(C_i)$ reprezentánsok változnak
3 **do for** minden $u \in D$ adatpontra egy rögzített sorrend szerint
4 **do** $h \leftarrow u$ klaszterének indexe
5 $j \leftarrow \operatorname{argmin}_i d(u, r(C_i))$
6 **if** $h \neq j$
7 **then** $C_j \leftarrow C_j \cup \{u\}$
8 $C_h \leftarrow C_h \setminus \{u\}$
9 $r(C_j) \leftarrow \dfrac{1}{|C_j|} \displaystyle\sum_{v \in C_j} v$
10 $r(C_h) \leftarrow \dfrac{1}{|C_h|} \displaystyle\sum_{v \in C_h} u$
11 **return** C

Meglepő módon euklideszi vektortérben adott pontok esetén a fenti k-KÖZÉP VÁLTOZAT futtatása ugyanannyi távolságszámítást igényel, mint az k-KÖZÉP alapváltozat. További al-

goritmus változathoz jutunk, ha a pszeudokód 3. sorát megváltoztatva a pontokat nem egy rögzített sorrend szerint, hanem véletlenül választva vizsgáljuk.

29.5.2. *k*-MEDOID

A *k*-KÖZÉP klaszterező algoritmus csak olyan adatpontok csoportosítására használható, ahol értelmezhető a klaszterek középpontja, például a pontok vektortérben vannak megadva. Így a *k*-KÖZÉP nem használható olyan alkalmazásokban, ahol az adatpontok nem attribútumokkal adottak, vagy amikor az adatpontokat leíró attribútumok között kategorikus értékek is szerepelnek. Ilyen helyzetekben a *k*-MEDOID algoritmus használható, mert futása során csak az adatpontok közötti távolságok összehasonlítását használja.

A *k*-MEDOID módszer a klasztereket egy-egy, az adott klaszter elemeihez legközelebbi adatponttal, a ***medoiddal*** reprezentálja. A *k*-KÖZÉP módszerhez hasonlóan itt is egy távolságnégyzetek összegén alapuló függvényt minimalizálunk, amit az összes adatpont és klaszterük medoidjának távolságnégyzetét összegezve számolunk ki:

$$E(C) = \sum_{i=1}^{k} \sum_{u \in C_i} d(u, m_i)^2 \ ,$$

ahol m_i a C_i klaszter medoidja és d az adatpontokon értelmezett metrika.

A *k*-MEDOID algoritmus menete megegyezik a *k*-KÖZÉP menetével. Kezdetben választunk k darab tetszőleges adatpontot, ezek lesznek először a medoidok. Ezután elkezdődik az adatpontok medoidokhoz történő hozzárendelése, illetve az új medoidok választásából álló iteratív folyamat. Az iteráció során a *k*-KÖZÉP algoritmus módosításánál látott stratégiához hasonló módszert célszerű követni. Tekintsünk egy véletlen nem medoid v adatpontot és vizsgáljuk meg, hogy van-e olyan klaszter, amelyet a v pont jobban reprezentál, mint a klaszter jelenlegi medoidja (az új medoid nem feltétlenül a klaszter eleme, de klaszterezendő adatpont). Ezt a lépést a következőképpen valósíthatjuk meg. Az összes m_i medoidra meghatározzuk, hogy mennyivel változna a klaszterezés jósága, ha v átvenné az m_i szerepét. Az algoritmus menetét egyszerűsíti, hogy a hiba változására azok az elemek nincsenek hatással, amelyekhez létezik v-nél és m_i-nél is közelebbi medoid. Ha találunk olyan m_i-t, amellyel v-t cserélve $E(C)$ csökken, akkor javítunk a klaszterezésen; azt a cserét hajtjuk mindig végre, amellyel pillanatnyilag a legnagyobb csökkenést érjük el.

A pszeudokódból látható, hogy a számoláshoz csak az adatpontok egymástól mért távolságainak értékeit használjuk. Az iteráció egy fázisához $O(nk^2)$ távolság kiszámítása kell, mert minden potenciális m_i-ről v-re történő medoid csere esetén a hiba változásának kiszámolásához $n \times k$ darab pont-medoid pár távolságára van szükségünk. A teljes futási időt az határozza meg, hogy az adatpontok számának növekedésével lineárisan nő a fázisok szükséges száma. Fontos előnye a *k*-MEDOID módszernek, hogy a *k*-középhez képest kevésbé érzékeny a kívülálló pontokra.

A pszeudokód a következő:

k-MEDOID(D, k)

```
1  Adjunk meg k darab tetszőleges m_i ∈ D (i = 1,...,k) adatpontot.   ▷ A első medoidok.
2  while C_i osztályok változnak
3        do for minden u ∈ D adatpont
4              do u-t tegyünk be az j = argmin_i d(u, m_i) indexű C_j klaszterbe.
5           v ← véletlen adatpont, ahol v ≠ m_i minden i-re
6           for i ← 1 to k
7              do Δ_i ← az E(C) hiba változása, ha az m_i medoidot v-re cseréljük
8           if min_i Δ_i < 0
9             then v legyen medoid m_j helyett, ahol j = argmin_i Δ_i
10          C ← az új klaszterezés
11 return C
```

Gyakorlatok

29.5-1. Mutassunk példát arra, hogy a k-KÖZÉP algoritmus által kimenetként adott klaszterezés hibája nem a hiba globális minimuma.

29.6. Hierarchikus eljárások

A *hierarchikus* eljárások onnan kapták a nevüket, hogy az elemeket egy hierarchikus adatszerkezetbe (fába, dendogramba, taxonómiába) rendezik el. Az adatpontok a fa leveleiben találhatók. A fa minden belső pontja egy klaszternek felel meg, amely azokat a pontokat tartalmazza, melyek a fában alatta találhatók. A gyökér az összes adatpontot tartalmazza.

29.6.1. Felhalmozó és lebontó módszerek

A hierarchikus klaszterezési módszereket két csoportra bonthatjuk: a ***felhalmozó*** (egyesítő, bottom-up) és ***lebontó*** (felosztó, top-down) módszerek. A lentről felfelé építkező *felhalmozó* eljárásoknál kezdetben minden elem különálló klaszter, majd az eljárás a legközelebbi klasztereket egyesíti, a hierarchiában egy szinttel feljebb újabb klasztert alakítva ki. A fentről lefelé építkező *lebontó* módszerek fordítva működnek: egyetlen, minden adatpontot tartalmazó klaszterből indulnak ki, amit kisebb klaszterekre particionálnak, majd ezeket is tovább bontják.

A kapott hierarchiából számos klaszterezést tudunk kinyerni, amennyiben például a futás során kialakult a fa egy-egy szintjén található pontokat tekintjük. Az így kinyert klaszterezések egymás finomításai lesznek. Általánosabban választhatjuk a fa gyökere és levelei közötti tetszőleges, tartalmazásra minimális elvágó ponthalmaz elemeit; az ezekhez rendelt klaszterek a ponthalmaz egy partícióját alkotják.

Általában tetszőleges hasonlóság vagy távolság fogalomra építve könnyen kidolgozható hierarchikus algoritmus, így bármilyen típusú adat klaszterezéséhez több rugalmas és többnyire hatékony módszerhez juthatunk.

A hierarchikus algoritmusok kulcslépése az egyesítendő vagy osztandó klaszterek kiválasztása. Miután egy *egyesítés* (*osztás*) megtörténik, minden további műveletet az új klasztereken végezzük el. Tehát egyik művelet sem fordítható meg, így egy rossz választás később

nem javítható ki.

Nehézséget jelent annak eldöntése, hogy mennyire, milyen magas szintig épüljön fel a hierarchikus klaszterezés fája, illetve az elkészült dendogram melyik vágásával tudjuk a feladatot legjobban megoldó klaszterezést kiválasztani.

A legegyszerűbb felhalmozó hierarchikus klaszterező eljárás az alábbi. Legyen D az adatpontok halmaza, $d_*(C_i, C_j)$ két klaszter távolságát mérő függvény és $S(C)$ egy megállási szabály. Kezdetben minden pont különálló egyelemű klaszterhez tartozik. Keressük meg, majd egyesítsük a két, d_* szerint legközelebbi klasztert. Iteráljuk ezt a lépést, amíg $S(C)$ meg nem állítja az eljárást. Például $S(C)$ definiálható úgy, hogy az algoritmus akkor álljon meg, amikor a $|C|$ vagy d_* aktuális értéke elér egy előre adott küszöböt.

EGYSZERŰ-FELHALMOZÓ(D, S)

```
1  C ← {{u}|u ∈ D}
2  while S(C) engedi
3        do (a, b) ← argmin_(i,j) d_*(C_i, C_j)
4           C ← C ∪ {C_a ∪ C_b} \ {C_a, C_b}
5  return C
```

29.6.2. Klasztertávolságok mértékei

A d_* klaszterek távolságát mérő függvény megválasztásától függően különböző hierarchikus klaszterező algoritmusokat kapunk. Mivel a klaszterezés bemenete az adatpontok d távolságfüggvénye, ezen értékekből számíthatjuk d_*-ot. A továbbiakban d_* leggyakrabban használt megvalósításait mutatjuk be.

Amennyiben két klaszter d_{min} távolságát azok legközelebbi pontjainak távolságával definiáljuk, akkor ***legkisebb távolságot*** (single linkage) használó eljárásról beszélünk. Ekkor

$$d_{min}(C_i, C_j) = \min_{\substack{u \in C_i \\ v \in C_j}} d(u, v) \ .$$

Általános esetben, távolságmátrixot használva az eljárás $O(n^2)$ összehasonlítást igényel.

A legkisebb távolságon alapuló algoritmusok alkalmasak jól elkülönülő, tetszőleges alakú klaszterek felfedezésére. Mivel a klaszterelemek minimális távolságát veszik figyelembe, hajlamosak azonban összekötni két klasztert, ha egy-egy elemük túl közel kerül egymáshoz. További hibája a módszernek, hogy érzékeny a kívülállókra.

Ha egy legkisebb távolságon alapuló klaszterzést gráfelméleti algoritmusként tekintünk úgy, hogy a gráf pontjai az adatpontok és az éleken adott költségeknek pedig a távolságok felelnek meg, akkor algoritmusunk a minimális költségű feszítőfát építő *Kruskal-algoritmussal* lesz azonos. A Kruskal-algoritmus megvalósításához nagyon hatékony adatstruktúrák ismertek. Amennyiben a távolságok *ritka mátrix formátumban* adottak, az algoritmus $O(e\alpha(e))$ futási idővel is megvalósítható, ahol e az élek száma és α az inverz Ackermann-függvényt[1] jelöli.

[1] Az $A(n) = A(n, n)$ Ackermann-függvény a következő rekurzióval adható meg:

$$A(n, m) = \begin{cases} m + 1, & \text{ha } n = 0, \\ A(n-1, 1), & \text{ha } n > 0 \text{ és } m = 0, \\ A\left(n-1, A(n; m-1)\right), & \text{ha } n > 0 \text{ és } m > 0 \ . \end{cases}$$

Ha a

$$d_{max}(C_i, C_j) = \max_{\substack{u \in C_i \\ v \in C_j}} d(u, v)$$

klasztertávolságot használjuk, akkor a ***legnagyobb távolságon*** (complete linkage) alapuló eljárást kapjuk. További klaszterek közötti távolságfüggvényt ad a

$$d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\substack{u \in C_i \\ v \in C_j}} d(u, v)$$

átlagos távolság.

Ha az adatpontok vektortérbeliek és d a vektortér metrikája, akkor d_* a klaszterközéppontok távolságaként értelmezhető:

$$d_{mean}(C_i, C_j) = d\left(\frac{1}{|C_i|}\sum_{u \in C_i} u, \frac{1}{|C_j|}\sum_{v \in C_j} v\right).$$

További hagyományos klasztermetrika az átlagos távolsághoz hasonló ***Ward-távolság***

$$d_{Ward}(C_i, C_j) = \sum_{u,v \in C_i \cup C_j} d^2(u, v) - \left(\sum_{u,v \in C_i} d^2(u, v) + \sum_{u,v \in C_j} d^2(u, v)\right).$$

Ward módszere a klasztereken belüli távolságnégyzetek összegének minimalizálását célozza. Azt a két klasztert egyesíti, amelyek a legkisebb négyzetes hibanövekedést okozzák.

A fenti klasztertávolságok mindegyike alkalmazható, mint az EGYSZERŰ-FELHALMOZÓ hierarchikus algoritmus d_* függvénye. Ehhez hasonlóan dolgozhatók ki a lebontó hirarchikus algoritmusok részletei.

Gyakorlatok

29.6-1. Legyen f monoton növő függvény. Mutassuk meg, hogy az $f(d_{min})$, illetve az $f(d_{max})$ klasztertávolságon alapuló hierarchikus klaszterező módszer esetén ugyanazt az eredményt kapjuk, mint a d_{min}, illetve a d_{max} klasztertávolság alkalmazása esetén.

29.6-2. Vektortérbeli adatpontok hierarchikus klaszterezését gyorsítják az úgynevezett CF-hármasok. Egy C klasztert egy $CF(C) = (s_0, \vec{s_1}, s_2)$ hármas jellemezzen, ahol $s_0 = |C|$ a klaszterben található adatpontok száma, $\vec{s_1} = \sum_{u \in C} u$ az adatpontok vektoriális összege és $s_2 = \sum_{u \in C} \|u\|^2$ az adatpontok normáinak négyzetösszege. Mutassuk meg, hogy a két klaszter CF-hármasából kiszámítható a két klaszter

a. uniójának CF-hármasa,

b. átlagos távolsága,

c. Ward-távolsága.

29.6.3. A ROCK algoritmus

A korábban tárgyalt klaszterező eljárások – a k-MEDOID kivételével – numerikus attribútumokkal rendelkező adatpontokra lettek kitalálva. A ROCK felhalmozó hierarchikus klaszterező algoritmus kifejezetten olyan kategorikus attribútumokkal jellemezhető adatok klaszterezésére szolgál, mint amilyenek a bináris attribútumokkal leírható *tranzakciók*. A ROCK

Az $\alpha(e) = \min\{n : A(n) \geq e\}$ inverz Ackermann-függvény rendkívül lassan tart a végtelenbe, a gyakorlatban előforduló esetekben értéke legfeljebb 4 lesz.

az Egyszerű-felhalmozó algoritmus sémáját követi. Specialitása a d_* klaszterek távolságát mérő függvény és az S megállási szabály megvalósításában van.

A Rock algoritmus az adatpontok közötti *kapcsolat* fogalma köré épül fel. Legyen ε az algoritmus paramétere. Az u és v pontok ***kapcsolatát*** közös szomszédainak

$$\ell(u,v) = |N_\varepsilon(u) \cap N_\varepsilon(v)|$$

száma adja. Két klaszter ***kapcsolat alapú hasonlóságát*** az

$$\ell(C_i, C_j) = \sum_{\substack{u \in C_i \\ v \in C_j}} \ell(u,v)$$

mennyiséggel, egy klaszter ***belső kapcsolatát*** az

$$\ell(C) = \ell(C,C) = \sum_{u,v \in C} \ell(u,v)$$

mennyiséggel definiáljuk.

A Rock egyik alapötlete, hogy mivel az $\ell(C_i, C_j)$ mennyiség érzékeny a C_i, C_j klaszterek méretére, az algoritmus az ℓ függvény egy normalizált változatát használja a klaszterek hasonlóságának mérésére. A következőkben egy olyan $h(n_i, n_j)$ mennyiséget definiálunk, amely két, n_i, illetve n_j méretű „átlagos” klaszter kapcsolat alapú távolságát modellezi. A h függvény meghatározásához a következő észrevételt használjuk, amelynek bizonyítását az Olvasóra hagyjuk (29.6-2. gyakorlat).

29.8. állítás. *Legyen C_i és C_j két diszjunkt klaszter. Ekkor*

$$\ell(C_i \cup C_j) = \ell(C_i) + \ell(C_j) + \ell(C_i, C_j)\ .$$

Ennek analógiájára megköveteljük, hogy $h(n_i, n_j) = h(n_i + n_j) - h(n_i) - h(n_j)$ is igaz legyen, ahol $h(n_i)$ az „átlagos” n_i méretű klaszter belső kapcsolata.

A Rock algoritmusban a $h(n_i) = n^{1+2f(\varepsilon)}$ függvény modellezi az átlagos klaszter belső kapcsolatfüggvényét, ahol $f : [0,1] \to [0,1]$ alkalmasan választott függvény. Például, ha tranzakciós adatok Jaccard-hasonlósággal adottak, akkor az $f(\varepsilon) = (1-\varepsilon)/(1+\varepsilon)$ függvény javasolható. Így két klaszter hasonlóságát az

$$s_{\text{Rock}}(C_i, C_j) = \frac{\ell(C_i, C_j)}{h(n_i, n_j)}$$

alakban írjuk fel, ahol $n_i = |C_i|$ és $n_j = |C_j|$.

A Rock algoritmusnak, mint hierarchikus módszernek minden klaszterösszevonási lépés előtt $O(n^2)$ lehetséges klaszterpár közül kell választania. A hierarchia felépítéséhez $O(n)$ összevonás szükséges, így a naív megvalósítás futási ideje $O(n^3)$ lesz. Gyakorlatban az adatpontok ℓ kapcsolatának kiszámítása, illetve a klaszterek összevonásának lépése adatstruktúrák alkalmazásával gyorsítható, így az algoritmus futási ideje hatékonyan csökkenthető.

Gyakorlatok

29.6-1. Mutassuk meg, hogy két klaszter kapcsolat alapú távolsága kiszámolható a következő módon:

$$\ell(C_i, C_j) = \sum_{u \in D} |N_\varepsilon(u) \cap C_i| \cdot |N_\varepsilon(u) \cap C_j|\ ,$$

ahol az összegzés az adatpontok D halmazán történik.

29.6-2. Bizonyítsuk be az 29.8. állítást.

29.7. Sűrűség alapú eljárások

Síkban, térben vizualizált adatpontokra „szemmel ránézve” tetszőleges formájú klasztereket tudunk azonosítani, ha a klaszteren belüli pontok megfelelő méretű (sugarú) környezete jellemzően több pontot tartalmaz, mint a klaszter határán, illetve általában a klaszterek között találhatóké. Ugyanakkor a kívülálló pontok is felismerhetők ritkább környezetükről. Ez az észrevétel adja a sűrűség alapú klaszterező eljárások alapötletét.

Egy ***sűrűség alapú*** klaszterező módszer a következő elemekből épül fel. Először az adatpontokon megadunk egy reflexív és szimmetrikus szomszédsági relációt. Ez a reláció meghatározza az adatpontok *környezetét*. A gyakorlatban a relációt úgy választjuk meg, hogy az minél jobban igazodjon a környezet esetleg előre adott intuitív fogalmához. Második lépésben lokális tulajdonságok alapján minden adatpontról eldöntjük, hogy az egy klaszter *belső pontja* lesz-e, azaz részt vesz-e a klaszter kialakításában, kohéziójában. A belső pontok meghatározása független lehet a korábban definiált szomszédsági relációtól. A következő lépésben a belső pontokra bevezetjük a szomszédsági reláció tranzitív lezártját. Ez az új ekvivalencia reláció alakítja ki a belső pontok partícióját. A többi, nem belső adatpontot a szomszédsági reláció segítségével (nem feltétlenül egyértelműen és teljesen) hozzárendeljük a klaszterekhez, mint azok határpontjai. A kimaradt adatpontok adják a *kívülállók* halmazát.

Egy *metrikus térbeli* ponthalmaz esetén adott sugarú gömbbel értelmezhetjük a pontok környezetét, és egy küszöbnél nagyobb számosságú környezettel rendelkező pontokat tekinthetjük belső pontoknak. Az így kapott sűrűség alapú módszer előnye robusztussága az adatpontok (pontosabban távolságaik) perturbációjával szemben. Az algoritmus időkomplexitása lényegesen függ a környezetek generálásától, például az itt szubrutinként használható *legközelebbi adatpontot* kiválasztó operáció megvalósításától.

29.7.1. A Dbscan algoritmus

A Dbscan algoritmus a sűrűségen alapú klaszterező módszerek legfontosabb példája. A Dbscan, mint sűrűség alapú algoritmus a *környezet* és *belső pont* fogalmát az alábbi módon definiálja. Legyen D a klaszterezendő adatpontok halmaza egy d metrikával ellátva. Az adatpontokat $u, v, \ldots$ betűkkel jelöljük. Legyen ε egy előre adott paraméter. Az ε paraméter segítségével fogjuk meghatározni az algoritmusban használt környezet fogalmát.

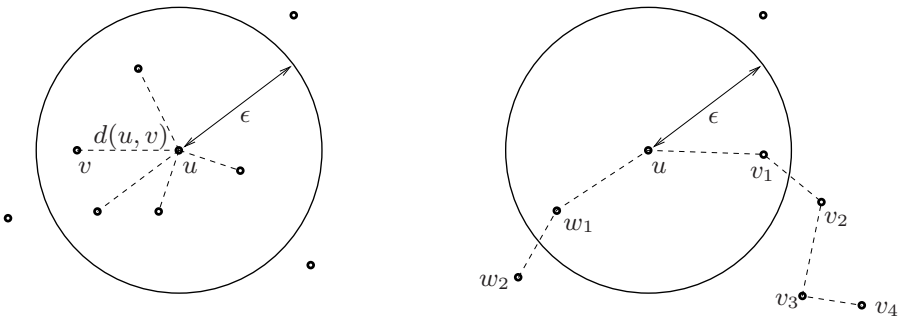
29.9. definíció. *Egy $u \in D$ pont ε sugarú **környezete** legyen $N_\varepsilon(u) = \{v \in D | d(u,v) \leq \varepsilon\}$. Ekkor $N_\varepsilon(u)$ elemeit u **szomszédainak** hívjuk.*

Az algoritmus további paramétere az alkalmasan választott μ ***sűrűségi korlát.*** A következő definícióval a klaszterek belső pontjait határozzuk meg.

29.10. definíció. *Egy $u \in D$ pont **sűrűségén** szomszédainak $|N_\varepsilon(u)|$ számát értjük. Egy $u \in D$ pontot **magpontnak** nevezünk, ha $|N_\varepsilon(u)| \geq \mu$.*

A klasztereket az alábbi definíciók segítségével fogjuk kialakítani.

29.11. definíció. *Egy $u \in D$ pont **közvetlenül elérhető** a $v \in D$ pontból, ha v magpont és u szomszédja v-nek. Egy u pont **elérhető** a v pontból, ha létezik a $v = v_1, v_2, \ldots, v_n, v_{n+1} = u$ $(v \geq 0)$ pontok olyan sorozata, hogy minden $i = 1, \ldots, n$ esetén v_{i+1} közvetlenül elérhető*



29.4. ábra. Illusztráció a 29.1. példához.

v_i-ből. Végül az u_1 és u_2 pontok ***összekapcsoltak,*** *ha létezik olyan v pont, ahonnan u_1 és u_2 is elérhető.*

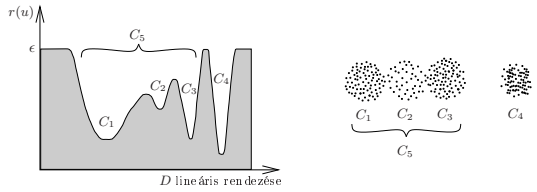
29.1. példa. A 29.4. ábra bal oldalán $|N_\varepsilon(u)| = 7$. Ha például $\mu = 5$, akkor az u magpont. Az ábrán az u-ból közvetlenül elérhető pontok szaggatott vonallal kapcsolódnak u-hoz. A jobboldali ábrán u-ból elérhetők a $v_1, \ldots, v_4$, illetve a w_1, w_2 pontok, így v_4 és w_2 összekapcsoltak.

Az algoritmussal olyan C klasztereket szeretnénk előállítani, melyek egyrészt összefüggőek, azaz minden $u, v \in C$ pont összekapcsolt, másrészt maximálisak, azaz ha $u \in C$ és v elérhető u-ből, akkor $v \in C$ is teljesül. Azokat az adatpontokat, amelyek nem érhetők el egyik klaszterből sem (és így nem is magpontok), *kívülállóknak* tekintjük.

Az Dbscan algoritmus menete a következő. Válasszunk egy tetszőleges $u \in D$ pontot. Ha u magpont, akkor határozzuk meg az u-ból elérhető pontokat, ezek fogják a C_u klasztert alkotni. Ha u nem magpont, válaszunk egy új, még nem vizsgált pontot. Ha már nem tudunk új pontot választani, akkor az algoritmus véget ér. A kívülálló elemek D' halmazát azok a pontok fogják alkotni, amelyeket nem soroltunk egyik klaszterbe sem. Az algoritmus pszeudokódja a következő oldalon van.

Látható, hogy minden pont környezetét elegendő egyszer megvizsgálni az algoritmus futása alatt. Egy környezet megvizsgálása általános esetben n távolság lekérdezését jelenti, ami $O(n^2)$-es futási időre vezet. Számos esetben (például alacsony dimenziós euklideszi vektortereken) megfelelő adatstruktúrával a környezetek kiszámítása $O(\log n)$ lépésben megoldható, ekkor a Dbscan futási ideje $O(n \log n)$ lesz.

A Dbscan algoritmus előnye, hogy tetszőleges alakú klasztereket képes felfedezni. Hátránya, hogy érzékeny a felhasználó által megadott két paraméterre (ε és μ). Amennyiben a klaszterekben található elemek sűrűsége eltérő, akkor még az sem biztos, hogy létezik olyan paraméterezés, amellyel a Dbscan jó eredményt ad.



29.5. ábra. Az Optics algoritmus kimenetének egy lehetséges vizualizációja. A bal oldali függvényábrázoláson $r(u)$ lokális minimumai jelzik a klasztereket.

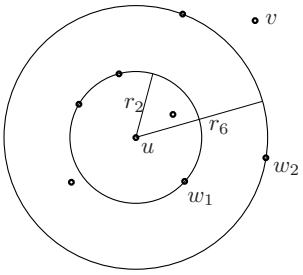
Dbscan(D, ε, μ)

```
1  C ← ∅
2  D' ← ∅
3  while D ≠ ∅
4        do u ← D egy tetszőleges eleme
5           if u magpont
6              then C_u ← {v ∈ D ∪ D' | v elérhető u-ból}
7                   C ← C ∪ {C_u}
8                   D ← D \ C_u
9                   D' ← D' \ C_u
10             else D' ← D' ∪ {u}
11                  D ← D \ {u}
12 return C, D'
```

29.7.2. Az Optics algoritmus

A szintén a sűrűség fogalmára építő Optics eljárás (lásd a 29.5. ábrát) a Dbscan ε és μ paraméterekre való érzékenységét küszöböli ki. Bár az Optics is használja ezeket a paramétereket, kimenete a Dbscan algoritmushoz képest kevésbé érzékeny értékükre. Az Optics algoritmus kimenetéből több klaszterezést is megkaphatunk. Az algoritmus listába rendezi az adatpontokat és midnen egyes adatpontra meghatároz a pont környezetét jellemző bizonyos numerikus értékeket. Ezek segítségével azután hatékonyan megjeleníthetjük a teljes adathalmazt, illetve megkaphatjuk a Dbscan minden $\varepsilon' \leq \varepsilon$ értéknél kisebb paraméteréhez tartozó kimeneti klaszterezést és jellemezhetjük azok kapcsolatát.

Az algoritmus a Dbscan tárgyalásánál bevezetett fogalmakon túl a következő definíciókat használja. Egy $u \in D$ pont ***k*-sugara** az az r szám, amelyre igaz, hogy legalább k olyan $w \in D \setminus \{u\}$ pont van, amelyre $d(u,w) \leq r$, és legfeljebb $k-1$ olyan $w \in D \setminus \{u\}$ pont van, amelyre $d(u,w) < r$. Ha egy előre megadott ε paraméter esetén $N_\varepsilon(u) < k$, akkor végtelennek tekintjük az u pont k-sugarát. Amikor a μ paraméter adott, akkor egy $u \in D$ pont μ-távolságát ***magsugárnak*** hívjuk. Egy $v \in D$ pont $u \in D$ magponttól tekintett ***elérhető távolsága*** pedig legyen $r_u(v) = \max\{u \text{ magsugara}, d(u,v)\}$.



29.6. ábra. Illusztráció a 29.2. példához.

29.2. példa. A 29.6. ábrán az u pont 2-távolsága r_2, a 6-távolsága pedig r_6. Ha $\mu = 6$, akkor a v pont u-tól mért elérhető távolsága $d(u, v)$; míg a w_1 és w_2 pontok u-tól mért elérhető távolsága r_6 magsugár lesz.

Az Optics algoritmus kimenete

- a D-beli adatpontok $\sigma : \{1, \ldots, |D|\} \to D$ lineáris rendezése;
- minden u adatponthoz annak $m(u)$ magsugara; valamint
- a pont környezetét jellemző $r(u)$ érték, amely egy alkalmasan választott másik ponttól tekintett elérhetőségi távolság lesz.

Az algoritmus fő ciklusa korábban nem vizsgált magpontot keres, majd a Bővít szubrutin során kiszámolja a magpontból elérhető klasztert. A Dbscan-től való lényeges eltérés az új klaszter bejárásában jelentkezik. Míg a Dbscan tetszőlegesen választ a közvetlenül elérhető pontokból, addig az Optics mindig a pillanatnyilag elérhető legnagyobb sűrűségű környezetből választ. Az algoritmus pszeudokódja a következő.

Optics(D, ε, μ)

```
 1  s ← 0                                  ▷ A kimeneti sorrend mutatójának kezdeti értékadása.
 2  while D ≠ ∅
 3        do u ← D egy tetszőleges eleme
 4           D ← D \ {u}
 5           σ(u) ← s
 6           s ← s + 1
 7           r(u) ← ε                      ▷ Az elérhető távolsághoz itt még nincs referenciapont.
 8           if u magpont
 9              then Bővít (D, ε, μ, s, u)
10  return σ,r
```

Bővít$(D, \varepsilon, \mu, s, u)$

```
1  B ← N_ε(u) \ {u}                                   ▷ Bővítési halmaz kezdetben.
2  D ← D \ N_ε(u)
3  while B ≠ ∅
4        do for minden v ∈ B
5              do r_i(v) ← max (d(v, σ(i)), σ(i) magsugara)
               ▷ A v pont aktuális elérhetőségi távolságai, végtelen ha σ(i) nem magpont.
6                 r(v) ← min_{i=0,...,s−1} r_i(v)
7           w ← argmin_{v∈B} r(v)
8           σ(w) ← s
9           s ← s + 1
10          if w magpont
11            then B ← B ∪ N_ε(w)
12                 D ← D \ N_ε(w)
13          B ← B \ {w}
14 return D, s, σ, r
```

A Bővít algoritmus 5. sorának megvalósítása során az $r(v)$ értékek minden, a σ sorrend végére kerülő új w adatpont környezetének lekérdezésével frissíthetőek. Így az Optics futása során egy adatpont környezetét legfeljebb kétszer kell lekérdezni, ezért a Dbscan esetében kapott $O(n^2)$, illetve az adatstruktúrák alkalmazásával elérhető $O(n \lg n)$ futási idő itt is érvényes. Látható az is, hogy az algoritmus futási ideje az ε és μ paraméterek függvényében monoton módon változik. Az ε paraméter a B bővítési halmaz méretén keresztül (lásd a Bővít algoritmus 3. sorát) döntően befolyásolja az egész Optics algoritmus futási idejét. Gyakorlati tapasztalatok szerint a μ paraméter értékének megváltoztatása az algoritmus kimenetét jellemzően csak kis (bár természetesen az adathalmaztól függő) mértékben változtatja meg.

Gyakorlatok

29.7-1. Az Optics algoritmus Bővít szubrutinjában (a pszeudokód 6. sorában) definiált $r(v)$ érték miért lesz véges?

29.7-2. Mutassuk meg, hogy az Optics kimenetéből (a klaszterek határpontjaitól eltekintve) megkapható tetszőleges $\varepsilon' \leq \varepsilon$ paraméterrel futtatott Dbscan által kimenetként adott klaszterezés.

Feladatok

29-1. A k-középpont probléma

Tekintsük egy d metrikával ellátott metrikus térben n pontot. Jelölje D a pontok halmazát. A k-középpont probléma megoldása a pontoknak az a k elemű S halmaza, amely minimalizálja a

$$\max_{u \in D} \min_{s \in S} d(u, s)$$

mennyiséget. Mutassuk meg, hogy a k-középpont probléma NP-nehéz! (*Útmutatás.* Vezessük vissza a gráfok domináns halmazának NP-nehéz problémáját a k-középpont problémára a 29.3-1. gyakorlatban szereplő metrika segítségével!)

29-2. Mintavételezés

Tegyük fel, hogy az adathalmaz egyforma klaszterekből áll. Mekkora véletlen mintát kell ahhoz venni, hogy legfeljebb 0.01 valószínűséggel veszítsünk el klasztert, azaz annak a valószínűsége, hogy a minta egyetlen egy elemet sem tartalmaz valamelyik klaszterből, legfeljebb 0.01 legyen?

Most tegyük fel, hogy az s méretű ($s = 1, 2, \ldots$) klaszterek száma egy hatványfüggvénnyel arányos, azaz $|\{C : |C| = s\}| \sim s^{-\alpha}$, ahol $\alpha > 1$. Mekkora minta szükséges, ha 0.01 valószínűséggel szeretnénk mintát kapni legalább a klaszterek feléből? (*Útmutatás.* Használjuk a Chernoff-egyenlőtlenséget!)

Megjegyzések a fejezethez

Az elmúlt évtizedekben számos monográfia készült a klaszterezés témakörében: Anderberg [17], Hartigan [158], Jain és Dubes [177], Kaufman és Rousseeuw [189]. A korábbi művek főleg a statisztika és a gépi tanulás szemszögéből vizsgálják a klaszterezést, az újabb összefoglalásokban jelenik meg az adatbányászati szemlélet. Frissebb összefoglalások a Jain, Murty és Flynn [176], Berkhin [38] áttekintő cikkek, illetve Han és Kamber [155] könyvének klaszterezésről szóló fejezete.

Ezzel a könyvvel egy időben jelennek meg a Khosrow-Pour [197], illetve a Wang [351] által szerkesztett enciklopédiák, amelyek gazdag anyagot tartalmaznak a klaszterezés gyakorlati kérdéseiről.

A Bell-számok pontos aszimptotikája megtalálható Lovász feladatgyűjteményének [233] első fejezetében. A klaszterezés korlátait mutató 29.4. tétel forrása Kleinberg [201].

A dimenzió-csökkentés keretében a 29.4.1. pontban ismertetett szinguláris felbontás a lineáris algebra egyik ismert, alapvető eszköze. A felbonthatóságról szóló 29.5. tétel bizonyítása, illetve a felbontás további alkalmazásairól olvashatunk Rózsa [302] könyvében magyarul, illetve Golub és Van Loan [139] művében. Utóbbi alkotás betekintést nyújt a felbontás kiszámítására használt numerikus módszerekbe is. A felbontással történő közelítés optimalitásáról szóló 29.6. tétel általánosításának bizonyítása, illetve mátrix normák és a szinguláris felbontás kapcsolatáról részletesen olvashatunk Stewart és Sun [321] monográfiájában. Szöveges dokumentumok dimenzió-csökkentésére először Deerwester, Dumais, Landauer, Furnas és Harshman [84] használták a felbontást, azóta számos cikk foglalkozik ezzel az alkalmazással.

A 29.4.2. pontban ismertetett ujjlenyomat alapú dimenzió-csökkentést szintén szöveges dokumentumokra Broder [53] vezette be. Az itt tárgyalt hasonlóság keresési eljárást és weboldalakra történő alkalmazását Haveliwala, Gionis és Indyk [159] tárgyalja részletesen.

A k-KÖZÉP algoritmus egyik korai megfogalmazása MacQueentől [238], a fejezetben említett módosítás Hartigantól [158] származik. A módosítás futási idejét Berkhin és Becher [39] vizsgálta. A k-MEDOID eljárás PAM algoritmus néven Kaufman és Rousseeuw [189] munkája. További particionáló algoritmus az adatokról valószínűségi modellt feltevő és a k-KÖZÉP algoritmus általánosításának tekinthető Lauritzentől [223] származó EM (Expectation-Maximization) algoritmus.

Néhány dimenziós vektortérben reprezentálható adatpontok és klaszterreprezentánsok esetében a *kd*-fa adatstruktúra jól használható az algoritmusok skálázására, lásd Moore és Pelleg [272]. A *k*-KÖZÉP algoritmus számos, például a klaszterszámot futás közben változtatni, vagy a kívülállók kezelésre képes variánsa is ismert (lásd [238] és [17]).

A hierarchikus klaszterezés módszerét Kaufman és Rousseeuw [189] tárgyalja. A Kruskal-algoritmus útösszenyomáson alapuló, rendkívül hatékony adatstruktúrát használó megvalósításának részletei Rónyai, Ivanyos és Szabó könyvének [297] 6.6.3. fejezetében olvashatóak. A ROCK algoritmust Guha, Rastogi és Shim [149] vezette be.

A DBSCAN algoritmus Ester, Kriegel, Sander és Xu [99], az OPTICS eljárás Ankerst, Breunig, Kriegel és Sanders [20] munkája.

A szerzők munkáját részben támogatta a T 042706 számú OTKA-szerződés és az ADATROSTA NKFP-2/0017/2002 számú projekt.

30. Lekérdezés átírás relációs adatbázisokban

Az első kötetben (lásd 12. fejezet) bevezettük a relációs adatbázisok alapfogalmait, többek között a relációs séma, reláció, reláció példány fogalmát. Az adatbázisokat tervezői oldalról közelítettük meg, a fő kérdés az volt, hogyan érhetjük el, hogy elkerüljük az adatok redundáns tárolását, illetve az adatbázis használata során fellépő különböző anomáliákat.

Jelen fejezetben a sémát adottnak tekintjük és megpróbáljuk a felhasználó kérdéseit minél gyorsabban és teljesebben megválaszolni. Ehhez először (30.1. alfejezet) áttekintjük az alapvető (elméleti) lekérdezési nyelveket, az ezek közti kapcsolatokat.

A fejezet második részében (30.2. alfejezet) a nézeteket tárgyaljuk. Informálisan, egy nézet nem más, mint egy lekérdezés eredménye. Bemutatjuk a nézetek kapcsolatát lekérdezések gyorsításával, a fizikai adatfüggetlenség biztosításával, valamint az adatok integrálásával.

A fejezet harmadik részében (30.3. alfejezet) lekérdezések átírásával foglalkozunk.

30.1. Lekérdezések

Tekintsük a budapesti mozihálózat adatbázisát. Tegyük fel, hogy a séma három relációból áll:

$$\textbf{PestiMűsor} = \{Filmek, Mozik, Műsor\} \; . \tag{30.1}$$

Az egyes relációk sémái a következőek:

$$\begin{aligned} Filmek &= \{\textbf{C}ím, \textbf{R}endező, \textbf{Sz}ínész\} \; , \\ Mozik &= \{\textbf{M}ozi, \textbf{U}tca, \textbf{T}elefon\} \; , \\ Műsor &= \{\textbf{M}ozi, \textbf{C}ím, \textbf{I}dőpont\} \; . \end{aligned} \tag{30.2}$$

Az egyes relációk példányainak lehetséges részletei a 30.1. ábrán láthatók.
Tipikus felhasználói kérdések lehetnek:

30.1 Ki rendezte a “Kontroll”-t?

30.2 Listázzuk az összes olyan mozi nevét és címét, ahol Kuroszava filmet játszanak!

30.3 Adjuk meg azon rendezők nevét, akik szerepeltek valamelyik saját filmjükben!

Ezek a kérdések leképezéseket definiálnak a **PestiMűsor** adatbázis séma relációiból valamilyen másik sémába (jelen esetben egyetlen relációból álló sémákba). Formálisan meg kell különböztetnünk a ***lekérdezést*** és a ***lekérdezés függvényt.*** Az előbbi szintaktikus fogalom,

Filmek

C*ím*	**R***endező*	**Sz***ínész*
Kontroll	Antal Nimród	Csányi Sándor
Kontroll	Antal Nimród	Mucsi Zoltán
Kontroll	Antal Nimród	Pindroch Csaba
⋮	⋮	⋮
A vihar kapujában	Kuroszava Akira	Mifune Tosiro
A vihar kapujában	Kuroszava Akira	Kjó Macsiko
A vihar kapujában	Kuroszava Akira	Maszajuki Mori

Mozik

M*ozi*	**U***tca*	**T***elefon*
Bem	II., Margit krt. 5/b.	316-8708
Corvin Budapest Filmpalota	VIII., Corvin köz 1.	459-5050
Európa	VII., Rákóczi út 82.	322-5419
Művész	VI., Teréz krt. 30.	332-6726
⋮	⋮	⋮
Uránia Nemzeti Filmszínház	VIII., Rákóczi út 21.	486-3413
Vörösmarty	VIII., Üllői út 4.	317-4542

Műsor

M*ozi*	**C***ím*	**I***dőpont*
Bem	A vihar kapujában	19:00
Bem	A vihar kapujában	21:30
Uránia Nemzeti Filmszínház	Kontroll	18:15
Művész	A vihar kapujában	16:30
Művész	Kontroll	17:00
⋮	⋮	⋮
Corvin Budapest Filmpalota	Kontroll	10:15

30.1. ábra. A **PestiMűsor** adatbázis.

az utóbbi pedig leképezés a bemeneti sémához tartozó példányok halmazából a kimeneti séma példányainak halmazába, amit a lekérdezés határoz meg, valamilyen alkalmas szemantikus értelmezés szerint. Azonban az egyszerűség kedvéért mindkét fogalomra a "lekérdezés" szót használjuk, a környezetből mindig világos lesz, hogy éppen melyikről beszélünk.

30.1. definíció. *Az* **R** *bemeneti séma feletti* q_1 *és* q_2 *lekérdezések* ***ekvivalensek,*** *jelölésben* $q_1 \equiv q_2$*, ha ugyanaz a kimeneti sémájuk, és minden* **R***-hez tartozó* $\mathcal{I}$ *példányra* $q_1(\mathcal{I}) = q_2(\mathcal{I})$.

A fejezet további részében áttekintjük a legfontosabb lekérdezési nyelveket. Szükségünk lesz a lekérdezési nyelvek kifejező erejének összehasonlítására.

30.2. definíció. *Legyenek* $\mathcal{Q}_1$ *és* $\mathcal{Q}_2$ *lekérdezési nyelvek (a megfelelő értelmezéssel).* $\mathcal{Q}_2$ ***gazdagabb,*** *mint* $\mathcal{Q}_1$ *(* $\mathcal{Q}_1$ ***szűkebb,*** *mint* $\mathcal{Q}_2$*), jelölésben* $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$*, ha minden* q_1 $\mathcal{Q}_1$*-beli lekérdezéshez van* $q_2 \in \mathcal{Q}_2$*, amelyre* $q_1 \equiv q_2$*.* $\mathcal{Q}_1$ *és* $\mathcal{Q}_2$ ***ekvivalensek,*** *ha* $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ *és* $\mathcal{Q}_1 \sqsupseteq \mathcal{Q}_2$.

30.1. példa. *Lekérdezés.* Tekintsük a 30.2. kérdést. Első közelítésben a következő megoldást találjuk:

> **if** léteznek a *Filmek*, *Mozik* és *Műsor* relációkban $(x_C$, "Kuroszava Akira", $x_{Sz})$, (x_M, x_U, x_T) és (x_M, x_C, x_I) sorok
> **then** vegyük be a $(Mozi : x_M, Utca : x_U)$ sort az eredmény relációba.

$x_C, x_{Sz}, x_M, x_I, x_U, x_T$ különböző változókat jelölnek, amelyek az értékeiket a megfelelő attribútum értelmezési tartományából veszik fel. Ugyanazon változók használatával közvetetten jeleztük, hogy a különböző sorokban hol kell egyenlő értékeknek szerepelniük.

30.1.1. Konjunktív lekérdezések

A lekérdezések legegyszerűbb és legtöbb jó tulajdonsággal rendelkező fajtája. Három, egymással ekvivalens változatát ismertetjük, amelyek közül kettő logikai alapú, a harmadik pedig algebrai. A név a logikai változatból ered, olyan elsőrendű kifejezéseken alapszik, amelyek csak egzisztenciális kvantorokat ($\exists$), valamint "és"-sel (konjunkcióval) összekötött atomi kifejezéseket tartalmaznak.

Datalog – szabály alapú lekérdezés

Az $(x_1, x_2, \ldots, x_m)$ sort ***szabad sornak*** nevezzük, ha az x_i-k változók vagy konstansok. A szabad sor a reláció példány egy sorának általánosítása. A 30.1. példában szereplő $(x_C$, "Kuroszava Akira", $x_{Sz})$ sor szabad.

30.3. definíció. *Legyen* **R** *relációs adatbázis séma.* ***Szabály alapú konjunktív lekérdezésen*** *a következő alakú kifejezést értjük:*

$$val(u) \leftarrow R_1(u_1), R_2(u_2), \ldots, R_n(u_n)\ , \tag{30.3}$$

ahol $n \geq 0$*,* $R_1, R_2, \ldots, R_n$ **R***-beli reláció nevek, val olyan reláció név, ami nincs* **R***-ben;* $u, u_1, u_2, \ldots, u_n$ *szabad sorok. Minden u-ban előforduló változónak elő kell fordulnia* $u_1, u_2, \ldots, u_n$ *valamelyikében is.*

A szabály alapú konjunktív lekérdezéseket egyszerűbben csak ***szabályoknak*** is nevezzük. $val(u)$ a szabály ***feje,*** $R_1(u_1), R_2(u_2), \ldots, R_n(u_n)$ a szabály ***teste.*** $R_i(u_i)$-t ***(relációs) atomnak*** nevezzük. Feltesszük, hogy a fejben előforduló összes változó előfordul valamelyik testbeli atomban is.

Egy szabályt úgy tekinthetünk, mint valamilyen eszközt, ami megmondja, hogyan vezethetünk le újabb és újabb ***tényeket,*** azaz sorokat, a lekérdezés eredmény relációjába. Ha

találunk a szabályban előforduló változóknak olyan értékeket, hogy minden $R_i(u_i)$ atom igaz (azaz a megfelelő sor az R_i relációban van), akkor a *val* relációba bevesszük az u sort. Mivel a fejben előforduló változók előfordulnak testbeli atomokban is, elérjük, hogy sohasem kell ***végtelen*** értelmezési tartományokkal foglalkozzunk, hiszen a változók csak az éppen lekérdezett példányban előforduló konstans értékeket vehetik fel. Formálisan, legyen $\mathcal{I}$ az $\mathbf{R}$ relációs séma feletti példány, q pedig (30.3)-mal adott lekérdezés. Jelölje $var(q)$ a q-ban előforduló változók halmazát, $dom(\mathcal{I})$ pedig az $\mathcal{I}$-beli konstansok halmazát. $\mathcal{I}$ q ***alatti képe***

$$q(\mathcal{I}) = \{\nu(u) | \nu\colon var(q) \rightarrow dom(\mathcal{I}) \text{ és } \nu(u_i) \in R_i \; i = 1, 2, \ldots, n\}. \tag{30.4}$$

$q(\mathcal{I})$ kiszámításának triviális módja, hogy valamely sorrendben végignézzük a lehetséges ν kiértékeléseket. Ennél hatékonyabb algoritmus is lehetséges, egyrészt a lekérdezés ekvivalens átalakításával, másrészt megfelelő indexek használatával.

Fontos különbség a fejben és a testben szereplő atomok között, hogy az $R_1, R_2, \ldots, R_n$ relációkat adottnak, (fizikailag) tároltnak tekintjük, míg a *val* relációt nem, azt úgy gondoljuk, hogy a szabály segítségével számoljuk ki. Ebből adódik az elnevezés: R_i-k ***extenzionális relációk,*** *val* ***intenzionális reláció.***

Az $\mathbf{R}$ séma feletti q lekérdezés ***monoton,*** ha az $\mathcal{I}$ és $\mathcal{J}$ $\mathbf{R}$ feletti példányokra $\mathcal{I} \subseteq \mathcal{J}$-ből $q(\mathcal{I}) \subseteq q(\mathcal{J})$ következik. q ***kielégíthető,*** ha létezik olyan $\mathcal{I}$ példány, amelyre $q(\mathcal{I}) \neq \emptyset$. A következő egyszerű észrevétel bizonyítását az Olvasóra bízzuk (30.1-1. gyakorlat).

30.4. állítás. *Szabály alapú lekérdezések monotonok és kielégíthetőek.*

A 30.4. állítás rámutat a szabály alapú lekérdezések korlátaira. Például a *Mely mozikban játszanak csak Jancsó filmeket?* lekérdezés nyilvánvalóan nem monoton, tehát nem is fejezhető ki (30.3) formájú szabállyal.

Táblázatos lekérdezések

Ha a változók és konstansok közti különbséget nem vesszük figyelembe, akkor egy szabály teste a séma feletti példánynak is tekinthető. Ez a nézőpont elvezet a konjunktív lekérdezések táblázatos formájához, ami leginkább hasonlatos a Microsoft Access adatbázis-kezelő rendszer (QBE: Query By Example) szemléletes lekérdezéseihez.

30.5. definíció. *Az* $\mathbf{R}$ *séma feletti* ***tábla*** *az* $\mathbf{R}$ *séma feletti példány általánosítása, a sorokban konstansok mellett változók is előfordulhatnak. A* $(\mathbf{T}, u)$ *pár* ***táblázatos lekérdezés,*** *ha* $\mathbf{T}$ *egy tábla és* u *egy szabad sor, úgy, hogy minden* u*-ban előforduló változó előfordul* $\mathbf{T}$*-ben is. Az* u *szabad sor a táblázatos lekérdezés* ***összegzése.***

A $(\mathbf{T}, u)$ táblázatos lekérdezés u összegzés sora mutatja meg, hogy mely sorok alkotják a lekérdezés eredményét. Az eljárás lényege, hogy megpróbáljuk a $\mathbf{T}$ táblával adott mintát az adatbázisban megtalálni, és ha sikerül, akkor az u-nak megfelelő sort bevesszük az eredmény relációba. Pontosabban, $\nu\colon var(\mathbf{T}) \rightarrow dom(\mathcal{I})$ a $(\mathbf{T}, u)$ tábla ***beágyazása*** az $\mathcal{I}$ példányba, ha $\nu(\mathbf{T}) \subseteq \mathcal{I}$. A $(\mathbf{T}, u)$ táblázatos lekérdezés eredmény relációja mindazon $\nu(u)$ sorokból áll, amelyekre ν a $(\mathbf{T}, u)$ tábla beágyazása az $\mathcal{I}$ példányba.

30.2. példa. *Táblázatos lekérdezés.* Legyen **T** a következő tábla.

Filmek	*Cím*	*Rendező*	*Színész*
	x_C	"Kuroszava Akira"	x_{Sz}

Mozik	*Mozi*	*Utca*	*Telefon*
	x_M	x_U	x_T

Műsor	*Mozi*	*Cím*	*Időpont*
	x_M	x_C	x_I

A $(\mathbf{T}, \langle Mozi\colon x_M, Utca\colon x_U\rangle)$ táblás lekérdezés a bevezetés 30.2. kérdését válaszolja meg.

A táblázatos lekérdezések szintaxisa nagyon hasonló a szabály alapú lekérdezésekéhez. A későbbiekben hasznos lesz számunkra, hogy a táblázatos lekérdezések nyelvén könnyen megfogalmazható lesz annak feltétele, hogy két lekérdezés közül az egyik mikor tartalmazza a másikat.

Relációs algebra*

Az adatbázis relációkból áll, a relációk pedig sorok halmazai. A lekérdezések eredménye is adott attribútum halmazzal rendelkező reláció. Természetes gondolat, hogy a lekérdezés eredményét a bemeneti relációkból halmazalgebrai, illetve a relációkon értelmezett egyéb műveletekkel fejezzük ki. A ***relációs algebra**** a következő műveleteket tartalmazza[1].

Kiválasztás: Az operáció alakja $\sigma_{A=c}$ vagy $\sigma_{A=B}$, ahol A és B attribútumok, c pedig konstans. A művelet alkalmazható minden olyan R relációra, amelyiknek A (és B) attribútuma, eredménye pedig értelemszerűen a *val* reláció, amelynek ugyanazok az attribútumai, mint R-nek, és azon R-beli sorokat tartalmazza, amelyekre igaz a kiválasztási feltétel.

Vetítés: Az operáció alakja $\pi_{A_1,A_2,\ldots,A_n}$, $n \geq 0$, ahol A_i-k különböző attribútumok. A művelet alkalmazható minden olyan R relációra, amelyik attribútumai közt minden A_i előfordul, eredménye pedig a *val* reláció, amelyik attribútum halmaza $\{A_1, A_2, \ldots, A_n\}$,

$$val = \{t[A_1, A_2, \ldots, A_n] | t \in R\} ,$$

azaz az R-beli sorok $\{A_1, A_2, \ldots, A_n\}$ attribútum halmazra való megszorításaiból áll.

Természetes összekapcsolás: Ezt a műveletet már az első kötet 12. fejezetében definiáltuk. Jelölése $\bowtie$, bemenete kettő R_1, R_2 (vagy több) reláció, V_1, V_2 attribútum halmazokkal. Az eredmény reláció attribútum halmaza $V_1 \cup V_2$.

$$R_1 \bowtie R_2 = \{t \text{ sor } V_1 \cup V_2 \text{ felett } | \exists v \in R_1, \exists w \in R_2, t[V_1] = v \text{ és } t[V_2] = w\} .$$

Átnevezés: Attribútum átnevezés nem más, mint egy egy-egy értelmű leképezés a véges U attribútum halmazról az összes attribútumok halmazába. Az f attribútum átnevezés megadható az $(A, f(A))$ párok listájával, ahol $A \neq f(A)$, ezt általában $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_n$ alakban írjuk, $f(A_i) = B_i$. Az ***átnevezés operátor*** δ_f, U feletti bemenetekről $f[U]$ feletti kimenetekre képez. Ha R az U feletti reláció, akkor

$$\delta_f(R) = \{v \ f[U] \text{ felett} | \exists u \in R, v(f(A)) = u(A), \ \forall A \in U\} .$$

[1] A relációs algebra* a később bevezetett (teljes) relációs algebra ***monoton*** része.

Relációs algebra* lekérdezéseket a fenti műveletek véges sokszor ismételt alkalmazásával nyerünk a ***relációs algebrai alap lekérdezésekből,*** amelyek

Bemenet reláció: R.

Egyetlen konstans: $\{\langle A : a\rangle\}$, ahol a konstans, A attribútum név.

30.3. példa. *Relációs algebra* lekérdezés.* A bevezetés 30.2. kérdése relációs algebrai műveletekkel a következőképpen fejezhető ki

$$\pi_{Mozi,Utca}\left((\sigma_{Rendező=\text{“Kuroszava Akira”}}(Filmek) \bowtie Műsor) \bowtie Mozik\right).$$

A relációs algebra* lekérdezés által megvalósított leképezést a műveleti fa szerinti indukcióval definiálhatjuk értelemszerűen. Könnyen látható (30.1-2. gyakorlat), hogy relációs algebra* használatával megadható olyan lekérdezés, ami nem elégíthető ki. Az ilyennel ekvivalens szabály alapú, illetve táblázatos lekérdezés nyilván nem létezik. Azonban igaz a következő.

30.6. tétel. *A szabály alapú lekérdezések, a táblázatos lekérdezések és a kielégíthető relációs algebra* lekérdezések nyelvei ekvivalensek.*

A tétel bizonyítása három fő lépésből áll:

1. Szabály alapú ≡ Táblázatos
2. Kielégíthető relációs algebra* ⊑ Szabály alapú
3. Szabály alapú ⊑ Kielégíthető relációs algebra*

Az első lépést az Olvasóra bízzuk (30.1-3. gyakorlat). A második lépéshez először be kell látni, hogy a szabály alapú lekérdezések nyelve zárt a lekérdezések egymásba ágyazására nézve. Pontosabban, legyen $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$ adatbázis, q lekérdezés $\mathbf{R}$ felett. Ha q eredmény relációja S_1, akkor egy következő lekérdezésben már S_1 is használható ugyanúgy, mint $\mathbf{R}$ tetszőleges extenzionális relációja. Így definiálhatjuk az S_2, majd annak segítségével az S_3, és így tovább, relációkat. Az S_i relációk ***intenzionális*** relációk. A P ***konjunktív lekérdezés program*** szabályok sorozata

$$\begin{array}{rcl} S_1(u_1) & \leftarrow & test_1 \\ S_2(u_2) & \leftarrow & test_2 \\ & \vdots & \\ S_m(u_m) & \leftarrow & test_m, \end{array} \tag{30.5}$$

ahol az S_i-k különbözőek, és nincsenek $\mathbf{R}$-ben. A $test_i$ szabály testben csak az $R_1, R_2, \ldots R_n$ és $S_1, S_2, \ldots, S_{i-1}$ relációk fordulhatnak elő. P eredmény relációjának S_m-t tekintjük, kiszámítása a szabályok sorrend szerinti kiértékelésével történik. Nem nehéz látni, hogy a változók megfelelő átnevezésével P egyetlen szabállyal helyettesíthető, ahogy azt a következő példa mutatja.

30.4. példa. *Konjunktív lekérdezés program.* Legyen $\mathbf{R} = \{Q, R\}$, és tekintsük a következő konjunktív lekérdezés programot

$$\begin{aligned} S_1(x,z) &\leftarrow Q(x,y), R(y,z,w) \\ S_2(x,y,z) &\leftarrow S_1(x,w), R(w,y,v), S_1(v,z) \\ S_3(x,z) &\leftarrow S_2(x,u,v), Q(v,z). \end{aligned} \tag{30.6}$$

Az (30.6) első két szabálya alapján S_2 felírható csak Q és R használatával

$$S_2(x,y,z) \leftarrow Q(x,y_1), R(y_1,w,w_1), R(w,y,v), Q(v,y_2), R(y_2,z,w_2). \tag{30.7}$$

Látható, hogy bizonyos változókat át kellett nevezni, hogy elkerüljük a különböző szabály testek nem kívánt egymásra hatását. A (30.7) kifejezést (30.6) harmadik szabályába írva S_2 helyére, és megfelelően átnevezve a változókat kapjuk

$$S_3(x,z) \leftarrow Q(x,y_1), R(y_1,w,w_1), R(w,u,v_1), Q(v_1,y_2), R(y_2,v,w_2), Q(v,z). \tag{30.8}$$

Ezek után elegendő a relációs algebra* műveleteit egyenként megvalósítani szabállyal.

$P \bowtie Q$: Jelölje $\overrightarrow{x}$ a P és Q közös attribútumainak megfelelő változók (konstansok) listáját, $\overrightarrow{y}$ a csak P-ben, míg $\overrightarrow{z}$ a csak Q-ban előforduló attribútumoknak megfelelő változókat (konstansokat). Ekkor a $val(\overrightarrow{x},\overrightarrow{y},\overrightarrow{z}) \leftarrow P(\overrightarrow{x},\overrightarrow{y}), Q(\overrightarrow{x},\overrightarrow{z})$ szabály a $P \bowtie Q$ relációt adja eredményül.

$\sigma_F(R)$: Tegyük fel, hogy $R = R(A_1, A_2, \ldots, A_n)$. F a szelekciós feltétel, $A_i = a$ vagy $A_i = A_j$ alakú, ahol A_i, A_j attribútumok, a konstans. Ekkor

$$val(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n) \leftarrow R(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n)\,,$$

illetve

$$val(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_{j-1}, y, x_{j+1}, \ldots, x_n) \leftarrow R(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_{j-1}, y, x_{j+1}, \ldots, x_n)$$

megfelelő szabály. Itt használjuk ki a relációs algebra* lekérdezés kielégíthetőségét. Ugyanis, a műveletek egymás utáni elvégzése során sohasem kapunk olyan kifejezést, amiben két különböző konstanst kéne egyenlővé tennünk.

$\pi_{A_{i_1},A_{i_2},\ldots,A_{i_m}}(R)$: Ha $R = R(A_1, A_2, \ldots, A_n)$, akkor

$$val(x_{i_1}, x_{i_2}, \ldots, x_{i_m}) \leftarrow R(x_1, x_2, \ldots, x_n)$$

jó lesz.

$A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_n$: Az átnevezés relációs algebrai műveletet a megfelelő változók átnevezésével oldhatjuk meg, amint azt a 30.4. példában láttuk.

A harmadik lépés bizonyításához tekintsünk egy

$$val(\overrightarrow{x}) \leftarrow R_1(\overrightarrow{x_1}), R_2(\overrightarrow{x_2}), \ldots, R_n(\overrightarrow{x_n}) \tag{30.9}$$

szabályt. Az R_i relációk attribútumainak megfelelő átnevezésével elérhető, hogy csupa különböző attribútumnév szerepeljen. Ezek után képezhetjük az $R = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$ relációt, ami ténylegesen az R_i relációk direkt szorzata lesz, az attribútum nevek különbözősége miatt. A (30.9) szabályban szereplő konstansokat és változó egyezéseket megfelelő szelekciós operátorok alkalmazásával szimulálhatjuk. Végül a $val(\overrightarrow{x})$ reláció változóinak megfelelő attribútum halmazra való vetítéssel kapjuk a végeredményt.

30.1.2. Kiterjesztések

A konjunktív lekérdezések a lekérdezési nyelvek jól kezelhető fajtája. Azonban viszonylag szűk az általuk kifejezhető kérdések köre. Tekintsük a következőket.

30.4. Adjuk meg az olyan párokat, amelyeknek az első tagja rendezte a második tagot valamilyen filmben, valamint fordítva, a második tag is rendezte az elsőt valamilyen filmben.

30.5. Melyik moziban megy a "Szegénylegények" vagy a "Vihar kapujában"?

30.6 Melyek azok a Hitchcock filmek, amelyekben nem szerepelt Hitchcock maga?

30.7. Listázzuk azokat a filmeket, amelyek összes színésze szerepelt Jancsó Miklós valamelyik filmjében.

30.8. Ismeretes a „Színészlánc" játék. Az első játékos mond egy színészt, a sorban következő pedig egy olyat, aki vele egy filmben játszott. Ez így folytatódik, mindig olyan színészt kell mondani, aki az előzővel játszott egy filmben, és még nem szerepelt a játék folyamán. (Az nyer, aki utolsóként tudja még folytatni a láncot.) Listázzuk azokat a színészeket, akikhez „Latinovits Zoltán"-tól el lehet jutni a játék folyamán.

Egyenlőség atomok

A **30.4** kérdésre könnyen megadhatjuk a választ, ha a szabály testében a relációs atomokon kívül egyenlőséget is megengedünk

$$val(y_1, y_2) \leftarrow Filmek(x_1, y_1, z_1), Filmek(x_2, y_2, z_2), y_1 = z_2, y_2 = z_1. \tag{30.10}$$

Az egyenlőség megengedése két problémát is felvet. Az első, hogy a lekérdezés eredménye végtelen is lehet. Például a

$$val(x, y) \leftarrow R(x), y = z \tag{30.11}$$

lekérdezés eredménye végtelen sok sor, hiszen az y és z változókat az R reláció nem korlátozza, így végtelen sok kiértékelés lehetséges, ami a szabály testet kielégíti. Ezért a q szabály alapú lekérdezés ***tartomány-korlátozott,*** ha minden változó, amelyik q testében előfordul, az előfordul valamely relációs atomban is.

A második probléma az, hogy egyenlőség atomok a szabály testben a testbeli feltétel kielégíthetetlenségét okozhatják, ellentétben a 30.4. állítással. Például a

$$val(x) \leftarrow R(x), x = a, x = b \tag{30.12}$$

szabály tartomány-korlátozott, viszont ha a és b különböző konstansok, akkor a válasz az üres reláció lesz. Könnyen eldönthető, hogy egy q egyenlőségeket is tartalmazó szabály alapú lekérdezés kielégíthető-e.

KIELÉGÍTHETŐ(q)

```
1  Számítsuk ki a q testében levő egyenlőségek tranzitív lezártját.
2  if Két különböző konstans egyenlő a tranzitivitás miatt
3     then return "Nem elégíthető ki."
4     else return "Kielégíthető."
```

Igaz az is (30.1-4. gyakorlat), hogy ha a q egyenlőségeket is tartalmazó szabály alapú lekérdezés kielégíthető, akkor van vele ekvivalens q', amelyik egyenlőség nélküli.

Diszjunkció – egyesítés
A **30.5** kérdés nem fejezhető ki konjunktív lekérdezéssel, azonban ha az egyesítés műveletét hozzávesszük a relációs algebrához, akkor az így kiterjesztett algebrával már kifejezhető:

$$\pi_{Mozi}\left(\sigma_{Cím=\text{“Szegénylegények”}}(Műsor) \cup \sigma_{Cím=\text{“A vihar kapujában”}}(Műsor)\right) . \tag{30.13}$$

Szabály alapú lekérdezések is képesek a **30.5** kérdés kifejezésére, ha megengedjük, hogy több különböző szabálynak is ugyanaz a reláció legyen a fejében:

$$\begin{aligned} val(x_M) &\leftarrow Műsor(x_M, \text{“Szegénylegények”}, x_V) , \\ val(x_M) &\leftarrow Műsor(x_M, \text{“A vihar kapujában”}, x_V) . \end{aligned} \tag{30.14}$$

Ennek általánosítása a ***nemrekurzív datalog program.***

30.7. definíció. *Az* **R** *séma feletti* ***nemrekurzív datalog program***

$$\begin{aligned} S_1(u_1) &\leftarrow test_1 \\ S_2(u_2) &\leftarrow test_2 \\ &\vdots \\ S_m(u_m) &\leftarrow test_m \end{aligned} \tag{30.15}$$

szabályok halmaza, ahol **R***-beli reláció nem szerepel szabály fejben, ugyanaz a reláció több szabály fejében is szerepelhet, valamint létezik a szabályok olyan* $r_1, r_2, \ldots, r_m$ *sorrendje, hogy az* r_i *szabály fejében levő reláció nem fordul elő semelyik* r_j *szabály testében sem, ha* $j \leq i$.

A (30.15) nemrekurzív datalog program eredményének kiszámítása a (30.5) konjunktív lekérdezés program kiszámításához hasonló. A szabályokat a 30.7. definícióban szereplő sorrendben számoljuk, ha több szabály fejében ugyanaz a reláció szerepel, akkor a szabályok által adott sorok halmazainak egyesítését vesszük.

A $(\mathbf{T}_i, u)$ $i = 1, 2, \ldots, n$ táblázatos lekérdezések egyesítését $(\{\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_n\}, u)$ jelöli. Kiszámításához az egyes $(\mathbf{T}_i, u)$ lekérdezéseket külön-külön kiszámítjuk, majd az eredmények egyesítését vesszük. Igaz a következő tétel.

30.8. tétel. *Az egyetlen célrelációval rendelkező nemrekurzív datalog programok nyelve és a relációs algebra kiegészítve az unió műveletével ekvivalensek.*

A 30.8. tétel bizonyítása hasonló a 30.6. tétel bizonyításához, az Olvasóra bízzuk (30.1-5. gyakorlat). Megjegyezzük, hogy a táblázatos lekérdezések egyesítésének kifejező ereje gyengébb, aminek az az oka, hogy ugyanazt az összegző sort követeljük meg minden egyes táblázathoz. A

$$\begin{aligned} val(a) &\leftarrow \\ val(b) &\leftarrow \end{aligned} \tag{30.16}$$

nemrekurzív datalog program nem valósítható meg táblázatos lekérdezések egyesítéseként.

Tagadás
A **30.6**. kérdés nyilvánvalóan nem monoton. Tegyük fel, hogy a *Filmek* relációban léteznek sorok Hitchcock *Psycho* című filmjéről, például (“Psycho”,”Hitchcock”,”A. Perkins”), (“Psycho”,”Hitchcock”,”J. Leigh”), ... , azonban

nincs (“Psycho”,”Hitchcock”,”Hitchcock”) sor. Ekkor a **30.6**. lekérdezés eredményében a (“Psycho”) sor is szerepel. Azonban kicsit alaposabb kutatással kideríthető, hogy Hitchcock szerepel a *Psycho* című filmben, mint “egy ember cowboy kalapban”. Ha ezek után hozzávesszük a *Filmek* relációhoz a (“Psycho”,”Hitchcock”,”Hitchcock”) sort, akkor a **PestiMűsor** séma aktuális példánya bővebb lesz, viszont a **30.6** lekérdezés eredménye szűkebb.

Könnyű látni, hogy az eddigi fejezetekben tárgyalt lekérdezési nyelvek által megvalósított lekérdezések monotonok, azaz a **30.6**. lekérdezés nem valósítható meg nemrekurzív datalog programmal, vagy vele ekvivalens nyelvvel. Azonban, ha a relációs algebrai műveletek közé felvesszük a kivonás ($-$) operátort is, akkor alkalmas a **30.6**. típusú lekérdezések megvalósítására. Például,

$$\pi_{Cím}\,(\sigma_{Rendező=\text{“Hitchcock”}}(Filmek)) - \pi_{Cím}\,(\sigma_{Színész=\text{“Hitchcock”}}(Filmek)) \tag{30.17}$$

pontosan a **30.6.** lekérdezést valósítja meg. A (teljes) relációs algebrát tehát a $\{\sigma, \pi, \bowtie, \delta, \cup, -\}$ műveletek alkotják. A relációs algebra fontosságát az is mutatja, hogy Codd a $\mathcal{Q}$ lekérdezési nyelvet pontosan akkor nevezi ***relációsan teljesnek,*** ha minden q algebrai lekérdezéshez van $q' \in \mathcal{Q}$, hogy $q \equiv q'$.

Ha megengedünk szabály testekben ***negatív literálokat,*** azaz $\neg R(u)$ alakú atomokat, akkor az így kapott ***nemrekurzív datalog tagadással,*** jelölésben ***nr-datalog***$^\neg$ már relációsan teljes lesz.

30.9. definíció. *Nemrekurzív datalog*$^\neg$ *(nr-datalog*$^\neg$*) szabály a*

$$q: \quad S(u) \leftarrow L_1, L_2, \ldots, L_n \tag{30.18}$$

alakú szabály, ahol S egy reláció, u egy szabad sor, az L_i pedig ***literál,*** *azaz $R(v)$ vagy $\neg R(v)$ alakú kifejezés, $i = 1, 2, \ldots, n$, v szabad sor. S nem fordul elő a szabály testében. A szabály* ***tartomány-korlátozott,*** *ha minden x változó, ami előfordul a szabályban, előfordul valamely* ***pozitív literálban*** *($R(v)$ alakú kifejezésben) is a szabály testében. Ha másképp nem jelezzük, akkor minden nr-datalog*$^\neg$ *szabályt tartomány-korlátozottnak tekintünk.*

A (30.18) szabály jelentése a következő. Legyen **R** relációs séma, amelyik tartalmazza a q testében szereplő összes relációt, továbbá legyen $\mathcal{I}$ **R** feletti példány. $\mathcal{I}$ ***q-szerinti képe***

$$\begin{aligned} q(\mathcal{I}) = \{\nu(u) \mid\ & \nu \text{ a változók kiértékelése és } i = 1, 2, \ldots, n\text{-re} \\ & \nu(u_i) \in \mathcal{I}(R_i), \text{ ha } L_i = R_i(u_i) \text{ és} \\ & \nu(u_i) \notin \mathcal{I}(R_i), \text{ ha } L_i = \neg R_i(u_i)\}. \end{aligned} \tag{30.19}$$

Az **R** séma feletti ***nr-datalog***$^\neg$ ***program*** nr-datalog$^\neg$ szabályok

$$\begin{aligned} S_1(u_1) &\leftarrow test_1 \\ S_2(u_2) &\leftarrow test_2 \\ &\vdots \\ S_m(u_m) &\leftarrow test_m \end{aligned} \tag{30.20}$$

halmaza, ahol **R**-beli reláció nem szerepel szabály fejben, ugyanaz a reláció több szabály fejében is szerepelhet, valamint létezik a szabályok olyan $r_1, r_2, \ldots, r_m$ sorrendje, hogy az r_i szabály fejében levő reláció nem fordul elő semelyik r_j szabály testében sem, ha $j \leq i$.

A (30.20) nr-datalog¬ program eredményének kiszámítása az **R** séma $\mathcal{I}$ példányára alkalmazva megegyezik a (30.15) nemrekurzív datalog program esetén használt módszerrel, azzal a különbséggel, hogy az egyes nr-datalog¬ szabályokat (30.19) szerint értelmezzük.

30.5. példa. *Nr-datalog¬ program.* Tegyük fel, hogy minden filmnek, amelyik a *Filmekben* szerepel, egyetlen rendezője van. (Nem mindig teljesül a valóságban!) A **30.6** lekérdezést

$$val(x) \leftarrow Filmek(x, \text{“Hitchcock”}, z), \neg Filmek(x, \text{“Hitchcock”}, \text{“Hitchcock”}) \tag{30.21}$$

nr-datalog¬ szabály valósítja meg. A **30.7.** lekérdezést pedig a

$$\begin{aligned} Jancsó\text{-}színész(z) &\leftarrow Filmek(x, \text{”Jancsó”}, z) \\ Nem\text{-}ez\text{-}a\text{-}válasz(x) &\leftarrow Filmek(x, y, z), \neg Jancsó\text{-}színész(z) \\ Válasz(x) &\leftarrow Filmek(x, y, z), \neg Nem\text{-}ez\text{-}a\text{-}válasz(x) \end{aligned} \tag{30.22}$$

nr-datalog¬ program válaszolja meg. Óvatosnak kell lennünk azonban nr-datalog¬ program írásakor. Ha a (30.22) program első két szabályát egybe vonnánk a 30.4. példához hasonlóan

$$\begin{aligned} Rossz\text{-}nem\text{-}v(x) &\leftarrow Filmek(x, y, z), \neg Filmek(x', \text{”Jancsó”}, z), Filmek(x', \text{”Jancsó”}, z') \\ Válasz(x) &\leftarrow Filmek(x, y, z), \neg Rossz\text{-}nem\text{-}v(x), \end{aligned} \tag{30.23}$$

akkor (30.23) nem a **30.7.** lekérdezést adná, hanem (feltéve, hogy minden filmnek egy rendezője van) a következő lekérdezést.

30.9. Listázzuk azokat a filmeket, amelyek minden színésze az ***összes*** Jancsó filmben szerepelt.

Könnyen látható, hogy minden kielégíthető nr-datalog¬ program, amelyik tartalmaz egyenlőség atomokat is, helyettesíthető olyannal, amelyikben nem szerepelnek egyenlőség atomok. Továbbá igaz az alábbi állítás is.

30.10. állítás. *A kielégíthető (teljes) relációs algebra és az egyetlen cél relációval rendelkező kielégíthető nr-datalog¬ program ekvivalens lekérdezési nyelvek.*

Rekurzió

A **30.8**. kérdést az eddigi lekérdezési nyelvekkel nem tudjuk megfogalmazni. Szükségünk lenne valamilyen ***a priori*** információra arról, hogy legfeljebb milyen hosszú *színészlánc* képezhető a kiindulási színészből. Tegyük fel, hogy tudjuk, “Latinovits”-ból indulva legfeljebb 117 hosszú lánc képezhető. (Érdekes lenne tudni a tényleges értéket!) Ekkor a következő nemrekurzív datalog program megadja a választ.

$$\begin{aligned} Film\text{-}társ(z_1, z_2) &\leftarrow Filmek(x, y, z_1), Filmek(x, y, z_2), z_1 < z_2{}^2 \\ Rész\text{-}válasz_1(z) &\leftarrow Film\text{-}társ(z, \text{“Latinovits”}) \\ Rész\text{-}válasz_1(z) &\leftarrow Film\text{-}társ(\text{“Latinovits”}, z) \\ Rész\text{-}válasz_2(z) &\leftarrow Film\text{-}társ(z, y), Rész\text{-}válasz_1(y) \\ Rész\text{-}válasz_2(z) &\leftarrow Film\text{-}társ(y, z), Rész\text{-}válasz_1(y) \\ &\vdots \quad\quad \vdots \\ Rész\text{-}válasz_{117}(z) &\leftarrow Film\text{-}társ(z, y), Rész\text{-}válasz_{116}(y) \\ Rész\text{-}válasz_{117}(z) &\leftarrow Film\text{-}társ(y, z), Rész\text{-}válasz_{116}(y) \\ Latinovits\text{-}lánc(z) &\leftarrow Rész\text{-}válasz_1(z) \\ Latinovits\text{-}lánc(z) &\leftarrow Rész\text{-}válasz_2(z) \\ &\vdots \quad\quad \vdots \\ Latinovits\text{-}lánc(z) &\leftarrow Rész\text{-}válasz_{117}(z) \end{aligned} \tag{30.24}$$

Sokkal egyszerűbben fejezhető ki a **30.8**. kérdés ***rekurzió*** segítségével. Ténylegesen egy gráf, a *Film-társ*, ***tranzitív lezártját*** akarjuk kiszámolni. Az egyszerűség kedvéért kicsit megváltoztatjuk a *Film-társ* definícióját, (körülbelül megkétszerezve a tárigényt).

$$\begin{array}{rcl} \mathit{Film\text{-}társ}(z_1, z_2) & \leftarrow & \mathit{Filmek}(x, y, z_1), \mathit{Filmek}(x, y, z_2) \\ \mathit{Színészlánc\text{-}társ}(x, y) & \leftarrow & \mathit{Film\text{-}társ}(x, y) \\ \mathit{Színészlánc\text{-}társ}(x, y) & \leftarrow & \mathit{Film\text{-}társ}(x, z), \mathit{Színészlánc\text{-}társ}(z, y)\ . \end{array} \tag{30.25}$$

A (30.25) datalog program rekurzív, mivel a *Színészlánc-társ* reláció definíciójában önmagát használjuk. Tegyük fel, hogy ez értelmezhető (lásd később), ekkor a **30.8** kérdésre a választ megadja a

$$\mathit{Latinovits\text{-}lánc}(x) \leftarrow \mathit{Színészlánc\text{-}társ}(x, \text{“Latinovits”}) \tag{30.26}$$

szabály.

30.11. definíció. *Az*

$$R_1(u_1) \leftarrow R_2(u_2), R_3(u_3), \ldots, R_n(u_n) \tag{30.27}$$

kifejezés ***datalog szabály,*** *ha $n \geq 1$, R_i-k reláció nevek, u_i-k megfelelő hosszúságú szabad sorok. Minden u_1-beli változó elő kell forduljon $u_2, \ldots u_n$ valamelyikében is. A szabály feje $R_1(u_1)$, teste $R_2(u_2), R_3(u_3), \ldots, R_n(u_n)$. A* ***datalog program*** *(30.27) alakú szabályok véges halmaza. Legyen P datalog program. A P-ben szereplő R reláció* ***extenzionális,*** *ha csak szabály testekben fordul elő.* ***Intenzionális*** *a reláció, ha valamelyik szabály fejében előfordul.*

Ha ν a (30.27) szabályban szereplő változók valamely kiértékelése, akkor $R_1(\nu(u_1)) \leftarrow R_2(\nu(u_2)), R_3(\nu(u_3)), \ldots, R_n(\nu(u_n))$ a (30.27) szabály ***megvalósítása.*** Az ***extenzionális (adatbázis) séma*** P extenzionális relációiból áll, jelölésben $edb(P)$. $idb(P)$, az ***intenzionális (adatbázis) séma*** hasonlóképpen P intenzionális relációit tartalmazza. Jelölje $sch(P) = edb(P) \cup idb(P)$. A P datalog program szemantikus jelentése egy leképezés az $edb(P)$ feletti példányok halmazáról az $idb(P)$ feletti példányok halmazába. Ezt a jelentést modellelméletileg, bizonyítás-elméletileg, illetve valamely leképezés fixpontjának segítségével lehet megadni. Mivel az első kettő lényegileg ekvivalens a harmadikkal, és tárgyalásuk túl messzire vezetne, ezért csak a fixpont alapú definícióval foglalkozunk.

A 30.11. definícióban nem használtunk negatív literálokat. Ennek fő oka, hogy általában a rekurzió és a tagadás együtt értelmetlen lehet. Azonban bizonyos esetekben szükségünk lehet negatív atomokra is, akkor majd speciálisan foglalkozunk a program értelmezésével.

Fixpont szemantika

Legyen P datalog program, $\mathcal{K}$ $sch(P)$ feletti példány. Az A ***tény,*** azaz konstansokból álló sor, $\mathcal{K}$ és P ***közvetlen következménye,*** ha vagy $A \in \mathcal{K}(R)$ valamely $R \in sch(P)$ relációra, vagy $A \leftarrow A_1, A_2, \ldots, A_n$ a P valamelyik szabályának megvalósítása és minden A_i $\mathcal{K}$-ban van. P ***közvetlen következmény operátora*** T_P az $sch(P)$ feletti példányok halmazából képez önmagára. $T_P(\mathcal{K})$ a $\mathcal{K}$ és P összes közvetlen következményéből áll.

30.12. állítás. *A T_P közvetlen következmény operátor monoton.*

[2] Az egyenlőség atomokhoz hasonlóan használhatunk más összehasonlítási atomokat is. Itt a $z_1 < z_2$ azt biztosítja, hogy minden pár csak egyszer szerepel a felsorolásban.

Bizonyítás. Tegyük fel, hogy $\mathcal{I}$ és $\mathcal{J}$ $sch(P)$ feletti példányok, valamint $\mathcal{I} \subseteq \mathcal{J}$. Legyen A $T_P(\mathcal{I})$-be tartozó tény. Ha $A \in \mathcal{I}(R)$ valamely $R \in sch(P)$ relációra, akkor $\mathcal{I} \subseteq \mathcal{J}$ alapján $A \in \mathcal{J}(R)$ is teljesül. Ha pedig $A \leftarrow A_1, A_2, \ldots, A_n$ a P valamelyik szabályának megvalósítása és minden A_i $\mathcal{I}$-ben van, akkor $A_i \in \mathcal{J}$ teljesül. ■

T_P definíciójából következik, hogy $\mathcal{K} \subseteq T_P(\mathcal{K})$. Innen, felhasználva a 30.12. állítást kapjuk, hogy

$$\mathcal{K} \subseteq T_P(\mathcal{K}) \subseteq T_P(T_P(\mathcal{K})) \subseteq \ldots \ . \tag{30.28}$$

30.13. tétel. *Minden $sch(P)$ feletti $\mathcal{I}$ példányhoz létezik egy egyértelmű minimális $\mathcal{I} \subseteq \mathcal{K}$ példány, ami a T_P* ***fixpontja,*** *azaz $\mathcal{K} = T_P(\mathcal{K})$.*

Bizonyítás. Jelölje $T_P^i(\mathcal{I})$ a T_P operátor i-szeres egymás utáni alkalmazását, és legyen

$$\mathcal{K} = \bigcup_{i=0}^{\infty} T_P^i(\mathcal{I}) \ . \tag{30.29}$$

(30.29) és T_P monotonitása alapján

$$T_P(\mathcal{K}) = \bigcup_{i=1}^{\infty} T_P^i(\mathcal{I}) \subseteq \bigcup_{i=0}^{\infty} T_P^i(\mathcal{I}) = \mathcal{K} \subseteq T_P(\mathcal{K}) \ , \tag{30.30}$$

azaz $\mathcal{K}$ fixpont. Könnyen látható, hogy minden olyan fixpont, ami tartalmazza $\mathcal{I}$-t, tartalmazza $T_P^i(\mathcal{I})$-t is minden i-re ($i = 1, 2, \ldots$), azaz $\mathcal{K}$-t is. ■

30.14. definíció. *A P datalog program* ***eredménye*** *az $edb(P)$ feletti $\mathcal{I}$ példányon T_P $\mathcal{I}$-t tartalmazó minimális fixpontja, jelölésben $P(\mathcal{I})$.*

Belátható, lásd 30.1-6. gyakorlat, hogy a (30.28)-beli lánc véges, azaz létezik olyan n, hogy $T_P(T_P^n(\mathcal{I})) = T_P^n(\mathcal{I})$. Ez az alapja a datalog program eredménye naiv kiszámítási módjának.

NAIV-DATALOG(P,$\mathcal{I}$)

```
1  K ← I
2  while T_P(K) ≠ K
3        do K ← T_P(K)
4  return K
```

Természetesen a NAIV-DATALOG eljárás nem optimális, hiszen minden egyes tényt, ami $\mathcal{K}$-ba belekerül, a **while** ciklus minden további végrehajtásánál újra kiszámol.

A FÉLIG-NAIV-DATALOG eljárás elve, hogy amennyire csak lehet, az éppen kiszámított új tényeket használja csak a **while** ciklus során, elkerülve ezzel a már ismert tények újraszámolását. Tekintsük a P datalog programot, $edb(P) = \mathbf{R}$, $idb(P) = \mathbf{T}$. A P-beli

$$S(u) \leftarrow R_1(v_1), \ldots, R_n(v_n), T_1(w_1), \ldots, T_m(w_m) \tag{30.31}$$

szabályhoz, ahol $R_k \in \mathbf{R}$ és $T_j \in \mathbf{T}$, elkészítjük a következő szabályokat $j = 1, 2, \ldots, m$ és $i \geq 1$-re

$$\begin{aligned} temp_S^{i+1}(u) \leftarrow\ & R_1(v_1), \ldots, R_n(v_n), \\ & T_1^i(w_1), \ldots, T_{j-1}^i(w_{j-1}), \Delta_{T_j}^i(w_j), T_{j+1}^{i-1}(w_{j+1}), \ldots, T_m^{i-1}(w_m)\ . \end{aligned} \tag{30.32}$$

A $\Delta_{T_j}^i$ reláció a T_j i-edik iterációban történő változását jelöli. Az i-edik szint S-re vonatkozó szabályainak együttesét P_S^i jelöli (azaz (30.32) szabályokat $temp_S^{i+1}$-re, $j = 1, 2, \ldots, m$ esetén). Tegyük fel, hogy $T_1, T_2, \ldots, T_\ell$ az S *idb* relációt meghatározó szabályok testében szereplő *idb* relációk listája. Jelölje

$$P_S^i(\mathcal{I}, T_1^{i-1}, \ldots, T_\ell^{i-1}, T_1^i, \ldots, T_\ell^i, \Delta_{T_1}^i, \ldots, \Delta_{T_\ell}^i) \tag{30.33}$$

a (30.32) szabályok az $\mathcal{I}$ bemeneti példányra és a $T_j^{i-1}, T_j^i, \Delta_{T_j}^i$ *idb* relációkra való alkalmazásával kapott tényeket (sorokat). Az $\mathcal{I}$ bemeneti példány az $edb(P)$ relációinak aktuális értéke.

FÉLIG-NAIV-DATALOG(P,$\mathcal{I}$)

```
 1  P' ← azon P-beli szabályok, amelyek testében nincs idb reláció
 2  for S ∈ idb(P)
 3      do S^0 ← ∅
 4         Δ_S^1 ← P'(I)(S)
 5  i ← 1
 6  repeat
 7         for S ∈ idb(P)
 8                ▷ T_1, ..., T_ℓ az S-t meghatározó szabályokban előforduló idb relációk.
 9             do S^i ← S^{i-1} ∪ Δ_S^i
10                Δ_S^{i+1} ← P_S^i(I, T_1^{i-1}, ..., T_ℓ^{i-1}, T_1^i, ..., T_ℓ^i, Δ_{T_1}^i, ..., Δ_{T_ℓ}^i) − S^i
11         i ← i + 1
12  until Δ_S^i = ∅ minden S ∈ idb(P)-re
13  for S ∈ idb(P)
14      do S ← S^i
15  return S
```

30.15. tétel. *A* FÉLIG-NAIV-DATALOG *helyesen számítja ki a P datalog program eredményét.*

Bizonyítás. i-szerinti teljes indukcióval belátjuk, hogy tetszőleges $S \in idb(P)$-re a 6–12. sorok ciklusának i-edik végrehajtása után az S^i értéke $T_P^i(\mathcal{I})(S)$, Δ_S^{i+1} pedig $T_P^{i+1}(\mathcal{I})(S) - T_P^i(\mathcal{I})(S)$-sel egyenlő. $T_P^i(\mathcal{I})(S)$ értelemszerűen a T_P közvetlen következmény operátor i-szeres alkalmazásával az $\mathcal{I}$ példányból kiindulva az S relációra kapott érték.

$i = 0$-ra a 4. sor pontosan $T_P(\mathcal{I})(S)$-t számítja ki minden $S \in idb(P)$-re. Az indukciós lépéshez mindössze azt kell látnunk, hogy $P_S^i(\mathcal{I}, T_1^{i-1}, \ldots, T_\ell^{i-1}, T_1^i, \ldots, T_\ell^i, \Delta_{T_1}^i, \ldots, \Delta_{T_\ell}^i) \cup S^i$ pontosan $T_P^{i+1}(\mathcal{I})(S)$-sel egyenlő, hiszen a 9–10. sorokban a FÉLIG-NAIV-DATALOG eljárás ennek használatával állítja elő S^i-t és Δ_S^{i+1}-et. Az indukciós feltétel szerint S^i értéke $T_P^i(\mathcal{I})(S)$, ehhez képest új sorokat csak úgy kaphatunk, ha valamely S-et meghatározó *idb* relációnak olyan sorait vesszük figyelembe, amelyek T_P legutolsó alkalmazásakor keletkeztek, ezek

pedig szintén az indukciós feltétel miatt a $\Delta^i_{T_1}, \ldots, \Delta^i_{T_\ell}$ relációkban vannak.

A 12. sor feltétele pontosan azt jelenti, hogy minden $S \in idb(P)$ reláció változatlan marad a T_P közvetlen következmény operátor alkalmazása során, tehát az algoritmus megtalálta annak minimális fixpontját. Az pedig a 30.14. definíció szerint pontosan a P datalog program eredménye az $\mathcal{I}$ bemeneti példányra. ■

A FÉLIG-NAIV-DATALOG eljárás ugyan sok felesleges számítást kiküszöböl, azonban bizonyos datalog programokon ez sem optimális (30.1-7. gyakorlat). Azonban a datalog program elemzésével és a számítás azon alapuló módosításával a legtöbb felesleges lépést meg lehet takarítani.

30.16. definíció. *Legyen P datalog program. P **előzmény gráfja** G_P a következő irányított gráf. Csúcshalmaza $idb(P)$ relációi, $R, R' \in idb(P)$ esetén (R, R') irányított él, ha létezik olyan szabály P-ben, amelynek feje R' és R a testében van. P **rekurzív,** ha G_P-ben van irányított kör. Az R és R' relációk **kölcsönösen rekurzívak,** ha G_P ugyanazon erősen összefüggő komponensébe esnek.*

A kölcsönös rekurzivitás ekvivalencia reláció $idb(P)$-be tartozó relációk halmazán. A JAVÍTOTT-FÉLIG-NAIV-DATALOG eljárás alapgondolata, hogy az $R \in idb(P)$ relációval "egyszerre" csak a vele kölcsönösen rekurzív relációkat kell számolni, minden más, R-t definiáló szabályban előforduló relációt már "előre" kiszámíthatunk, és *edb* relációnak tekinthetünk.

JAVÍTOTT-FÉLIG-NAIV-DATALOG(P,$\mathcal{I}$)

1 Határozzuk meg $idb(P)$ kölcsönös rekurzivitás szerinti ekvivalencia osztályait.
2 Készítsük el az $[R_1], [R_2], \ldots, [R_n]$ ekvivalencia osztályok listáját G_P topologikus rendezése szerint.
3 ▷ Minden $i < j$-re teljesül, hogy G_P-ben nincs irányított út R_j-ből R_i-be.
4 **for** $i \leftarrow 1$ **to** n
5 **do** Használjuk a FÉLIG-NAIV-DATALOG eljárást az $[R_i]$-beli relációk kiszámítására, az $[R_j]$-beli relációkat *edb* relációkként kezelve $j < i$-re.

Mélységi keresés alkalmazásával az 1–2. sorok $O(v_{G_P}+e_{G_P})$ időben végrehajthatók, ahol v_{G_P} és e_{G_P} a G_P gráf csúcs-, illetve élszámát jelölik. Az eljárás helyességének bizonyítását az Olvasóra bízzuk (30.1-8. gyakorlat).

30.1.3. Bonyolultsági kérdések lekérdezések közti tartalmazásról

A jelen részben visszatérünk a konjunktív lekérdezésekhez. Lekérdezések eredményének számításakor a legköltségesebb feladat relációk természetes összekapcsolásának elvégzése. Különösen igaz ez, ha a közös attribútumokhoz nincs index megadva, és így csak TELJES-SORONKÉNTI-ÖSSZEKAPCSOLÁS eljárás alkalmazható.

TELJES-SORONKÉNTI-ÖSSZEKAPCSOLÁS(R_1, R_2)

```
1  S ← ∅
2  for minden u ∈ R1
3      do for minden v ∈ R2
4             do if u és v összekapcsolható
5                    then S ← S ∪ {u ⋈ v}
6  return S
```

Világos, hogy a TELJES-SORONKÉNTI-ÖSSZEKAPCSOLÁS futási ideje $O(|R_1| \times |R_2|)$. Nem mindegy tehát, hogy egy lekérdezést milyen sorrendben számítunk ki, hiszen az eljárás folyamán különböző méretű relációk természetes összekapcsoltjait kell képezni. Táblázatos lekérdezések esetén a ***homomorfizmus tétel*** lehetőséget ad a lekérdezés olyan átírására, amelyik kevesebb összekapcsolást használ, mint az eredeti.

Az $\mathbf{R}$ séma feletti q_1, q_2 lekérdezésekre q_2 ***tartalmazza*** q_1-t, jelben $q_1 \sqsubseteq q_2$, ha minden $\mathbf{R}$ feletti $\mathcal{I}$ példányra $q_1(\mathcal{I}) \subseteq q_2(\mathcal{I})$ teljesül. $q_1 \equiv q_2$ a 30.1. definíció értelmében pontosan akkor, ha $q_1 \sqsubseteq q_2$ és $q_1 \sqsupseteq q_2$. Szükségünk lesz a kiértékelések általánosítására. ***Helyettesítésen*** olyan leképezést értünk, amelyik a változók halmazából képez a változók és a konstansok halmazának egyesítésébe, és amelyiket konstansokra identitásként terjesztünk ki. Természetesen értelmezhető a helyettesítés kiterjesztése szabad sorokra, illetve táblázatokra.

30.17. definíció. *Legyen $q = (\mathbf{T}, u)$ és $q' = (\mathbf{T}', u')$ két táblázatos lekérdezés az $\mathbf{R}$ séma felett. A θ helyettesítés* ***homomorfizmus*** *q'-ről q-ra, ha $\theta(\mathbf{T}') = \mathbf{T}$ és $\theta(u') = u$.*

30.18. tétel (homomorfizmus tétel). *Legyen $q = (\mathbf{T}, u)$ és $q' = (\mathbf{T}', u')$ két táblázatos lekérdezés az $\mathbf{R}$ séma felett. $q \sqsubseteq q'$ akkor és csak akkor, ha létezik homomorfizmus q'-ről q-ra.*

Bizonyítás. Tegyük fel először, hogy θ homomorfizmus q'-ről q-ra, és legyen $\mathcal{I}$ az $\mathbf{R}$ séma feletti példány. Legyen $w \in q(\mathcal{I})$. Ez pontosan akkor teljesül, ha létezik egy ν kiértékelés, amelyik a $\mathbf{T}$ táblát $\mathcal{I}$-be képezi és $\nu(u) = w$. Könnyen látható, hogy $\theta \circ \nu$ a $\mathbf{T}'$ táblát képezi $\mathcal{I}$-be és $\theta \circ \nu(u') = w$, azaz $w \in q'(\mathcal{I})$. Tehát $w \in q(\mathcal{I}) \Longrightarrow w \in q'(\mathcal{I})$, ami pontosan $q \sqsubseteq q'$-vel egyenértékű.

Másik oldalról, tegyük fel, hogy $q \sqsubseteq q'$. A bizonyítás gondolata, hogy q-t és q'-t is alkalmazzuk a $\mathbf{T}$ „példányra”. q eredménye az u szabad sor, tehát q' eredménye szintén tartalmazza az u sort, vagyis létezik $\mathbf{T}'$ egy θ beágyazása $\mathbf{T}$-be, amelyik u'-t u-ra képezi. A gondolatmenet szabatossá tételéhez elkészítjük a $\mathbf{T}$-vel izomorf $\mathcal{I}_\mathbf{T}$ példányt.

Legyen V a $\mathbf{T}$-ben előforduló változók halmaza. Minden $x \in V$-hez rendeljük az a_x konstanst, ami különbözik a $\mathbf{T}$-ben illetve $\mathbf{T}'$-ben előforduló konstansoktól, valamint $x \neq x' \Longrightarrow a_x \neq a_{x'}$. Legyen μ az a kiértékelés, amelyik $x \in V$-hez a_x-t rendeli, valamint legyen $\mathcal{I}_\mathbf{T} = \mu(\mathbf{T})$. μ bijekció V-ről $\mu(V)$-re, és $\mu(V)$-ben nem fordulnak elő $\mathbf{T}$-beli konstansok, ezért μ^{-1} jól definiált az $\mathcal{I}_\mathbf{T}$-ben előforduló konstansokon.

Világos, hogy $\mu(u) \in q(\mathcal{I}_\mathbf{T})$, tehát $q \sqsubseteq q'$ alapján $\mu(u) \in q'(\mathcal{I}_\mathbf{T})$ is teljesül. Azaz, létezik egy ν kiértékelés, ami a $\mathbf{T}'$ táblát beágyazza $\mathcal{I}_\mathbf{T}$-be úgy, hogy $\nu(u') = \mu(u)$. Könnyen látható, hogy $\nu \circ \mu^{-1}$ homomorfizmus q'-ről q-ra. ■

Lekérdezés optimalizálás tábla minimalizálással

A 30.6. tétel alapján táblázatos lekérdezések és a kielégíthető relációs algebrai lekérdezések (kivonás nélkül) ekvivalensek. A bizonyítás során kiderül, hogy a táblázatos lekérdezéssel ekvivalens relációs algebra kifejezés $\pi_{\vec{a}}(\sigma_F(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k))$ alakú, ahol k a tábla sorainak száma. Ebből következik, hogy ha a természetes összekapcsolások számát akarjuk minimalizálni, akkor az ekvivalens tábla sorainak számát kell a lehető legkisebbre csökkenteni.

A $(\mathbf{T}, u)$ táblázatos lekérdezés ***minimális,*** ha nincs olyan $(\mathbf{S}, v)$ lekérdezés, amelyik ekvivalens $(\mathbf{T}, u)$-val és $|\mathbf{S}| < |\mathbf{T}|$, azaz $\mathbf{S}$-nek kevesebb sora van. Meglepő, de igaz, hogy a $(\mathbf{T}, u)$-val ekvivalens minimális lekérdezés megkapható egyszerűen néhány sor elhagyásával $\mathbf{T}$-ből.

30.19. tétel. *Legyen $q = (\mathbf{T}, u)$ táblázatos lekérdezés. Van $\mathbf{T}'$ részhalmaza $\mathbf{T}$-nek, hogy $q' = (\mathbf{T}', u)$ minimális és ekvivalens $q = (\mathbf{T}, u)$-val.*

Bizonyítás. Legyen $(\mathbf{S}, v)$ minimális, q-val ekvivalens lekérdezés. A Homomorfizmus tétel szerint létezik θ homomorfizmus q-ról $(\mathbf{S}, v)$-re, valamint λ $(\mathbf{S}, v)$-ről q-ra. Legyen $\mathbf{T}' = \theta \circ \lambda(\mathbf{S})$. Könnyen ellenőrizhető, hogy $(\mathbf{T}', u) \equiv q$ és $|\mathbf{T}'| \leq |\mathbf{S}|$. Azonban $(\mathbf{S}, v)$ minimális, így $(\mathbf{T}', u)$ is az. ■

30.6. példa. *Tábla minimalizálás alkalmazása.* Tekintsük az $\{A, B, C\}$ attribútum halmazú $\mathbf{R}$ séma feletti

$$q = \pi_{AB}(\sigma_{B=5}(R)) \bowtie \pi_{BC}(\pi_{AB}(R) \bowtie \pi_{AC}(\sigma_{B=5}(R))) \tag{30.34}$$

relációs algebrai lekérdezést. A q-nak megfelelő táblás lekérdezés a következő $\mathbf{T}$ tábla:

	A	B	C
R	x	5	z_1
	x_1	5	z_2
	x_1	5	z
u	x	5	z

(30.35)

Olyan homomorfizmust keresünk, ami a $\mathbf{T}$ tábla néhány sorát $\mathbf{T}$ más soraira képezi, ezáltal mintegy "összehajtogatja" a táblát. Az első sor nem hagyható el, mert a homomorfizmus az u szabad soron azonosság, tehát x-t önmagának kell megfeleltesse. Hasonló a helyzet a harmadik sorral, mert z-nek is saját maga a képe minden homomorfizmusnál. Azonban a második sort ki lehet küszöbölni, x_1-t x-re, z_2-t z-re képezve. Tehát a $\mathbf{T}$-vel ekvivalens minimális tábla $\mathbf{T}$ első és harmadik sorát tartalmazza. Visszaírva algebrai lekérdezésre,

$$\pi_{AB}(\sigma_{B=5}(R)) \bowtie \pi_{BC}(\sigma_{B=5}(R)) \tag{30.36}$$

az eredmény. A (30.36) lekérdezés a (30.34) lekérdezéshez képest eggyel kevesebb összekapcsolás műveletet tartalmaz.

A következő tétel azt mondja ki, hogy táblák közti tartalmazás és ekvivalencia eldöntésének kérdése NP-teljes, következésképpen a tábla minimalizálás NP-nehéz feladat.

30.20. tétel. *Adott q és q' táblázatos lekérdezések esetén az alábbi döntési feladatok NP-teljesek:*

30.10. $q \sqsubseteq q'$?

30.11. $q \equiv q'$*?*

30.12. Tegyük fel, hogy q*-ból néhány szabad sor elhagyásával kaptuk* q'*-t. Igaz-e ekkor, hogy* $q \equiv q'$*?*

Bizonyítás. A Pontos fedés feladatot vezetjük vissza a különböző tábla feladatokra. A Pontos fedés feladat bemenete egy $X = \{x_1, x_2, \ldots, x_n\}$ halmaz, valamint részhalmazainak $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ rendszere. Eldöntendő, hogy létezik-e olyan $\mathcal{S}' \sqsubseteq \mathcal{S}$, hogy az $\mathcal{S}'$-beli részhalmazok pontosan lefedik X-et (azaz, minden $x \in X$-hez pontosan egy $S \in \mathcal{S}'$ létezik, amelyre $x \in S$). A Pontos fedés ismert NP-teljes feladat.

Legyen $\mathcal{E} = (X, \mathcal{S})$ a Pontos fedés bemenete. Vázolunk egy konstrukciót, ami $\mathcal{E}$-hez táblázatos $q_\mathcal{E}, q'_\mathcal{E}$ lekérdezés párt készít polinomiális időben. Ezt a konstrukciót lehet aztán a különböző NP-teljességi állítások bizonyítására használni.

Az $\mathbf{R}$ séma attribútumai legyenek a páronként különböző $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m$ attribútumok. $q_\mathcal{E} = (\mathbf{T}_\mathcal{E}, t)$ és $q'_\mathcal{E} = (\mathbf{T}'_\mathcal{E}, t)$ az $\mathbf{R}$ séma feletti táblázatos lekérdezések, mindkettő összegzése a $t = \langle A_1 : a_1, A_2 : a_2, \ldots, A_n : a_n \rangle$ szabad sor, ahol $a_1, a_2, \ldots, a_n$ páronként különböző változók.

Legyenek $b_1, b_2, \ldots, b_m, c_1, c_2, \ldots c_m$ további páronként különböző változók. $\mathbf{T}_\mathcal{E}$ n sorból áll, X minden elemének megfelel egy. Az x_i elem sorában a_i áll az A_i attribútum oszlopában, b_j a B_j attribútum oszlopában minden olyan j-re, amelyre $x_i \in S_j$ teljesül. A $\mathbf{T}_\mathcal{E}$ n tábla többi helyén csupa különböző új változó áll.

Hasonlóan, $\mathbf{T}'_\mathcal{E}$ m sorból áll, $\mathcal{S}$ minden elemének megfelel egy. Az S_j részhalmaz sorában a_i áll az A_i attribútum oszlopában, minden olyan i-re, amelyre $x_i \in S_j$, valamint $c_{j'}$ a $B_{j'}$ attribútum oszlopában, minden $j' \neq j$-re. A $\mathbf{T}'_\mathcal{E}$ n tábla többi helyén csupa különböző új változó áll.

A 30.10. kérdés NP-teljessége következik abból, hogy X-nek akkor és csak akkor létezik pontos fedése $\mathcal{S}$-beli halmazokkal, ha $q'_\mathcal{E} \sqsubseteq q_\mathcal{E}$ teljesül. A bizonyítást, valamint a 30.11. és 30.12. kérdések NP-teljességének bizonyítását az Olvasóra bízzuk (30.1-9. gyakorlat). ■

Gyakorlatok

30.1-1. Bizonyítsuk be a 30.4. állítást, azaz hogy minden szabály alapú q lekérdezés monoton és kielégíthető. *Útmutatás.* A kielégíthetőség bizonyításához legyen a q lekérdezésben szereplő összes konstans halmaza K, $a \notin K$ pedig egy további konstans. Minden (30.3)-beli R_i reláció sémához készítsük el az összes olyan $(a_1, a_2, \ldots, a_r)$ sort, ahol $a_i \in K \cup \{a\}$, és r az R_i attribútumainak száma. Legyen $\mathcal{I}$ az így kapott példány. Lássuk be, hogy $q(\mathcal{I})$ nem üres.

30.1-2. Adjunk meg egy $\mathbf{R}$ relációs sémát és q relációs algebrai lekérdezést $\mathbf{R}$ felett, amelynek eredménye üres halmaz tetszőleges $\mathbf{R}$ feletti példányra.

30.1-3. Bizonyítsuk be, hogy a szabály alapú lekérdezések nyelve és a táblázatos lekérdezések nyelve ekvivalens.

30.1-4. Bizonyítsuk be, hogy minden szabály alapú q lekérdezés, amelyik egyenlőség atomokat is tartalmazhat, vagy ekvivalens a $q^\emptyset$ üres lekérdezéssel, vagy létezik egy q' szabály alapú lekérdezés, amely nem tartalmaz egyenlőség atomokat úgy, hogy $q \equiv q'$. Adjunk polinomiális algoritmust, ami egy adott egyenlőségeket is tartalmazó szabály alapú q lekérdezésről eldönti, hogy $q \equiv q^\emptyset$ teljesül-e, és ha nem, akkor elkészít egy q' szabály alapú lekérdezést, amely nem tartalmaz egyenlőség atomokat úgy, hogy $q \equiv q'$.

30.1-5. A 30.6. tétel bizonyításának gondolatát általánosítva bizonyítsuk be a 30.8. tételt.
30.1-6. Legyen P datalog program, $\mathcal{I}$ példány $edb(P)$ felett, $C(P,\mathcal{I})$ az $\mathcal{I}$-ben és P-ben szereplő konstansok (véges) halmaza. Legyen $\mathbf{B}(P,\mathcal{I})$ az alábbi $sch(P)$ feletti példány:

1. Minden $edb(P)$-beli R relációra az $R(u)$ tény $\mathbf{B}(P,\mathcal{I})$-ben van pontosan akkor, ha $\mathcal{I}$-ben van, valamint
2. minden $idb(P)$-beli R relációra minden $C(P,\mathcal{I})$-beli konstansokból képzett $R(u)$ tény $\mathbf{B}(P,\mathcal{I})$-ben van.

Bizonyítsuk be, hogy

$$\mathcal{I} \subseteq T_P(\mathcal{I}) \subseteq T_P^2(\mathcal{K}) \subseteq T_P^3(\mathcal{K}) \subseteq \cdots \subseteq \mathbf{B}(P,\mathcal{I}). \tag{30.37}$$

30.1-7. Adjunk példát olyan bemenetre, P datalog programra és $\mathcal{I}$ edb példányra, amelyre ugyanazt a sort a FÉLIG-NAIV-DATALOG ciklusának különböző végrehajtásai is előállítják.
30.1-8. Bizonyítsuk be, hogy a JAVÍTOTT-FÉLIG-NAIV-DATALOG eljárás minden bemenetre véges időben megáll, és helyes eredményt ad. Mutassunk példát olyan bemenetre, amelyiken a JAVÍTOTT-FÉLIG-NAIV-DATALOG kevesebb sort számít ki többször, mint a FÉLIG-NAIV-DATALOG eljárás.
30.1-9.

1. Bizonyítsuk be, hogy a 30.20. tétel bizonyításában szereplő $q_{\mathcal{E}} = (\mathbf{T}_{\mathcal{E}}, t)$ és $q'_{\mathcal{E}} = (\mathbf{T}'_{\mathcal{E}}, t)$ táblázatos lekérdezésekre pontosan akkor létezik homomorfizmus $(\mathbf{T}_{\mathcal{E}}, t)$-ről $(\mathbf{T}'_{\mathcal{E}}, t)$-re, ha az $\mathcal{E} = (X, \mathcal{S})$ PONTOS FEDÉS feladatnak van megoldása.
2. Bizonyítsuk be, hogy a 30.11. és 30.12. kérdések eldöntése NP-teljes feladat.

30.2. Nézetek

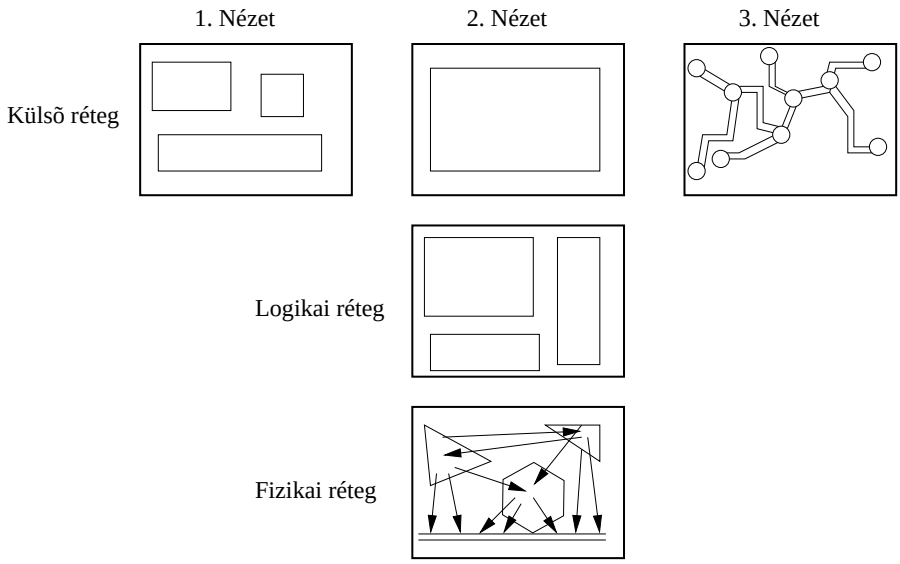
Egy adatbázis rendszer felépítésének három fő szintje van:

- fizikai réteg;
- logikai réteg;
- külső (felhasználói) réteg.

A szintek elválasztásának célja a fizikai adatfüggetlenség elérése, valamint a felhasználói kényelem. A 30.2. ábrán a három nézet lehetséges felhasználói felületeket mutat: multirelációs, univerzális relációs, illetve grafikus felület.

A ***fizikai réteg*** a ténylegesen tárolt adatállományokat, a rájuk épített sűrű és ritka indexeket jelenti.

A ***logikai réteg*** elválasztása a fizikai rétegtől lehetővé teszi, hogy a felhasználó az adatok logikai összefüggéseire koncentráljon, ami sokkal jobban közelíti a modellezni kívánt valóságról alkotott képét. A logikai réteghez tartozik az adatbázis séma leírása a különböző integritási feltételekkel, függőségekkel. Ez az a szint, ahol az adatbázis adminisztrátorok kezelik a rendszert. A logikai és a fizikai réteg közti kapcsolatot az adatbáziskezelő program



30.2. ábra. Az adatbázis rendszerek felépítésének három szintje.

tartja fent.

A ***külső réteg*** és a logikai réteg elválasztásának célja, hogy a végfelhasználók az adatbázist a saját (szűk) igényeik és szempontjaik szerint lássák. Például egy banki adatbázis esetén a külső réteg egyik nagyon egyszerű nézete lehet a pénzkiadó automata, vagy egy jóval bonyolultabb nézete lehet kölcsön igénylés elbírálásához az ügyfél teljes banki hitel története.

30.2.1. Nézet, mint lekérdezés eredménye

Kérdés az, hogy a külső réteghez tartozó nézeteket hogyan adhatjuk meg. Ha egy relációs algebrai kifejezéssel adott lekérdezést úgy tekintünk, mint valamely formulát, amit majd reláció példányokra alkalmazunk, akkor kapjuk a ***nézetet***. A datalog szabályok jól szemléltetik a lekérdezések és nézetek közti különbséget. A szabályok által meghatározott relációkat ***intenzionálisnak*** neveztük, mivel ezek azok a relációk, amelyeknek nem kell külső tárolókon, extenzionálisan létezniük, szemben az ***extenzionális*** relációkkal.

30.21. definíció. *Az* **R** *séma feletti valamely* $\mathcal{Q}$ *lekérdezési nyelven megadott* $\mathcal{V}$ *kifejezést* **R** *séma feletti* ***nézetnek*** *nevezünk.*

Természetesen datalog intenzionális relációkhoz hasonlóan nézeteket is használhatunk lekérdezések megadásakor, illetve újabb nézetek meghatározásakor.

30.7. példa. *SQL nézet.* Az SQL adatbázis-kezelő nyelvben az alábbi módon lehet nézetet megadni. Tegyük fel, hogy a **PestiMűsor** sémából számunkra érdekes adat csak annyi, hogy mikor és hol játszanak Kuroszava filmeket. A **KuroszavaIdőpontok** nézetet

KUROSZAVAIDŐPONTOK

```
1  create view KuroszavaIdőpontok as
2          select Mozi, Időpont
3          from Filmek, Műsor
4          where Filmek.Cím=Műsor.Cím and Filmek.Rendező="Kuroszava Akira"
```

SQL utasítás definiálja. Relációs algebrai alakban a következő.

$$KuroszavaIdőpontok(Mozi, Időpont) = \pi_{Mozi, Időpont}(Mozik \bowtie \sigma_{Rendező=\text{"Kuroszava Akira"}}(Filmek)) \tag{30.38}$$

Végül datalog szabállyal ugyanez

$$KuroszavaIdőpontok(x_M, x_I) \leftarrow Mozik(x_M, x_C, x_I), Filmek(x_C, \text{"Kuroszava Akira"}, x_{Sz}). \tag{30.39}$$

A KUROSZAVAIDŐPONTOK utasítás 2. sora jelöli ki a használt szelekciós operátort, a 3. sor, hogy melyik két relációt kell összekapcsolni, végül a 4. sor feltétele mutatja meg, hogy természetes összekapcsolásról van szó, nem pedig direkt szorzatról.

Amennyiben a $\mathcal{V}$ nézetet már definiáltuk, akkor a további lekérdezésekben, illetve nézet definíciókban ugyanúgy alkalmazható, mint akármilyen másik (extenzionális) reláció.

Nézetek használatának előnyei

- Adatok automatikus elrejtése: Olyan adatok, amelyek nem részei a használt nézetnek, nyilván nem is jelennek meg a felhasználó előtt, így azokat nem is olvashatja illetéktelenül, illetve nem is módosíthatja. Tehát azzal, hogy az adatbázis hozzáférést nézeteken keresztül engedjük meg, egyszerű, de hatékony biztonsági mechanizmust üzemeltetünk.
- Nézetek egyszerű „makró képességet” szolgáltatnak. A 30.7. példában definiált KuroszavaIdőpontok nézet használatával könnyedén meg tudjuk keresni, melyik moziban adnak délelőtt Kuroszava filmet:

  $$KuroszavaDélelőtt(Mozi) \leftarrow KuroszavaIdőpontok(Mozi, x_I), x_I < 12\,. \tag{30.40}$$

  Természetesen a felhasználó beírhatná a kódba a *KuroszavaIdőpontok* definícióját közvetlenül, de a makró utasításokkal szoros hasonlatosságban, itt is a kényelmi szempontok az elsők.
- A nézetek lehetővé teszik, hogy ugyanazt az adatot különböző felhasználók különböző módon lássák ugyanabban az időben.
- Nézetek biztosítják a ***logikai adatfüggetlenséget.*** A logikai adatfüggetlenség lényege, hogy a felhasználók és programjaik védettek legyenek az adatbázis séma szerkezeti változtatásaitól. Ezt úgy lehet elérni, hogy a szerkezet–változtatás előtti relációkat mint nézetek definiáljuk a szerkezet változtatás utáni új sémában.
- A nézetek lehetővé teszik az ellenőrzött adatbevitelt is. SQL-ben a **create view** utasítás **with check option** záradéka ezt a célt szolgálja.

Materializált nézet

Előfordulhat, hogy valamely nézetet több különböző lekérdezésben is használunk. Ilyen

esetekben hasznos lehet, ha nézet által meghatározott reláció(k) sorait nem kell minden esetben újra kiszámolnunk, hanem a nézet definíciójában szereplő lekérdezés eredményét tároljuk, és a további feldolgozásokkor csak beolvassuk. Az ilyen tárolt eredményt nevezzük ***materializált nézetnek.***

Gyakorlatok

30.2-1. Tekintsük az alábbi sémát:

Filmsztár(*Név*,*Cím*,*Nem*,*SzülDátum*)
FilmMogul(*Név*,*Cím*,*Igazolvány#*,*Vagyon*)
Stúdió(*Név*,*Cím*,*ElnökIg#*) .

A *FilmMogul* reláció a filmszakma nagymenőinek (stúdió elnökök, producerek, stb) adatait tartalmazza. Az egyes attribútumok nevei értelemszerűen megadják jelentésüket, illetve az *Igazolvány#* a mogul működési engedélyének száma, az *ElnökIg#* a stúdió elnöke működési engedélyének száma. Adjuk meg az alábbi nézeteket datalog szabállyal, relációs algebrai kifejezéssel, illetve SQL nyelven:

1. *GazdagMogul*: a legalább 100,000,000 forint vagyonú filmmogulok nevét, címét, igazolvány számát és vagyonát listázza.
2. *StúdióElnök*: az olyan filmmogulok nevét, címét, igazolvány számát listázza, akik stúdió elnökök is egyben.
3. *MogulSztár*: az összes olyan személy nevét, címét, igazolvány számát és vagyonát listázza, aki egyszerre filmsztár és filmmogul is.

30.2-2. A **PestiMűsor** séma felett adjuk meg az alábbi nézeteket datalog szabállyal, relációs algebrai kifejezéssel, illetve SQL nyelven:

1. *Marilyn*(*Cím*): Marilyn Monroe szereplésével készült filmek címét listázza.
2. *CorvinInfo*(*Cím*,*Időpont*,*Telefon*): a Corvin moziban játszott filmek címét, vetítési időpontjait, valamint a mozi telefonszámát listázza.

30.3. Lekérdezés átírás

A lekérdezések megválaszolása nézetek felhasználásával, más néven lekérdezések átírása nézetek felhasználásával a közelmúltban nagyon sok figyelmet és érdeklődést kiváltó feladattá vált. Ennek oka a feladat széles körű alkalmazhatósága a legkülönfélébb adatkezelési feladatokban: lekérdezés optimalizálásban, fizikai adatfüggetlenség megvalósításában, adat- , illetve információ-egyesítésnél, valamint adattárházak tervezésénél.

A feladat lényege a következő. Tegyük fel, hogy az **R** séma felett adott a Q lekérdezés, valamint $V_1, V_2, \ldots, V_n$ nézetek. Meg lehet-e válaszolni a Q lekérdezést pusztán a $V_1, V_2, \ldots, V_n$ nézetek eredményeinek ismeretében? Avagy, mi azon sorok legbővebb halmaza, amit a nézetek ismeretében meg tudunk határozni? Ha elérhetjük a nézeteket és az alapséma relációit is, melyik a legolcsóbb kiszámítási mód Q megválaszolására?

30.3.1. Motiváció

Mielőtt a lekérdezés átírás algoritmusait részletesen tárgyalnánk, néhány alkalmazás bemutatásával indokoljuk, hogy miért érdemes a kérdéssel foglalkozni. A példákhoz az alábbi **Egyetem** adatbázist használjuk:

$$\textbf{Egyetem} = \{\mathit{Tanár}, \mathit{Kurzus}, \mathit{Tanít}, \mathit{Felvett}, \mathit{Szakirány}, \mathit{Dolgozik}, \mathit{Témavezető}\}\ . \tag{30.41}$$

Ahol az egyes relációk sémái a következőek:

$$\begin{aligned} \mathit{Tanár} &= \{\mathit{TNév}, \mathit{Szakterület}\} \\ \mathit{Kurzus} &= \{\mathit{K\text{-}szám}, \mathit{Cím}\} \\ \mathit{Tanít} &= \{\mathit{TNév}, \mathit{K\text{-}szám}, \mathit{Félév}, \mathit{Véleményezés}\} \\ \mathit{Felvett} &= \{\mathit{Diák}, \mathit{K\text{-}szám}, \mathit{Félév}\} \\ \mathit{Szakirány} &= \{\mathit{Diák}, \mathit{Tanszék}\} \\ \mathit{Dolgozik} &= \{\mathit{TNév}, \mathit{Tanszék}\} \\ \mathit{Témavezető} &= \{\mathit{TNév}, \mathit{Diák}\}\ . \end{aligned} \tag{30.42}$$

Feltesszük, hogy a tanárokat, diákokat és tanszékeket a nevük egyértelműen meghatározza, valamint a kurzusokat a számuk. A *Felvett* reláció sorai azt mutatják, hogy melyik diák melyik tárgyat melyik félévben vette fel, a *Szakirány* pedig azt, hogy melyik tanszéket választotta szakosodáskor (feltesszük az egyszerűség kedvéért, hogy egy tanszéken csak egy szakirány van hirdetve).

Lekérdezés optimalizálás
Ha egy lekérdezés megválaszolásához szükséges számítások egy részét már előzőleg elvégeztük és tároltuk valamely materializált nézetben, akkor használhatjuk a nézetet a lekérdezés megválaszolásának meggyorsítására.

Tekintsük azt a lekérdezést, amelyik azokat a (*Diák*,*Cím*) párokat keresi, ahol a diák az adott doktorandusz tárgyat felvette, a tárgyat *adatbázis* szakterületű tanár tanítja (választható tárgyak K-száma legalább 400, ebből a doktorandusz tárgyak azok, amelyek K-száma legalább 500).

$$\begin{aligned} \mathit{val}(x_D, x_C) \quad \leftarrow \quad & \mathit{Tanít}(x_T, x_K, x_F, y_1), \mathit{Tanár}(x_T, \text{“adatbázis”}), \\ & \mathit{Felvett}(x_D, x_K, x_F), \mathit{Kurzus}(x_k, x_C), x_K \geq 500\ . \end{aligned} \tag{30.43}$$

Tegyük fel, hogy rendelkezésre áll az alábbi materializált nézet, amelyik választható tárgyak regisztrációs adatait tartalmazza

$$\mathit{Választható}(x_D, x_C, x_K, x_F) \leftarrow \mathit{Felvett}(x_D, x_K, x_F), \mathit{Kurzus}(x_K, x_C), x_K \geq 400. \tag{30.44}$$

A *Választható* nézet felhasználható (30.43) megválaszolására

$$\begin{aligned} \mathit{val}(x_D, x_C) \quad \leftarrow \quad & \mathit{Tanít}(x_T, x_K, x_F, y_1), \mathit{Tanár}(x_T, \text{“adatbázis”}), \\ & \mathit{Választható}(x_D, x_C, x_K, x_F), x_K \geq 500\ . \end{aligned} \tag{30.45}$$

(30.45) kiszámítása gyorsabb lesz, mint (30.43)-é, mivel a *Felvett* és a *Kurzus* természetes összekapcsolását már a *Választható* nézet elvégezte, valamint leválasztotta a kötelező tárgyakat (amelyek a regisztráció legnagyobb részét teszik ki a legtöbb egyetemen). Érdemes

megjegyezni, hogy a *Választható* nézet annak ellenére használható, hogy ***szintaktikusan*** a (30.43) lekérdezés egyetlen részével sem egyezik meg.

Másfelől azonban előfordulhat, hogy az eredeti lekérdezést gyorsabban tudjuk megválaszolni. Ha a *Felvett* és a *Kurzus* relációknak a *K-szám* attribútumon létezik indexük, viszont a *Választható*-hoz semmilyen index sincs felépítve, akkor gyorsabb lehet a (30.43) lekérdezést közvetlenül az adatbázis relációkból számítani. Az igazi kihívás tehát nem csak az, hogy eldöntsük egy materializált nézetről, hogy logikailag használható-e valamely lekérdezés megválaszolására, hanem alapos költség elemzést kell végeznünk, hogy mikor érdemes használni a létező nézeteket.

Fizikai adatfüggetlenség

A mai modern adatbázis rendszerek egyik alapelve az adatok logikai szerkezetének és fizikai tárolási módjának szétválasztása. A relációs adatbázis rendszerek esetében, eltekintve a relációk vízszintes vagy függőleges szétvágásától, még mindig lényegileg egy-az-egyhez megfeleltetés van a séma relációi és az őket tartalmazó fizikai állományok között. Objektum orientált rendszerek esetében a fizikai–logikai elválasztás elengedhetetlen, mivel a logikai séma jelentős redundanciát tartalmaz, ezért nem felel meg jó fizikai elhelyezésnek. Másik példa a fizikai adatfüggetlenség fontosságára az, amikor a logikai modellt mint közbenső réteget határozzuk meg azután, hogy a fizikai megjelentetést már eldöntöttük. Ez általános amikor XML adatokat tárolunk relációs adatbázisokban, illetve adat egyesítésnél. A STORED rendszer például XML adatokat tárol relációs adatbázisban úgy, hogy nézeteket használ az „XML→relációk” leképezés leírására.

A fizikai adatfüggetlenség fenntartására az egyik elterjedt módszer, hogy az adat tényleges fizikai tárolását a logikai séma feletti nézetek segítségével írjuk le. Például Tsatalos, Solomon és Ioannidis ÁTEU-kat (Általánosított Többszintű Elérési Utakat) használnak az adattárolás leírására.

Az ÁTEU leírja a tárolási szerkezet fizikai szervezését és indexeit. Az első kulcsszó (**as**) leírja a használt adatszerkezetet, amelyben a sorokat tárolják (B^+-fa, hasítási index, stb). A leírás többi része a tartalmat adja meg, nézetek megadásához nagyon hasonlóan. A **given** és a **select** kulcsszavak leírják a rendelkezésre álló attribútumokat, ahol a **given** adja meg, hogy melyik attribútumok alapján indexeljük a struktúrát. A **where** kulcsszóval adott nézet definícióban infix jelölést használunk.

A 30.3. ábrán látható példában az A1 ÁTEU olyan (*Diák,Tanszék*) párokat tartalmaz, ahol a *Diák* a *Tanszéket* választotta szakosodáskor. Ezek a párok egy B^+ fával vannak indexelve a *Diák.név* attribútumon. Az A2 ÁTEU egy indexet tárol a diákok nevéből azon kurzusok *K-számaiba,* amelyeket felvettek. Az A3 ÁTEU a kurzusok *K-számaiból* azon *Tanszékekbe* mutató indexet tartalmaz, amely *Tanszékeknél* szakosodott diákok felvették az adott tárgyat.

Mivel az adatokat az ÁTEU-kban leírt adatszerkezetekben tároljuk, felmerül a kérdés, hogyan lehet ezeket az adatszerkezeteket felhasználni lekérdezések megválaszolására. Az ÁTEU-k logikai tartalmát nézetek írják le, ezért a lekérdezés megválaszolása pontosan az a feladat, hogy találjuk meg a lekérdezésnek egy olyan átírását, ami az adott nézeteket használja. Ha több átírás is lehetséges, akkor a legolcsóbb átírást keressük. Jegyezzük meg, hogy a lekérdezés optimalizálással szemben itt ***szükségszerűen*** használnunk kell a nézeteket, mivel az adatokat ÁTEU-kban tároljuk.

def.áteu A1 **as** b$^+$-fa **by**
given *Diák.Név*
select *Tanszék*
where *Diák* szakosodik *Tanszék*

def.áteu A2 **as** b$^+$-fa **by**
given *Diák.Név*
select *Kurzus.K-szám*
where *Diák* felvett *Kurzus*

def.áteu A3 **as** b$^+$-fa **by**
given *Kurzus.K-szám*
select *Tanszék*
where *Diák* felvett *Kurzus* **and** *Diák* szakosodik *Tanszék*

30.3. ábra. Az **Egyetem** tartomány ÁTEU-i.

Tekintsük a következő lekérdezést, amelyik azon diákok nevét és a szakosodásnál választott tanszékét kérdezi, akik doktorandusz kurzust vettek fel.

$$val(Diák,Tanszék) \leftarrow Felvett(Diák,K\text{-}szám,y), Szakirány(Diák,Tanszék), K\text{-}szám \geq 500. \tag{30.46}$$

A lekérdezést két módon is megválaszolhatjuk. Először, mivel a *Diák.név* egyértelműen meghatározza a diákot, vehetjük A1 és A2 természetes összekapcsolását, majd alkalmazhatunk egy kiválasztási operátort, amivel kiválasztjuk azokat a sorokat, amelyekre *K-szám* $\geq$ 500, végül egy vetítéssel kiküszöbölhetjük a szükségtelen attribútumokat. Másik végrehajtási terv lehet, hogy A3-at és A2-t kapcsoljuk össze, majd elvégezzük a *K-szám* $\geq$ 500 szelekciót. Ez utóbbi megoldás hatékonyabb lehet, mert A3 tartalmaz indexet a *K-szám* attribútumon, így a közbenső összekapcsolások sokkal gyorsabbak lehetnek.

Adategyesítés

Egy ***adategyesítési rendszer*** (más néven ***adatközvetítő rendszer***) célja, hogy ***egységes*** lekérdezési felületet biztosítson nagyszámú és eltérő szerkezetű adatforráshoz. Legfontosabb példák a vállalati integráció, több különböző webes forrás egyidejű lekérdezése, valamint elosztott tudományos kísérletek eredményének egyesítése.

Az egységes lekérdezési felület elérése érdekében az adategyesítési rendszer a felhasználó felé egy ***közvetített sémát*** mutat. A közvetített séma ***virtuális*** relációkból áll, abban az értelemben, hogy fizikai valóságukban ezeket a relációkat sehol nem tárolják ebben a formában. A közvetített sémát az adategyesítési alkalmazás szempontjából kell egyedileg megtervezni. Ahhoz, hogy a lekérdezéseket meg tudja válaszolni, a rendszernek tartalmaznia kell ***forrás leírásokat.*** Egy adatforrás leírása meghatározza a forrás tartalmát, a benne megtalálható attribútumokat, valamint a forrás tartalmára vonatkozó integritási feltételeket. Az adatforrás leírás egyik elterjedt megközelítési módja, hogy a forrás tartalmát a közvetített séma feletti ***nézetként*** adjuk meg. Ez lehetővé teszi új adatforrások beillesztését, illetve az integritási feltételek megadását.

A lekérdezés megválaszolásához az adategyesítési rendszernek a közvetített sémában megfogalmazott lekérdezést le kell fordítania olyanra, amelyik közvetlenül az adatforrásokra hivatkozik. Mivel a források nézetként adottak, a fordítás azzal egyenértékű, hogy a lekérdezést nézetek segítségével válaszoljuk meg.

Példaként tekintsük az **Egyetem** sémát, mint a felhasználó felé közvetített sémát, azzal a különbséggel, hogy a *Tanít* és a *Kurzus* relációknak eggyel több attribútumuk van, nevezetesen az *Egyetem* attribútum, amelyik megmondja, hogy az adott tárgyat melyik egyetemen tanítják:

$$\begin{aligned} \mathit{Kurzus} &= \{\mathit{K\text{-}szám}, \mathit{Cím}, \mathit{Egyetem}\} \\ \mathit{Tanít} &= \{\mathit{TNév}, \mathit{K\text{-}szám}, \mathit{Félév}, \mathit{Véleményezés}, \mathit{Egyetem}\} \end{aligned} \tag{30.47}$$

Tegyük fel, hogy az alábbi két adatforrás áll rendelkezésre. Az első az összes "Adatbázisok" című tárgyakat, és azok előadóját listázza az összes egyetemről. A következő nézet meghatározással írható le:

$$\begin{aligned} \mathit{DBkurz}(\mathit{Cím}, \mathit{Tnév}, \mathit{K\text{-}szám}, \mathit{Egyetem}) \leftarrow & \mathit{Kurzus}(\mathit{K\text{-}szám}, \mathit{Cím}, \mathit{Egyetem}), \\ & \mathit{Tanít}(\mathit{TNév}, \mathit{K\text{-}szám}, \mathit{Félév}, \mathit{Véleményezés}, \mathit{Egyetem}), \\ & \mathit{Cím} = \text{“Adatbázisok”} \ . \end{aligned} \tag{30.48}$$

A második adatforrás a Budapesti Műszaki és Gazdaságtudományi Egyetem doktorandusz kurzusait adja meg, az alábbi módon.

$$\begin{aligned} \mathit{BMEPhD}(\mathit{Cím}, \mathit{Tnév}, \mathit{K\text{-}szám}, \mathit{Egyetem}) \leftarrow & \mathit{Kurzus}(\mathit{K\text{-}szám}, \mathit{Cím}, \mathit{Egyetem}), \\ & \mathit{Tanít}(\mathit{TNév}, \mathit{K\text{-}szám}, \mathit{Félév}, \mathit{Véleményezés}, \mathit{Egyetem}), \\ & \mathit{Egyetem} = \text{“BME”}, \mathit{K\text{-}szám} \geq 500. \end{aligned} \tag{30.49}$$

Ha az adategyesítési rendszertől azt kérdezzük, hogy ki tanít *Adatbázisokat* a Műegyetemen, akkor az a lekérdezést egyszerűen megválaszolhatja a *DBkurz* relációra alkalmazott szelekciós operátorral:

$$\mathit{Val}(\mathit{Tnév}) \leftarrow \mathit{DBkurz}(\mathit{Cím}, \mathit{Tnév}, \mathit{K\text{-}szám}, \mathit{Egyetem}), \mathit{Egyetem} = \text{"BME"} \ . \tag{30.50}$$

Azonban, ha azt szeretnénk tudni, hogy milyen választható (nem csak adatbázis) tárgyak léteznek a Műegyetemen, akkor az adategyesítési rendszer nem tudja a lekérdezés eredményéhez tartozó összes sort megtalálni, mivel csak a (30.48) és a (30.49) források állnak a rendelkezésére. Ehelyett, a források alapján meghatározható legbővebb sorhalmazt keresheti meg. Azaz, meghatározhatja az összes műegyetemi választható *adatbázis* kurzust a *DBkurz* forrásból, valamint a doktorandusz tárgyakat a *BMEPhD* forrásból. Tehát a következő nemrekurzív datalog program megadja a legbővebb választ:

$$\begin{aligned} \mathit{val}(\mathit{Cím}, \mathit{K\text{-}szám}) \leftarrow & \mathit{DBkurz}(\mathit{Cím}, \mathit{Tnév}, \mathit{K\text{-}szám}, \mathit{Egyetem}), \\ & \mathit{Egyetem} = \text{"BME"}, \mathit{K\text{-}szám} \geq 400 \\ \mathit{val}(\mathit{Cím}, \mathit{K\text{-}szám}) \leftarrow & \mathit{BMEPhD}(\mathit{Cím}, \mathit{Tnév}, \mathit{K\text{-}szám}, \mathit{Egyetem}) \ . \end{aligned} \tag{30.51}$$

Vegyük észre, hogy azok a választható tárgyak, amelyek nem doktorandusz tárgyak, vagy nem adatbázis témájúak, nem szerepelnek az eredményben. A lekérdezés optimalizálás és fizikai adatfüggetlenség esetében ***ekvivalens*** lekérdezés átírást kellett találni, itt olyan lekérdezés kifejezést keresünk, amelyik a nézetekből kapható ***legbővebb*** eredményhalmazt találja meg.

Szemantikus gyorstárolás
Kliens-szerver felépítésű adatbázis elérés esetén a kliens által már letöltött adatok szemantikusan modellezhetők, mint bizonyos nézetek eredményei. Tehát a kliens gépen eltárolt adatokat nem mint fizikai adategységeket, lapokat, sorokat, hanem mint nézeteket tekintjük. Ekkor annak eldöntésére, hogy a kliens újabb lekérdezésének megválaszolásához mely további adatok letöltése szükséges, a szervernek azt a feladatot kell megoldania, hogy a kliens oldalon létező nézetek segítségével a lekérdezés mely részei válaszolhatók meg.

30.3.2. Átírás bonyolultsági kérdései

Ebben az alfejezetben a lekérdezés átírás ***elméleti*** bonyolultságát tárgyaljuk. Elsősorban konjunktív lekérdezésekkel foglalkozunk. Megkülönböztetjük a ***minimális*** és a ***teljes*** átírásokat. Belátjuk, hogy ha a lekérdezés konjunktív és a nézetek is konjunktív lekérdezések materializált eredményeként adottak, akkor az átírási probléma *NP-teljes,* feltéve, hogy sem a lekérdezés, sem a nézetek nem tartalmaznak összehasonlítási atomokat. A konjunktív lekérdezéseket szabály alakban tekintjük, ez a legkényelmesebb számunkra.

Tegyük fel, hogy Q lekérdezés adott a $\mathbf{R}$ séma felett.

30.22. definíció. *A Q' konjunktív lekérdezés a Q lekérdezés $\mathcal{V} = V_1, V_2, \ldots, V_m$ nézeteket használó* ***átírása,*** *ha*

- *Q és Q' ekvivalensek, és*
- *Q' egy, vagy több $\mathcal{V}$-beli literált tartalmaz.*

Azt mondjuk, hogy Q' ***lokálisan minimális,*** *ha Q'-ből nem hagyható el literál anélkül, hogy az ekvivalencia megsérülne. Az átírás* ***globálisan minimális,*** *ha nem létezik kevesebb literált használó átírás. (A literálok számába az összehasonlítási atomok $=, \neq, \leq, <$ nem számítanak bele!)*

30.8. példa. *Lekérdezés átírás.* Tekintsük a következő Q lekérdezést és V nézetet.

$$\begin{array}{rcl} Q\colon q(X,U) & \leftarrow & p(X,Y), p_0(Y,Z), p_1(X,W), p_2(W,U) \\ V\colon v(A,B) & \leftarrow & p(A,C), p_0(C,B), p_1(A,D) \end{array} \tag{30.52}$$

Q átírható V használatával:

$$Q'\colon q(X,U) \leftarrow v(X,Z), p_1(X,W), p_2(W,U)\,. \tag{30.53}$$

A V nézet a Q lekérdezés első két literálját helyettesíti. Vegyük észre, hogy a lekérdezés harmadik literálját is biztosan kielégíti a nézet, azonban nem hagyható el az átírásból, mert a D változó már nem szerepel V fejében, ezért ha a p_1 literált is elhagynánk, akkor a p_1 és p_2 közti természetes összekapcsolást már nem kényszerítené ki semmi.

Mivel az alkalmazások egy részében az adatbázis relációkat nem érjük el, csak a nézeteket, például az adategyesítés és adattárházak esetében, ezért szükségünk van a ***teljes átírás*** fogalmára.

30.23. definíció. *A Q lekérdezés $\mathcal{V} = V_1, V_2, \ldots, V_m$ nézeteket használó Q' átírása* ***teljes átírás,*** *ha Q' csak $\mathcal{V}$-beli literálokat és összehasonlítási atomokat tartalmaz.*

30.9. példa. *Teljes átírás.* Tegyük fel, hogy a 30.8. példában szereplő V nézet mellett még a

$$V_2\colon v_2(A,B) \leftarrow p_1(A,C), p_2(C,B), p_0(D,E) \tag{30.54}$$

nézet is adott. A Q lekérdezés teljesen átírható:

$$Q''\colon q(X,U) \leftarrow v(X,Z), v_2(X,U)\,. \tag{30.55}$$

Fontos látnunk, hogy ezt az átírást nem kaphatjuk meg ***lépésenként,*** először csak V-t használva, majd megpróbálni V_2-t beépíteni (vagy éppen a másik sorrendben), hiszen p_0 reláció a V_2-ben szereplő p_0 reláció nem szerepel Q'-ben. Tehát a teljes átírás megtalálásához a két nézet használatát egyszerre, ***párhuzamosan*** kell tekinteni.

A lekérdezés átírás megtalálása szoros kapcsolatban áll a lekérdezések közti tartalmazás eldöntésének feladatával. Ez utóbbit táblázatos lekérdezésekre már a 30.1.3. pontban tárgyaltuk. Táblázatos lekérdezések közti homomorfizmus értelmezhető szabály alapú konjunktív lekérdezésekre is. Az egyetlen különbség, hogy nem követeljük meg ebben a fejezetben, hogy a szabály fejét a homomorfizmus a másik szabály fejére képezze. (A táblázatos lekérdezés összegző sorának a szabály feje felel meg.) A 30.20. tétel szerint annak eldöntése, hogy a Q_1 konjunktív lekérdezés tartalmazza-e Q_2 konjunktív lekérdezést, NP-teljes. Ez igaz marad akkor is, ha Q_2 összehasonlítási atomokat is tartalmaz. Azonban, ha Q_1 is tartalmaz összehasonlítási atomokat, akkor homomorfizmus létezése Q_1-ről Q_2-re csak elégséges feltételt ad a lekérdezések tartalmazására, amelyik ebben az esetben Π_2^p-teljes feladat. Ez utóbbi feladatosztály tárgyalása túlmutat a fejezet keretein, ezért nem részletezzük. A következő állítás szükséges és elégséges feltételt ad arra, hogy létezik-e a Q lekérdezésnek a V nézetet használó átírása.

30.24. állítás. *Tegyük fel, hogy Q és V konjunktív lekérdezések, amelyek tartalmazhatnak összehasonlítási atomokat is. Akkor és csak akkor létezik Q-nak V-t használó átírása, ha $\pi_\emptyset(Q) \subseteq \pi_\emptyset(V)$, azaz V vetítése az üres attribútum halmazra tartalmazza Q vetítését.*

Bizonyítás. Vegyük észre, hogy $\pi_\emptyset(Q) \subseteq \pi_\emptyset(V)$ ekvivalens az alábbi állítással: Ha V eredménye üres halmaz valamely adatbázis példányon, akkor Q eredménye is üres.

Tegyük fel először, hogy létezik átírás, azaz olyan Q-val ekvivalens szabály, amelynek testében V szerepel. Ha r olyan adatbázis példány, amelyiken V eredménye üres halmaz, akkor minden olyan szabály eredménye is üres, amelyiknek testében V szerepel.

Tegyük fel fordítva, hogy ha V eredménye üres halmaz valamely adatbázis példányon, akkor Q eredménye is üres. Legyen

$$\begin{aligned} Q\colon q(\tilde{x}) &\leftarrow q_1(\tilde{x}), q_2(\tilde{x}), \ldots, q_m(\tilde{x}) \\ V\colon v(\tilde{a}) &\leftarrow v_1(\tilde{a}), v_2(\tilde{a}), \ldots, v_n(\tilde{a})\,. \end{aligned} \tag{30.56}$$

Legyen $\tilde{y}$ az $\tilde{x}$-beli változóktól diszjunkt változók listája. Ekkor a

$$Q'\colon q'(\tilde{x}) \leftarrow q_1(\tilde{x}), q_2(\tilde{x}), \ldots, q_m(\tilde{x}), v_1(\tilde{y}), v_2(\tilde{y}), \ldots, v_n(\tilde{y}) \tag{30.57}$$

lekérdezésre teljesül, hogy $Q \equiv Q'$. Világos, hogy $Q' \subseteq Q$. Másfelől, ha valamely r adatbázis példányra létezik az $\tilde{y}$ változóinak olyan kiértékelése, amelyik kielégíti V testét, akkor ezt rögzítve, az $\tilde{x}$-beli változók tetszőleges kiértékelésére pontosan akkor kapunk egy eredménysort Q-ban, amikor a rögzített $\tilde{y}$ kiértékeléssel együtt Q'-ben. ■

A 30.24. állítás és a 30.20. tétel következménye az alábbi tétel.

30.25. tétel. *Legyen Q konjunktív lekérdezés, amelyik tartalmazhat összehasonlítási atomokat, $\mathcal{V}$ nézetek halmaza. Ha a $\mathcal{V}$-beli nézetek összehasonlítási atomokat nem tartalmazó konjunktív lekérdezéssel adottak, akkor NP teljes annak eldöntése, hogy létezik-e Q-nak $\mathcal{V}$-t használó átírása.*

A 30.25. tétel bizonyítását az Olvasóra bízzuk (30.3-1. gyakorlat).

A 30.24. állítás bizonyításában új változókat vezettünk be. Azonban, a következő lemma szerint erre nincs szükség. Másik fontos észrevétel, hogy elegendő az eredeti lekérdezésben szereplő adatbázis relációk egy részhalmazát tekinteni, amikor lokálisan minimális átírást keresünk, új adatbázis relációkat nem kell felhasználni.

30.26. lemma. *Legyen Q konjunktív lekérdezés, amelyben nem szerepelnek összehasonlítási atomok*

$$Q\colon\ q(\tilde{X}) \leftarrow p_1(\tilde{U}_1), p_2(\tilde{U}_2), \ldots, p_n(\tilde{U}_n)\ , \tag{30.58}$$

valamint legyen $\mathcal{V}$ nézetek halmaza, szintén összehasonlítási atomok nélkül.

1. *Ha Q $\mathcal{V}$-t használó lokálisan minimális átírása Q', akkor a Q'-ben szereplő adatbázis literálok halmaza izomorf a Q-beli adatbázis literálok egy részhalmazával.*

2. *Ha*

$$q(\tilde{X}) \leftarrow p_1(\tilde{U}_1), p_2(\tilde{U}_2), \ldots, p_n(\tilde{U}_n), v_1(\tilde{Y}_1), v_2(\tilde{Y}_2), \ldots v_k(\tilde{Y}_k) \tag{30.59}$$

a Q egy a nézeteket használó átírása, akkor létezik egy

$$q(\tilde{X}) \leftarrow p_1(\tilde{U}_1), p_2(\tilde{U}_2), \ldots, p_n(\tilde{U}_n), v_1(\tilde{Y'}_1), v_2(\tilde{Y'}_2), \ldots v_k(\tilde{Y'}_k) \tag{30.60}$$

átírás is, amelyre teljesül, hogy $\{\tilde{Y'}_1 \cup \cdots \cup \tilde{Y'}_k\} \subseteq \{\tilde{U}_1 \cup \cdots \cup \tilde{U}_n\}$, azaz az átírás nem vezet be új változókat.

A 30.26. lemma bizonyításának részleteit az Olvasóra hagyjuk (30.3-2. gyakorlat). A következő lemma alapvető fontosságú: A Q $\mathcal{V}$-t használó minimális átírása nem ***növelheti*** a literálok számát.

30.27. lemma. *Legyen Q konjunktív lekérdezés, $\mathcal{V}$ konjunktív lekérdezésekkel megadott nézetek halmaza, mindkettő összehasonlítási atomok nélkül. Ha Q teste p literált tartalmaz, és Q' a Q $\mathcal{V}$-t használó lokálisan minimális teljes átírása, akkor Q' legfeljebb p literált tartalmaz.*

Bizonyítás. A Q'-beli nézet literálok helyére írjuk be a definíciójukat, így kapjuk a Q'' lekérdezést. Legyen φ homomorfizmus Q testéből Q''-be. φ létezik a Homomorfizmus tétel (30.18. tétel) és $Q \equiv Q''$ alapján. A Q testében szereplő $l_1, l_2, \ldots, l_p$ literálok mindegyike legfeljebb egy nézet literál kifejtéséből kapott tagra képződik. Ha Q'-ben több, mint p nézet literál szerepel, akkor Q'' testében néhány nézet literál kifejtése diszjunkt φ képétől. Ezek a nézet literálok elhagyhatók Q'-ből, úgy, hogy az ekvivalencia nem változik. ■

A 30.27. lemma alapján a következő tétel mondható ki minimális átírások bonyolultságáról.

30.28. tétel. *Legyen Q konjunktív lekérdezés, $\mathcal{V}$ konjunktív lekérdezésekkel megadott nézetek halmaza, mindkettő összehasonlítási atomok nélkül. Tegyük fel, hogy Q testében p literál szerepel.*

1. *Annak eldöntése, hogy létezik-e Q-nak $\mathcal{V}$-t használó Q' átírása, amelyik legfeljebb k ($\leq p$) literált használ, NP-teljes.*
2. *Annak eldöntése, hogy létezik-e Q-nak $\mathcal{V}$-t használó Q' átírása, amelyik legfeljebb k ($\leq p$) adatbázis literált használ, NP-teljes.*
3. *Annak eldöntése, hogy létezik-e Q-nak $\mathcal{V}$-t használó teljes átírása, NP-teljes.*

Bizonyítás. Az első állítást bizonyítjuk, a másik kettő bizonyítása hasonló. A 30.27. és 30.26. lemmák alapján csak olyan átírásokat kell tekintenünk, amelyekben legfeljebb annyi literál szerepel, mint a lekérdezésben, a lekérdezés literáljainak egy részhalmazát tartalmazza, és nem használ új változókat. Egy ilyen átírást valamint az ekvivalenciát bizonyító homomorfizmusokat polinomiális időben tudunk ellenőrizni, tehát a feladat NP-beli. Az NP-nehézség bizonyításához a 30.25. tételt használjuk. Adott Q lekérdezéshez és V nézethez legyen V' az a nézet, amelyiknek a feje megegyezik V fejével, a teste pedig Q és V testének konjunkciója. Könnyen látható, hogy pontosan akkor létezik V'-t használó átírás egyetlen literállal, amikor létezik V-t használó (korlátozások nélküli) átírás. ■

30.3.3. Gyakorlati algoritmusok

Ebben a fejezetben csak teljes átírásokkal foglalkozunk. Ez nem jelent igazi korlátozást, mert ha adatbázis literálokat is szeretnénk használni, akkor bevezethetünk olyan nézeteket, amelyek egy az egyben tükrözik az adatbázis relációkat. A 30.22. definícióban bevezetett ***ekvivalens*** átírás fogalma megfelelő, ha az átírás célja lekérdezés optimalizálás, illetve a fizikai adatfüggetlenség biztosítása. Azonban, adategyesítési, illetve adattárház környezetben nem törekedhetünk arra, hogy az átírás ekvivalens legyen, mert nem feltétlenül áll rendelkezésre minden adat. Ezért bevezetjük a maximálisan tartalmazott átírás fogalmát, amelyik függ attól, melyik lekérdezési nyelvet használjuk, ellentétben az ekvivalens átírásokkal.

30.29. definíció. *Legyen Q lekérdezés, $\mathcal{V}$ nézetek halmaza, $\mathcal{L}$ pedig lekérdezési nyelv. Q-nak $\mathcal{V}$-t használó $\mathcal{L}$-re vonatkozó **maximálisan tartalmazott átírása** Q', ha*

1. *Q' az $\mathcal{L}$ nyelv olyan lekérdezése, amelyik csak a $\mathcal{V}$-beli nézeteket használja,*
2. *Q tartalmazza Q'-t,*
3. *ha $Q_1 \in \mathcal{L}$ lekérdezésre teljesül, hogy $Q' \sqsubseteq Q_1 \sqsubseteq Q$, akkor $Q' \equiv Q_1$.*

Lekérdezés optimalizálás materializált nézetek használatával

Mielőtt rátérünk arra, hogyan lehet egy hagyományos optimalizáló eljárást módosítani, hogy adatbázis relációk helyett materializált nézetekkel dolgozzon, át kell tekintenünk, mikor használható egy nézet valamely lekérdezés megválaszolásához. Lényegileg a V nézet

használható a Q lekérdezéshez, ha a V definíciójában szereplő adatbázis relációk és a Q-ban szereplő relációk halmazainak közös része nem üres, és V olyan attribútumokat is kiválaszt, amelyeket Q is. Ezen kívül, ekvivalens átírás esetén, ha V-ben vannak összehasonlítási atomok olyan attribútumokra, amelyek Q-ban is szereplenek, akkor a nézet logikailag ekvivalens, vagy gyengébb összehasonlítási feltételt kell alkalmazzon, mint a lekérdezés. Ha logikailag erősebb feltételt alkalmaz, akkor a nézet egy (maximálisan) tartalmazott átírás része lehet. Ezt legegyszerűbben egy példán keresztül világíthatjuk meg. Tekintsük a Q lekérdezést az **Egyetem** séma felett, amelyik az olyan tanár, diák, félév hármasokat sorolja fel, ahol a tanár a diák témavezetője és az adott félévben a diák a tanár valamelyik óráját felvette.

$$Q\colon\ q(Tnév,Diák,Félév) \leftarrow Felvett(Diák,K\text{-}szám,Félév),\ Témavezető(Tnév,Diák),\\ Tanít(Tnév,\ K\text{-}szám,Félév,x_V),\ Félév \geq \text{“2000ősz”}\ . \tag{30.61}$$

Az alábbi V_1 nézet használható Q megválaszolásánál, mivel ugyanazt az összekapcsolási feltételt használja a *Felvett* és *Tanít* relációkra, mint Q, amint azt az azonos nevű változók mutatják. Ezen kívül, V_1 kiválasztja a *Diák, Tnév, Félév* attribútumokat, ami szükséges ahhoz, hogy a *Témavezető* relációval megfelelően összekapcsolhassuk, és a végeredménybe kiválaszthassuk a három attribútumot. Végül, a $Félév >$ “1999ősz” összehasonlítási atom logikailag gyengébb feltétel, mint a Q-ban szereplő $Félév \geq$ “2000ősz”.

$$V_1\colon\ v_1(Diák,Tnév,Félév) \leftarrow Tanít(Tnév,K\text{-}szám,Félév,x_V),\\ Felvett(Diák,K\text{-}szám,Félév),\ Félév > \text{“1999ősz”}\ . \tag{30.62}$$

A következő négy nézet mutatja, hogy V_1-t csak kicsit módosítva hogyan változik a felhasználhatóság.

$$V_2\colon\ v_2(Diák,Félév) \leftarrow Tanít(x_T,K\text{-}szám,Félév,x_V),\\ Felvett(Diák,K\text{-}szám,Félév),\ Félév > \text{“1999ősz”}\ . \tag{30.63}$$

$$V_3\colon\ v_3(Diák,Tnév,Félév) \leftarrow Tanít(Tnév,\ K\text{-}szám,x_F,x_V),\\ Felvett(Diák,K\text{-}szám,Félév),\ Félév > \text{“1999ősz”}\ . \tag{30.64}$$

$$V_4\colon\ v_4(Diák,Tnév,Félév) \leftarrow Tanít(Tnév,\ K\text{-}szám,Félév,x_V),\\ Témavezető(Tnév,x_D),\ Tanár(Tnév,x_{Sz}),\\ Felvett(Diák,K\text{-}szám,Félév),\ Félév > \text{“1999ősz”}\ . \tag{30.65}$$

$$V_5\colon\ v_5(Diák,Tnév,Félév) \leftarrow Tanít(Tnév,\ K\text{-}szám,Félév,x_V),\\ Felvett(Diák,K\text{-}szám,Félév),\ Félév > \text{“2001ősz”}\ . \tag{30.66}$$

A V_2 nézet majdnem ugyanaz, mint V_1, az egyetlen különbség, hogy nem választja ki a *Tnév* attribútumot a *Tanít* relációból. Az azonban szükséges a *Témavezető* relációval való természetes összekapcsoláshoz, valamint Q eredményében is szerepel. Így, ha használni akarjuk V_2-t az átíráshoz, akkor újra össze kell kapcsolni a *Tanít* relációval. Azonban, ha a *Felvett* és a *Tanít* relációk természetes összekapcsolása már csak kevés sort tartalmaz az eredeti relációk sorszámához képest, akkor megérheti az újabb összekapcsolás.

A V_3 nézetben a *Felvett* és a *Tanít* relációk összekapcsolása csak a *K-szám* attribútum szerint történik, a *Félév* és a x_F változók egyenlőségét nem követeli meg. Mivel az x_F attribútumot V_3 nem választja ki, ezért nem lehet később összekapcsolási feltételben alkalmazni, így V_3 használata nem jelent nyereséget.

A V_4 nézet csak olyan tanárokat vesz figyelembe, akiknek van legalább egy szakterületük. Ezzel több feltételt alkalmaz, mint az eredeti Q lekérdezés, tehát nem alkalmazható ekvivalens átírásra, ha nem engedjük meg egyesítés és tagadás használatát is a lekérdezési nyelvben. Azonban, ha az adatbázisban van olyan integritási feltétel, hogy minden tanárnak van legalább egy szakterülete, akkor a lekérdezés optimalizáló észre kell vegye, hogy V_4 használható.

Végül, V_5 összehasonlítási operátora erősebb feltételt használ, mint a Q-ban szereplő, így ekvivalens átíráshoz nem, csak (maximálisan) tartalmazott átíráshoz használható.

System-R stílusú optimalizálás

Mielőtt a hagyományos optimizálás változtatásait tárgyalnánk, röviden összefoglaljuk hogyan dolgozik a ***System-R stílusú optimizáló.*** A legjobb végrehajtási sorrendet alulról felfelé építkezve keresi meg. Első fázisban 1 méretű rész-lekérdezéseket tekint, azaz a lekérdezésben szereplő minden egyes relációs táblához megkeresi a legjobb elérési utat. Az n-edik fázisban N méretű végrehajtási terveket tekint, amelyeket kisebbek (k és $n - k$ méretűek) kombinációjával kap. Az eljárás akkor fejeződik be, ha olyan tervet talál, amelyik a lekérdezés összes relációját lefedi. Az eljárás hatékonysága abból adódik, hogy a végrehajtási terveket ***ekvivalencia osztályokra*** bontja, és minden osztályból csak egyetlen tervet tekint. Két terv ugyanabban az osztályban van, ha

- a lekérdezésben szereplő relációk közül ugyanazokat fedik le (tehát ugyanaz a méretük), valamint
- a választ ugyanabban a (lekérdezéstől függő) érdekes sorrendben adják.

A mi esetünkben az optimizáló a lekérdezés végrehajtási tervét nem adatbázis relációkra, hanem nézetekre alapozza. Ezért a szokásosan rendelkezésre álló meta-adatokon kívül (pl. statisztikák, indexek) az optimalizáló rendelkezésére állnak a nézeteket definiáló lekérdezés kifejezések is. Az alábbi változtatásokra van szükség.

A. A lekérdezés megválaszolásához használható nézeteket ki kell választani, a fentiekben vázolt feltételek alapján. A hagyományos optimizáló esetében ez triviális, hiszen egy adatbázis reláció pontosan akkor használható a lekérdezés megválaszolásához, ha szerepel a lekérdezésben.

B. Mivel a lekérdezés végrehajtás terve nézetek összekapcsolását jelenti, nem pedig adatbázis relációkét, a tervek nem oszthatók szépen ekvivalencia osztályokba, mint a hagyományos esetben, és így nem lehet őket méret szerint növekvő sorrendben végignézni. Ezért a következő módosítások szükségesek.

1. ***Megállási feltétel:*** Az eljárás megkülönbözteti a *részleges lekérdezés végrehajtási terveket* a *teljes lekérdezés végrehajtási tervektől.* A lehetséges összekapcsolási sorrendek végigtekintése akkor fejeződik be, ha már nincs több ellenőrizetlen részleges lekérdezés végrehajtási terv. Ezzel szemben a hagyományos optimizáló eljárás akkor ér véget, ha áttekintette azon ekvivalencia osztályokat, amelyek a lekérdezés összes relációját tartalmazzák.

2. ***Végrehajtási tervek elhagyása:*** A hagyományos optimizáló eljárás az *egy ekvivalencia osztályba* tartozó terveket hasonlítja össze páronként, és csak a legolcsóbbat tárolja el minden osztályból. A mi esetünkben *tetszőleges* két eddig előállított tervet hasonlít össze. A p tervet elhagyja, ha létezik olyan p' terv, amelyikre igaz, hogy
   (a) p' olcsóbb, mint p, és
   (b) p' legalább annyit hozzájárul a lekérdezés megválaszolásához, mint p. Ez lényegileg azt jelenti, hogy legalább annyi adatbázis relációt lefed, mint p, és legalább annyi szükséges attribútumot ki is választ.

3. ***Részleges tervek társítása:*** A hagyományos esetben, ha két részleges tervet társítunk egy nagyobb tervvé, akkor a hozzájuk tartozó összekapcsolási feltétel egyértelműen adott a lekérdezésben, az optimalizáló eljárás csak a leghatékonyabb megvalósítást kell megtalálja. A mi esetünkben azonban *a priori* – előzetesen – egyáltalán nem világos, hogy milyen összekapcsolási feltétel eredményez ekvivalens átírást. Tehát az optimalizáló eljárás több, különböző összekapcsolási feltételt kell megvizsgáljon. Szerencsére, a gyakorlatban a rendelkezésre álló meta-adatok lényegesen szűkítik a vizsgálandó feltételek körét. Például, nincs túl sok értelme megpróbálni összekapcsolni egy szöveg típusú attribútumot egy numerikus attribútummal. Hasonlóan az integritási feltételek is csökkenthetik a számba vehető összekapcsolások számát. Miután az összes lehetséges összekapcsolást végigvizsgálta, az optimalizáló azt is ellenőrzi, hogy a kapott terv még mindig a lekérdezés részleges megoldása-e.

A fentieket az alábbi összehasonlító táblázatban foglalhatjuk össze.

Hagyományos optimalizáló	**Nézeteket használó optimalizáló**
1. Fázis	***1. Fázis***
a) Keressük meg az összes lehetséges elérési utat.	a1) Keressük meg az összes nézetet, amelyik használható a lekérdezés megválaszolásához a2) Különböztessük meg a részleges és teljes terveket.
b) Hasonlítsuk össze a költségüket és tartsuk meg a legolcsóbbat.	b) Hasonlítsuk össze páronként a nézeteket. Ha valamelyik nem járul többel hozzá a lekérdezés megválaszolásához mint egy másik, és nem is olcsóbb annál, akkor hagyjuk el.
c) Ha a lekérdezésben egy reláció szerepel, **stop**.	c) Ha nincs részleges megvalósítási terv, **stop**.
2. Fázis	***2. Fázis***
Tekintsük az előző fázisban talált elérési utak összekapcsolásait, amelyek a lekérdezésnek megfelelőek, az összes lehetséges összekapcsolási eljárással.	a1) Tekintsük az előző fázisban talált részleges megoldások összekapcsolásait minden lehetséges összekapcsolási eljárással, minden lehetséges összekapcsolási feltételt alkalmazva. a2) Különböztessük meg a részleges és teljes terveket.
b) Hasonlítsuk össze a kapott összekapcsolási tervek költségét, és tartsuk meg a legolcsóbbat.	b) Ha valamelyik újonnan előállított megoldás nem használható a lekérdezés megválaszolásához, vagy valamelyik másik minden tekintetben jobb nála, akkor hagyjuk el.
c) Ha a lekérdezésben két reláció szerepel, **stop**.	c) Ha nincs részleges megvalósítási terv, **stop**.
3. Fázis	***3. Fázis***
⋮	⋮

Ekvivalens átírások másik módozata az átalakítási szabályok alkalmazása. A közös gondolat, hogy a hagyományos optimizáló átalakítási szabályaihoz hozzáveszik azt, hogy a lekérdezés valamely részét helyettesíteni lehet egy nézettel. Ezekkel részletesebben nem foglalkozunk.

A fentiekben tárgyalt optimalizáló eljárások elsősorban olyan helyzetekre készültek, amikor a szereplő nézetek száma nem nagy, legalábbis összehasonlítható az adatbázis relációk számával. Ezzel ellentétben az adategyesítés környezetben nagyszámú nézet kezelésére kell felkészülnünk, mivel minden egyes adatforrás egy-egy újabb nézetet jelent. Ezenkívül a nézetek bonyolult predikátumokat tartalmazhatnak, hiszen a céljuk, hogy az egyes adatforrások közti finom különbségeket leírják. További különbség, hogy az adategyesítési környezetben a nézetekről általában feltesszük, hogy nem teljesek, azaz nem tartalmazzák a definíciójukat kielégítő összes sort, csak azok egy részhalmazát. A továbbiakban ismertetünk néhány, az adategyesítés céljára kifejlesztett eljárást.

Vödör algoritmus

A vödör algoritmus célja, hogy a felhasználó közvetített sémában megfogalmazott lekérdezését átfogalmazza olyan lekérdezésre, amelyik közvetlenül a rendelkezésre álló adatforrásokra hivatkozik. Feltesszük, hogy konjunktív lekérdezésekről van szó, melyekben lehetnek összehasonlítási atomok. A Q lekérdezés összehasonlítási atomjainak halmazát $C(Q)$-val jelöljük.

Mivel a lehetséges átírások száma exponenciális a lekérdezés méretében, ezért a vödör algoritmus fő gondolata, hogy a lehetséges átírások számát drasztikusan csökkenthetjük, ha a lekérdezés ***részcéljait*** – a benne szereplő relációs atomokat – egyenként tekintjük és meghatározzuk, mely nézetek használhatók a részcélokhoz külön-külön.

Az eljárás általános menete a következő. Először minden részcélhoz egy ***vödröt*** rendelünk, amelyik azon nézeteket tartalmazza, ahonnan a részcél sorait vehetjük. A második lépésben az összes olyan összekapcsolást tekintjük, amelyik minden vödörből tartalmaz egy nézetet, és ellenőrizzük, hogy az így kapott V konjunktív lekérdezés átírás szemantikusan helyes-e, azaz $V \sqsubseteq Q$ teljesül-e, vagy szemantikusan helyessé tehető-e összehasonlítási atomok hozzáadásával. Végül a megmaradó terveket minimalizáljuk a redundáns részcélok elhagyásával. Az alábbi Vödör-készítő eljárás az első lépést hajtja végre. A bemenet adatforrások leírásának $\mathcal{V}$ halmaza, valamint Q konjunktív lekérdezés,

$$Q\colon\ Q(\tilde{X}) \ \leftarrow\ R_1(\tilde{X}_1), R_2(\tilde{X}_2), \ldots, R_m(\tilde{X}_m), C(Q) \tag{30.67}$$

formában.

Vödör-készítő(Q,$\mathcal{V}$)

```
1  for i ← 1 to m
2      do Vödör[i] ← ∅
3         for minden V ∈ 𝒱
4                                          ▷ V: V(Ỹ) ← S_1(Ỹ_1),...S_n(Ỹ_n), C(V) formájú.
5             do for j ← 1 to n
6                if R_i = S_j
7                  then Legyen φ a V változóin következőképpen definiált leképezés:
8                       if Ỹ_j k-adik változója y és y ∈ Ỹ
9                         then φ(y) = x_k, ahol x_k az X̃_i k-adik változója
10                        else φ(y) egy új változó, ami nem szerepel sem Q-ban sem V-ben.
11                      Q'() ← R_1(X̃_1), R_m(X̃_m), C(Q), S_1(φ(Ỹ_1)), ..., S_n(φ(Ỹ_n)), φ(C(V))
12                      if Kielégíthető^≥(Q')
13                        then adjuk φ(V)-t Vödör[i]-hez.
14 return Vödör
```

A Kielégíthető$^{\geq}$ eljárás a 30.1.2. pontban leírt Kielégíthető eljárás kiterjesztése arra az esetre, ha egyenlőség atomok mellett egyenlőtlenség atomok is szerepelhetnek a szabály testében. A szükséges változtatás annyi, hogy minden olyan y változóra, amelyik egyenlőtlenség atomban szerepel, ellenőrizni kell, hogy az y-ra kirótt egyenlőtlenségek egyszerre teljesíthetőek-e.

A Vödör-készítő eljárás polinomiális lépésszámú Q és $\mathcal{V}$ méretének függvényében. Valóban, a 3. és 5. sorok egymásba ágyazott ciklusának magja $n\sum_{V\in\mathcal{V}}|V|$-szer fut le. A 6–13. sorok utasításai a 12. sor kivételével konstans sok lépést jelentenek. A 12. sor **if**

utasításának feltételét polinomiális időben lehet ellenőrizni.

A VÖDÖR-KÉSZÍTŐ eljárás helyességének igazolásához nézzük meg, hogy milyen feltételekkel teszi be a V nézetet $Vödör_i$-be. A 6. sorban ellenőrzi, hogy V-ben szerepel-e részcélként az R_i reláció. Ha nem, akkor nyilván V nem adhat használható információt a Q-beli R_i részcélhoz. Ha V-ben szerepel részcélként az R_i reláció, akkor a 9–10. sorokban elkészíti azt a megfeleltetést, amelyet a változókra alkalmazva az S_j és R_i részcélok megfeleltethetők egymásnak a Q, illetve V fejében levő relációkkal összhangban. Végül a 12. sor ellenőrzi, hogy az így kapott változó megfeleltetésekkel az összehasonlítási atomok nem mondanak-e ellent.

A második lépésben, miután a vödröket a VÖDÖR-KÉSZÍTŐ eljárással elkészítette, a vödör algoritmus ***konjunktív lekérdezés átírások*** halmazát állítja elő. Minden átírás olyan konjunktív lekérdezés, amelyik minden vödörből tartalmaz pontosan egy tényezőt. Az algoritmus eredménye ezen konjunktív lekérdezés átírások egyesítése, hiszen a különböző átírások különböző sorokat adhatnak az eredményhez. Adott Q' konjunktív lekérdezés ***konjunktív lekérdezés átírás,*** ha

1. $Q' \sqsubseteq Q$, vagy
2. Q' kiegészíthető összehasonlítási atomokkal úgy, hogy az előző teljesüljön.

30.10. példa. *Vödör algoritmus.* Tekintsük a következő Q lekérdezést, amelyik azokat az x cikkeket listázza, amely cikkekhez létezik y cikk ugyanabban a témában, hogy x és y kölcsönösen hivatkoznak egymásra. Rendelkezésre áll három nézet V_1, V_2, V_3.

$$\begin{array}{lcl} Q(x) & \leftarrow & idéz(x,y), idéz(y,x), uaTéma(x,y) \\ V_1(a) & \leftarrow & idéz(a,b), idéz(b,a) \\ V_2(c,d) & \leftarrow & uaTéma(c,d) \\ V_3(f,h) & \leftarrow & idéz(f,g), idéz(g,h), uaTéma(f,g)\ . \end{array} \tag{30.68}$$

Első lépésben a VÖDÖR-KÉSZÍTŐ eljárással az alábbi vödröket állítjuk elő.

$idéz(x,y)$	$idéz(y,x)$	$uaTéma(x,y)$
$V_1(x)$	$V_1(x)$	$V_2(x)$
$V_3(x)$	$V_3(x)$	$V_3(x)$

(30.69)

A második lépésben a vödrök direkt szorzatának minden eleméből szerkeszt az algoritmus egy Q' konjunktív lekérdezést, és ellenőrzi, hogy Q tartalmazza-e Q'-t. Ha igen, akkor hozzáadja a válaszhoz.

Esetünkben megpróbálja V_1-t a többi nézettel összerakni, de így nem kap helyes eredményt. Ennek oka, hogy b nem szerepel V_1 fejében, így a Q-ban szereplő összekapcsolási feltételt – az x és y változók szerepelnek *uaTéma* relációban is – nem tudja alkalmazni. Ezek után a V_3-t tartalmazó átírásokat tekinti, és észreveszi, hogy a V_3 fejében található változókat egyenlővé téve tartalmazott átírást kap. Végül, az algoritmus azt is megtalálja, hogy V_3-t és V_2-t kombinálva is átírást kap. Egyszerű további ellenőrzéssel kapjuk, hogy ez utóbbi átírás redundáns, V_2-t el lehet hagyni belőle. Tehát a vödör algoritmus eredménye a (30.68) lekérdezésre és nézetekre a (ténylegesen ekvivalens)

$$Q'(x) \leftarrow V_3(x,x)\ . \tag{30.70}$$

A vödör algoritmus előnye, hogy jelentősen lecsökkenti az ellenőrizendő konjunktív átírás jelöltek számát. Ha az adatforrások alapvetően az összehasonlítási atomokban különböznek egymástól, akkor várhatóan a vödrök mérete kicsi lesz.

A vödör algoritmus fő hátránya pont abban rejlik, amiben az előnye is. Semmilyen becslésünk nincs arra, hogy a vödrök direkt szorzatának a mérete mekkora lesz, lehet, hogy nagy. Továbbá az eljárás minden egyes lehetséges átírásra elvégez egy lekérdezés tartalmazás ellenőrzést, ami már akkor is NP-teljes, ha nincsenek összehasonlítási atomok.

Inverz szabályok

A vödör algoritmusnál általánosabban használható eljárás az ***inverz szabályok*** alkalmazása. Tetszőleges, negáció nélküli, de rekurziót megengedő datalog programmal adott lekérdezéshez megtalálja a maximálisan tartalmazott átírást polinomiális időben.

Az első kérdés az, hogy adott $\mathcal{P}$ datalog program és $\mathcal{V}$ konjunktív nézetek halmaza esetén létezik-e olyan $\mathcal{P}$-vel ekvivalens $\mathcal{P}_v$ datalog program, amelynek *edb* relációi a $\mathcal{V}$-beli $v_1, v_2, \ldots, v_n$ relációk. Sajnos azonban ez a kérdés algoritmikusan eldönthetetlen. Meglepő viszont az, hogy el tudjuk készíteni a lehető legjobb, maximálisan tartalmazott átírást. Abban az esetben, ha létezik $\mathcal{P}$-vel ekvivalens $\mathcal{P}_v$ datalog program, akkor az eljárásunk azt fogja előállítani, hiszen a maximálisan tartalmazott átírás tartalmazza $\mathcal{P}_v$-t is. Ez csak látszólagos ellentmondás avval az állítással, hogy az ekvivalens átírás algoritmikusan eldönthetetlen, hiszen az inverz szabályokkal előállított maximálisan tartalmazott átírásról nem tudjuk eldönteni, hogy ténylegesen ekvivalens-e.

30.11. példa. *Ekvivalens átírás.* Tekintsük a következő $\mathcal{P}$ datalog programot, ahol *él* és $fekete$ relációk *edb* relációk, egy G gráf éleit, illetve feketére színezett csúcsait tartalmazzák:

$$\begin{array}{lrcl} \mathcal{P}: & q(X,Y) & \leftarrow & él(X,Z), él(Z,Y), fekete(Z) \\ & q(X,Y) & \leftarrow & él(X,Z), fekete(Z), q(Z,Y)\ . \end{array} \tag{30.71}$$

Könnyen ellenőrizhető, hogy $\mathcal{P}$ a G gráf olyan útjainak (pontosabban sétáinak) végpontjait adja meg, amelyek minden belső pontja fekete. Tegyük fel, hogy csak az alábbi két nézet érhető el.

$$\begin{array}{lcl} v_1(X,Y) & \leftarrow & él(X,Y), fekete(X) \\ v_2(X,Y) & \leftarrow & él(X,Y), fekete(Y) \end{array} \tag{30.72}$$

v_1 a fekete kezdőpontú, v_2 a fekete végpontú éleket tárolja. Ekkor a $\mathcal{P}$ datalog programnak létezik ekvivalens $\mathcal{P}_v$ átírása, amelyik csak a v_1 és v_2 nézeteket használja *edb* relációként:

$$\begin{array}{lrcl} \mathcal{P}_v: & q(X,Y) & \leftarrow & v_2(X,Z), v_1(Z,Y) \\ & q(X,Y) & \leftarrow & v_2(X,Z), q(Z,Y) \end{array} \tag{30.73}$$

Azonban, ha csak a v_1, vagy v_2 nézet érhető el, akkor nem lehetséges az ekvivalens átírás, mert csak olyan utakat kaphatunk, amelyeknek a kezdő, illetve végpontja fekete.

Az inverz szabály eljárás leírásához szükségünk lesz a ***datalog program,*** illetve a ***datalog szabály*** általánosítására, a ***Horn-szabályra.*** Ha a 30.11. definícióban szereplő (30.27) szabály u_i szabad soraiban a változók és konstansok mellett még ***függvény szimbólumokat*** is megengedünk, akkor ***Horn-szabályról*** beszélünk. Horn-szabályok halmazát ***logikai programnak*** nevezzük. Ebben az értelemben egy függvény szimbólum mentes logikai program lesz datalog program. A 30.11. definíció *edb*, *idb* fogalma logikai programra ugyanúgy értelmezhető.

Az inverz szabály eljárás két lépésből áll. Először olyan logikai programot készítünk, amelyik tartalmazhat függvény szimbólumokat. Azonban ezek a függvény szimbólumok nem szerepelnek rekurzív szabályokban, így a második lépében a logikai programot datalog programmá lehet alakítani.

30.30. definíció. *A*

$$v(X_1,\ldots,X_m) \leftarrow v_1(\tilde{Y}_1),\ldots,v_n(\tilde{Y}_n) \tag{30.74}$$

szabállyal meghatározott v nézet v^{-1} ***inverze*** *Horn szabályok következő halmaza. Minden $v_i(\tilde{Y}_i)$ részcélnak megfelel egy szabály, amelyiknek teste a $v(X_1,\ldots,X_m)$ literál. A szabály feje $v_i(\tilde{Z}_i)$, ahol a $\tilde{Z}_i$-t $\tilde{Y}_i$-ból úgy kapjuk, hogy a (30.74) szabály fejében szereplő változókat meghagyjuk, ezen kívül minden, a fejben nem szereplő Y változó helyére pedig az $f_Y(X_1,\ldots,X_m)$ függvény szimbólumot írjuk. Különböző változókhoz különböző függvény szimbólumok tartoznak. A $\mathcal{V}$ nézet halmaz $\mathcal{V}^{-1}$ inverze a $\{v^{-1}: v \in \mathcal{V}\}$ halmaz, ahol a különböző nézetek inverzeiben különböző függvény szimbólumok szereplenek.*

Az inverz definíció gondolata, hogy ha a v nézetben megjelenik a $(x_1,\ldots x_m)$ sor valamilyen $x_1,\ldots x_m$ konstansokkal, akkor minden a fejben nem szereplő y változónak van valamilyen kiértékelése, ami a szabály testét igazzá teszi. Ezt az „ismeretlen” kiértékelést jelöljük az $f_Y(X_1,\ldots,X_m)$ szimbólummal.

30.12. példa. *Nézetek inverze.* Legyen $\mathcal{V}$ az alábbi nézetek halmaza.

$$\begin{array}{lcl} v_1(X,Y) & \leftarrow & \acute{e}l(X,Z),\acute{e}l(Z,W),\acute{e}l(W,Y) \\ v_2(X)) & \leftarrow & \acute{e}l(X,Z)\ . \end{array} \tag{30.75}$$

Ekkor $\mathcal{V}^{-1}$ a következő Horn szabályokból áll.

$$\begin{array}{lcl} \acute{e}l(X,f_{1,Z}(X,Y)) & \leftarrow & v_1(X,Y) \\ \acute{e}l(f_{1,Z}(X,Y),f_{1,W}(X,Y)) & \leftarrow & v_1(X,Y) \\ \acute{e}l(f_{1,W}(X,Y),Y) & \leftarrow & v_1(X,Y) \\ \acute{e}l(X,f_{2,Z}(X)) & \leftarrow & v_2(X)\ . \end{array} \tag{30.76}$$

Ezek után $\mathcal{P}$ datalog programhoz és konjunktív nézetek $\mathcal{V}$ halmazához könnyű elkészíteni azt a logikai programot, amelyből majd azt a datalog programot kapjuk, ami $\mathcal{P}$ $\mathcal{V}$-t használó maximálisan tartalmazott átírása.

$\mathcal{P}$-ből töröljük az összes olyan szabályt, amelyikben olyan *edb* reláció szerepel, ami nem fordul elő $\mathcal{V}$-beli nézet definíciójában. Az így kapott $\mathcal{P}^-$ programhoz hozzávesszük a $\mathcal{V}^{-1}$ szabályait, és ezáltal nyerjük a $(\mathcal{P}^-,\mathcal{V}^{-1})$ logikai programot. Vegyük észre, hogy $\mathcal{P}$ megmaradt *edb* relációi a $(\mathcal{P}^-,\mathcal{V}^{-1})$ logikai programban *idb* relációk, mivel a $\mathcal{V}^{-1}$ szabályai fejében szerepelnek. Az *idb* relációk elnevezése tetszőleges, így átnevezhetjük őket, hogy ne egyezzen a nevük $\mathcal{P}$ *edb* relációinak nevével. Ezt azonban itt a könnyebb érthetőség kedvéért nem tesszük meg.

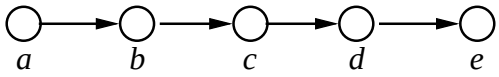
30.13. példa. *Logikai program.* Tekintsük az alábbi datalog programot, ami az *él* reláció tranzitív lezártját számítja ki.

$$\begin{array}{lcl} \mathcal{P}: \quad q(X,Y) & \leftarrow & \acute{e}l(X,Y) \\ q(X,Y) & \leftarrow & \acute{e}l(X,Z),q(Z,Y) \end{array} \tag{30.77}$$

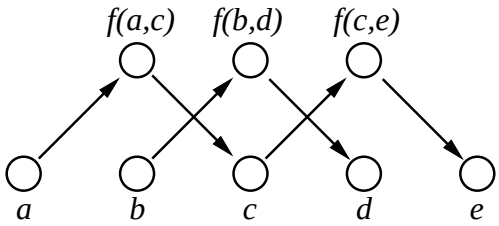
Tegyük fel, hogy csak a

$$v(X,Y) \leftarrow \acute{e}l(X,Z),\acute{e}l(X,Y) \tag{30.78}$$

materializált nézet érhető el, amelyik a kettő hosszúságú utak végpontjait tárolja. Ha csak ezt a nézetet használhatjuk, akkor a legtöbb amit remélhetünk, hogy a páros hosszúságú utak végpontjait tudjuk előállítani. Mivel $\mathcal{P}$ egyetlen *edb* relációja az *él*, ami szerepel v definíciójában, ezért $(\mathcal{P}^-,\mathcal{V}^{-1})$ logikai



30.4. ábra. A G gráf.



30.5. ábra. A G' gráf.

programot úgy kapjuk, hogy $\mathcal{P}$-hez hozzávesszük $\mathcal{V}^{-1}$ szabályait.

$$
\begin{array}{lllll}
(\mathcal{P}^-, \mathcal{V}^{-1}): & q(X,Y) & \leftarrow & él(X,Y) & \\
& q(X,Y) & \leftarrow & él(X,Z), q(Z,Y) & \\
& él(X, f(X,Y)) & \leftarrow & v(X,Y) & \\
& él(f(X,Y),Y) & \leftarrow & v(X,Y)\ . &
\end{array}
\tag{30.79}
$$

A $\mathcal{P}$ datalog program *él edb* relációjának példánya legyen a 30.4. ábrán látható G gráf. Ekkor $(\mathcal{P}^-, \mathcal{V}^{-1})$ három új konstanst vezet be, melyek $f(a,c), f(b,d)$ és $f(c,e)$. A $\mathcal{V}^{-1}$ program *él idb* relációja a 30.5. ábrán látható G' gráfot adja. $\mathcal{P}^-$ a G' gráf tranzitív lezártját számítja ki. Vegyük észre, hogy azok a párok a tranzitív lezártban, amelyek nem tartalmaznak egyet sem az új konstansokból, pontosan a G-beli páros hosszú utak végpontjai.

A 30.13. példában a $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai program eredményét például a NAIV-DATALOG eljárással számíthatjuk ki. Azonban logikai programokra általában nem igaz, hogy az eljárás véget ér. Tekintsük ugyanis a

$$
\begin{array}{lll}
q(X) & \leftarrow & p(X) \\
q(f(X)) & \leftarrow & q(X)
\end{array}
\tag{30.80}
$$

logikai programot. Ha a p *edb* reláció az a konstanst tartalmazza, akkor a program eredménye az $a, f(a), f(f(a)), f(f(f(a))), \ldots$ végtelen sorozat lesz. Ezzel ellentétben az inverz szabály eljárás által adott $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai program eredménye ***garantáltan*** véges, és a kiszámítási algoritmus véges időben véget ér.

30.31. tétel. *Tetszőleges $\mathcal{P}$ datalog programra, konjunktív nézetek $\mathcal{V}$ halmazára és a nézetek tetszőleges véges példányaira teljesül, hogy a $(\mathcal{P}^-, \mathcal{V}^{-1})$ logikai programnak egyértelmű minimális fixpontja van, valamint a* NAIV-DATALOG *és* FÉLIG-NAIV-DATALOG *eljárások ezt a fixpontot adják eredményül.*

A 30.31. tétel bizonyításának lényege, hogy függvény szimbólumokat csak az inverz szabályok vezetnek be, amelyek azonban nem rekurzívak, így egymásba ágyazott függvény

szimbólumokat tartalmazó tényezők nem keletkeznek. A bizonyítás részletezését az Olvasóra bízzuk (30.3-3. gyakorlat).

Még ha egy adatbázis *edb* relációiból indulunk is ki, egy logikai program eredményében lehetnek olyan sorok, amelyek függvény szimbólumokat tartalmaznak. Ezért bevezetünk egy szűrőt, ami eltávolítja a szükségtelen sorokat. Ha a $\mathcal{P}$ logikai program *edb* relációinak példánya a D adatbázis, akkor $\mathcal{P}(D)\downarrow$ jelöli azon $\mathcal{P}(D)$-beli sorok halmazát, amelyek nem tartalmaznak függvény szimbólumokat. Jelölje $\mathcal{P}\downarrow$ azt a programot, amelyik adott D példányra $\mathcal{P}(D)\downarrow$-t számítja ki. Az alábbi tétel bizonyítása meghaladja jelen fejezet kereteit.

30.32. tétel. *Tetszőleges $\mathcal{P}$ datalog programra, valamint konjunktív nézetek $\mathcal{V}$ halmazára teljesül, hogy $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ logikai program $\mathcal{P}$ $\mathcal{V}$-t használó maximálisan tartalmazott átírása. Továbbá $(\mathcal{P}^-, \mathcal{V}^{-1})$ előállítható $\mathcal{P}$ és $\mathcal{V}$ méretében polinomiális időben.*

A 30.32. tétel jelentése, hogy az az egyszerű eljárás, hogy a nézet definíciók inverzeit hozzáadjuk a datalog programhoz, olyan logikai programot eredményez, ami a lehető legjobban használja fel a nézeteket. Az, hogy $(\mathcal{P}^-, \mathcal{V}^{-1})$ előállítható $\mathcal{P}$ és $\mathcal{V}$ méretében polinomiális időben, könnyen látható, hiszen minden $v_i \in \mathcal{V}$ minden részcéljához egyetlen inverz szabályt kell elkészíteni.

Az átírási feladat teljes megoldásához szükséges azonban egy olyan datalog programot előállítani, amelyik ekvivalens a $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ logikai programmal. Ehhez adja a kulcsot az az észrevétel, hogy $(\mathcal{P}^-, \mathcal{V}^{-1})$-ben csak véges sok függvény szimbólum van, továbbá az alulról felfelé történő kiszámításban, mint a Naiv-datalog eljárás és változatai, egymásba ágyazott függvény szimbólumok nem jönnek létre. Megfelelő könyveléssel nyomon követhetjük a függvény szimbólumok megjelenését, anélkül, hogy ténylegesen előállítanánk azokat tartalmazó sorokat.

Az átalakítást alulról felfelé végezzük, a Naiv-datalog eljáráshoz hasonlóan. $\mathcal{V}^{-1}$-beli *idb* relációban megjelenő $f(X_1, \ldots, X_k)$ függvény szimbólumot a $X_1, \ldots, X_k$ változó listával helyettesítjük. Ugyanakkor az *idb* reláció nevet meg kell jelölni, hogy tudjuk, az $X_1, \ldots, X_k$ lista a $f(X_1, \ldots, X_k)$ függvényhez tartozott. Ezzel új, "ideiglenes" reláció neveket vezetünk be. Tekintsük a 30.13. példa (30.79) logikai programjában szereplő

$$él(X, f(X, Y)) \leftarrow v(X, Y) \tag{30.81}$$

szabályt az

$$él^{\langle 1, f(2,3)\rangle}(X, X, Y) \leftarrow v(X, Y) \tag{30.82}$$

szabállyal helyettesítjük. A $\langle 1, f(2, 3)\rangle$ jelölés értelmezése, hogy $él^{\langle 1, f(2,3)\rangle}$ első argumentuma megegyezik *él* első argumentumával, a második és harmadik argumentum $él^{\langle 1, f(2,3)\rangle}$-ben az f függvény szimbólummal együtt adja az *él* második argumentumát. Ha $(\mathcal{P}^-, \mathcal{V}^{-1})$ alulról felfelé kiszámítása során $\mathcal{P}^-$ valamelyik *idb* relációjának argumentumába függvény szimbólum kerülne, akkor egy új szabályt adunk a programhoz, a megfelelően megjelölt reláció nevekkel.

30.14. példa. *Logikai program átalakítása datalog programmá.* A 30.13. példa logikai programját az alábbi datalog programmá alakítja át a fent vázolt eljárás. A Naiv-datalog alulról felfelé végrehajtásá-

nak különböző fázisait vonalak határolják el.

$$
\begin{array}{lcl}
\acute{e}l^{\langle 1,f(2,3)\rangle}(X,X,Y) & \leftarrow & v(X,Y) \\
\acute{e}l^{\langle f(1,2),3\rangle}(X,Y,Y) & \leftarrow & v(X,Y) \\
\hline
q^{\langle 1,f(2,3)\rangle}(X,Y_1,Y_2) & \leftarrow & \acute{e}l^{\langle 1,f(2,3)\rangle}(X,Y_1,Y_2) \\
q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) & \leftarrow & \acute{e}l^{\langle f(1,2),3\rangle}(X_1,X_2,Y) \\
\hline
q(X,Y) & \leftarrow & \acute{e}l^{\langle 1,f(2,3)\rangle}(X,Z_1,Z_2), q^{\langle f(1,2),3\rangle}(Z_1,Z_2,Y) \\
q^{\langle f(1,2),f(3,4)\rangle}(X_1,X_2,Y_1,Y_2) & \leftarrow & \acute{e}l^{\langle f(1,2),3\rangle}(X_1,X_2,Z), q^{\langle 1,f(2,3)\rangle}(Z,Y_1,Y_2) \\
\hline
q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) & \leftarrow & \acute{e}l^{\langle f(1,2),3\rangle}(X_1,X_2,Z), q(Z,Y) \\
q^{\langle 1,f(2,3)\rangle}(X,Y_1,Y_2) & \leftarrow & \acute{e}l^{\langle 1,f(2,3)\rangle}(X,Z_1,Z_2), q^{\langle f(1,2),f(3,4)\rangle}(Z_1,Z_2,Y_1,Y_2)
\end{array}
\tag{30.83}
$$

Az így kapott datalog program egyértelműen mutatja, hogy melyik argumentumokban keletkezhet függvény szimbólum az eredeti logikai programban. Azonban, bizonyos függvény szimbólumokat tartalmazó sorok sohasem eredményeznek függvény szimbólumokat nem tartalmazó sorokat a program eredményének kiszámítása során.

A p relációt ***fontosnak*** nevezzük, ha a 30.16. definícióban megadott előzmény gráfban[3] p-ből van irányított út a program q eredmény relációjába. Ha p nem fontos, akkor a program eredményének kiszámításához nincs szükség p-beli sorokra, így p elhagyható a programból.

30.15. példa. *Nem fontos relációk elhagyása.* A 30.14. példában kapott datalog program előzmény gráfjában a $q^{\langle 1,f(2,3)\rangle}$ és $q^{\langle f(1,2),f(3,4)\rangle}$ relációkból nincs irányított út a program q eredmény relációjába, ezért nem fontosak, azaz el lehet hagyni őket és a hozzájuk tartozó szabályokat. Az alábbi datalog programot kapjuk ezután:

$$
\begin{array}{lcl}
\acute{e}l^{\langle 1,f(2,3)\rangle}(X,X,Y) & \leftarrow & v(X,Y) \\
\acute{e}l^{\langle f(1,2),3\rangle}(X,Y,Y) & \leftarrow & v(X,Y) \\
q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) & \leftarrow & \acute{e}l^{\langle f(1,2),3\rangle}(X_1,X_2,Y) \\
q^{\langle f(1,2),3\rangle}(X_1,X_2,Y) & \leftarrow & \acute{e}l^{\langle f(1,2),3\rangle}(X_1,X_2,Z), q(Z,Y) \\
q(X,Y) & \leftarrow & \acute{e}l^{\langle 1,f(2,3)\rangle}(X,Z_1,Z_2), q^{\langle f(1,2),3\rangle}(Z_1,Z_2,Y)\ .
\end{array}
\tag{30.84}
$$

Még egy egyszerűsítő lépést hajthatunk végre, ami ugyan nem csökkenti a szükséges levezetések számát az eredmény kiszámítása során, viszont elkerül felesleges adat másolásokat. Ha p olyan reláció egy datalog programban, amelyet egyetlen szabály definiál, és annak az egyetlen szabálynak a testében csak egyetlen reláció áll, akkor p elhagyható a programból: Minden olyan szabályban, amelyiknek a testében p előfordul, p-t helyettesíthetjük a p-t definiáló szabály testében szereplő relációval, a változók megfelelő egyenlővé tétele után.

30.16. példa. *Adatmásolás elkerülése.* A 30.14. példában $\acute{e}l^{\langle 1,f(2,3)\rangle}$ és $\acute{e}l^{\langle f(1,2),3\rangle}$ relációkat egyetlen szabály határozza meg, amely szabályok testében egyetlen reláció szerepel. Ezért a (30.84) program tovább egyszerűsíthető.

$$
\begin{array}{lcl}
q^{\langle f(1,2),3\rangle}(X,Y,Y) & \leftarrow & v(X,Y) \\
q^{\langle f(1,2),3\rangle}(X,Z,Y) & \leftarrow & v(X,Z), q(Z,Y) \\
q(X,Y) & \leftarrow & v(X,Z), q^{\langle f(1,2),3\rangle}(X,Z,Y)\ .
\end{array}
\tag{30.85}
$$

[3] Itt az előzmény gráf definícióját ki kell terjesztenünk a datalog program *edb* relációira is.

A fenti két egyszerűsítés eredményeként kapott datalog programot $(\mathcal{P}^-, \mathcal{V}^{-1})^{datalog}$-gal jelöljük. Világos, hogy $(\mathcal{P}^-, \mathcal{V}^{-1})$ és $(\mathcal{P}^-, \mathcal{V}^{-1})^{datalog}$ alulról felfelé történő kiszámítása közt kölcsönösen egyértelmű megfeleltetés létezik. Mivel a függvény szimbólumokat $(\mathcal{P}^-, \mathcal{V}^{-1})^{datalog}$-ban nyomon követjük, ezért tudjuk, hogy az eredményül kapott példány megegyezik $(\mathcal{P}^-, \mathcal{V}^{-1})$ eredményének függvény szimbólumot nem tartalmazó sorainak részhalmazával.

30.33. tétel. *Tetszőleges $\mathcal{P}$ datalog programra és konjunktív nézetek $\mathcal{V}$ halmazára, a $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ program ekvivalens a $(\mathcal{P}^-, \mathcal{V}^{-1})^{datalog}$ programmal.*

MiniCon

A vödör algoritmus hátránya alapvetően az, hogy a nézetek részcéljai közti kölcsönhatások nagy részét nem figyeli meg azáltal, hogy minden egyes részcélt elkülönítve vizsgál. Így a vödrök sok használhatatlan nézetet tartalmazhatnak, és az algoritmus második fázisa nagyon költségessé válhat.

Az inverz szabály eljárás előnye a fogalmi egyszerűsége és modularitása. A nézetek inverzeit csak egyszer kell kiszámítani, utána már tetszőleges datalog programmal adott lekérdezéshez használhatóak. Azonban, az inverzek használatával elveszíthetjük azt az előnyt, hogy a nézet már kiszámított néhány szükséges összekapcsolást. A lekérdezés megválaszolása során ugyanis az inverz szabály eljárás lényegileg újra előállítja az adatbázis relációkat.

A MiniCon algoritmus az előző kettő hátrányait próbálja kiküszöbölni. A kulcs gondolata, hogy ahelyett, hogy a lekérdezés ***részcéljaihoz*** keresnénk átírásokat, azt vizsgáljuk, hogy a lekérdezés ***változói*** hogyan működnek együtt a rendelkezésre álló nézetekkel. A továbbiakban visszatérünk a konjunktív lekérdezésekhez, és a könnyebb érthetőség kedvéért csak olyan lekérdezéseket és nézeteket tekintünk, amelyek nem tartalmaznak konstansokat.

A MiniCon eljárás úgy kezdődik, mint a vödör algoritmus, azt vizsgálja, mely nézetek tartalmaznak a lekérdezés valamelyik részcéljának megfelelő részcélt. Azonban, amikor az algoritmus talál egy részleges leképezést a lekérdezés g részcéljáról valamelyik V nézet g_1 részcéljára, nézőpontot vált, és a lekérdezés változóit tekinti. Az algoritmus megvizsgálja a lekérdezés összekapcsolási feltételeit – amelyeket a változók többszörös előfordulásai határoznak meg – és megkeresi további részcélok olyan minimális halmazát, amit még a V részcéljaira kell képezni, feltéve, hogy g-t g_1-re képezzük. Részcéloknak ez a halmaza, valamint a leképezési információ együttese lesz a ***MiniCon leírás*** (MCL). A második fázisban az MCL-eket kapcsolja össze a MiniCon algoritmus. Az MCL-ek előállítása szükségtelenné teszi a vödör algoritmus legköltségesebb részének, az átírások és a lekérdezés közti tartalmazás ellenőrzésének végrehajtását, mert az MCL-ek előállítási szabálya biztosítja, hogy az összekapcsolásuk helyes eredményt ad.

Adott $\tau\colon Var(Q) \longrightarrow Var(V)$ leképezés esetén azt mondjuk, hogy a V nézet g_1 részcélja ***fedi*** a Q lekérdezés g részcélját, ha $\tau(g) = g_1$. $Var(Q)$, illetve $Var(V)$ a lekérdezés, valamint a nézet változóinak halmazát jelöli. Ahhoz, hogy belássuk egy átírásról, hogy csupa, a lekérdezés eredményéhez tartozó sort ad, meg kell adnunk egy homomorfizmust a lekérdezésről az átírásra. Egy MCL ezen homomorfizmus egy részletének tekinthető, így a részletek majd könnyen összekapcsolhatók lesznek.

A Q lekérdezés átírása a nézeteket használó konjunktív lekérdezések egyesítése. Ezekben az egyes nézetek fejében szereplő változók közül néhány lehet, hogy azonosítva lett, mint a 30.10. példában kapott (30.70) ekvivalens átírásban. Tehát célszerű bevezetnünk a ***fej homomorfizmus*** fogalmát. A $h\colon Var(V) \longrightarrow Var(V)$ leképezés fej homomorfizmus, ha

identitás a V fejében nem szereplő változókon, de azonosíthat fejben szereplő változókat. Minden x V fejében szereplő változóra $h(x)$ is V fejében szerepel, továbbá $h(x) = h(h(x))$. Ezek után megadhatjuk az MCL pontos definícióját.

30.34. definíció. *A Q lekérdezéshez a V nézet feletti* ***MiniCon leírás (MCL)*** *a $C = (h_C, V(\tilde{Y})_C, \varphi_C, G_C)$ négyes, ahol*

- *h_C fej homomorfizmus V-n,*
- *$V(\tilde{Y})_C$-t a h_C alkalmazásával kapjuk V-ből, azaz $\tilde{Y} = h_C(\tilde{A})$, ahol $\tilde{A}$ a V fejében szereplő változók halmaza,*
- *φ_C részleges leképezés $Var(Q)$-ról $h_C(Var(V))$-be,*
- *G_C a Q olyan részcéljainak egy halmaza, amelyeket valamelyik $H_C(V)$-beli részcél fed, a φ_C leképezéssel.*

Az alábbi állításon alapszik az MCL-eket előállító eljárás.

30.35. állítás. *Legyen C a V nézet feletti MiniCon leírás a Q lekérdezéshez. C csak akkor használható a Q nem redundáns átírásához, ha*

> ***F1.*** *minden olyan x változóra, amelyik Q fejében van és φ_C értelmezési tartományában is, $\varphi_C(x)$ a $h_C(V)$ fejében található, valamint*
>
> ***F2.*** *ha $\varphi_C(y)$ nem szerepel $h_C(V)$ fejében, akkor Q minden olyan g részcéljára, amelyik tartalmazza y-t, teljesül, hogy*
>
> > *1. g minden változója szerepel φ_C értelmezési tartományában, és*
> >
> > *2. $\varphi_C(g) \in h_C(V)$.*

Az **F1.** feltétel lényegileg ugyanaz, mint a vödör algoritmus feltétele arra, mikor kerül bele egy nézet egy vödörbe. **F2.** jelentése, hogy ha az x változó szerepel a lekérdezés valamelyik összekapcsolási feltételében, amelyik feltételt a nézet nem teljesíti, akkor x-nek szerepelnie kell a nézet fejében, hogy az őt tartalmazó összekapcsolási feltétel később alkalmazható legyen valamilyen másik részcéllal az átírás folyamán. A MCL-készítő eljárás Q konjunktív lekérdezéshez és konjunktív nézetek $\mathcal{V}$ halmazához megadja a használható MiniCon leírásokat.

MCL-KÉSZÍTŐ($Q, \mathcal{V}$)

```
1  𝒞 ← ∅
2  for Q minden g részcéljára
3      do for V ∈ 𝒱
4          do for V minden v részcéljára
5              do Legyen h a legáltalánosabb fej homomorfizmus V-n, amelyikre van
                  φ leképezés, hogy φ(g) = h(v).
6                 if φ és h létezik
7                    then Adjuk 𝒞-hez azon C MCL-eket, amelyek megadhatók úgy, hogy:
8                             (a) φ_C (h_C) a φ (h) kiterjesztése,
9                             (b) G_C a Q részcéljainak olyan minimális részhalmaza, melyre
                                  30.35. állítás teljesül, és
10                            (c) φ és h nem terjeszthető ki φ'_C és h'_C-re úgy, hogy (b) teljesül,
                                  és a (b)-ben meghatározott G'_C-re G'_C ⊊ G_C.
11 return 𝒞
```

Tekintsük újra a 30.10. példa (30.68) lekérdezését és nézeteit. Az MCL-KÉSZÍTŐ eljárás először a lekérdezés *idéz*(x, y) részcélját tekinti. Nem állít elő MCL-t a V_1 nézethez, mivel a 30.35. állítás **F2.** feltétele megsérülne. Ugyanis V_1-nek az *uaTéma*(x, y) részcélt is fednie kéne a $\varphi(x) = a$, $\varphi(y) = b$ leképezésnél, hiszen b nincs V_1 fejében.[4] Ugyanezen ok miatt a lekérdezés többi részcéljánál sem készít MCL-t a V_1 nézethez. Valamilyen értelemben a MiniCon eljárás a vödör algoritmus második lépésének bizonyos részeit az MCL-KÉSZÍTŐ eljárásban elvégzi. Az alábbi táblázat mutatja az MCL-KÉSZÍTŐ eredményét.

$V(\tilde{Y})$	h	φ	G
$V_2(c, d)$	$c \to c, d \to d$	$x \to c, y \to d$	3
$V_3(f, f)$	$f \to f, h \to f$	$x \to f, y \to f$	1, 2, 3

(30.86)

Az MCL-KÉSZÍTŐ eljárás minimális olyan G_C részcél halmazt ad meg, ami teljesíti a 30.35 állítás feltételeit. Ez lehetővé teszi, hogy a MiniCon algoritmus második fázisában csak olyan MCL-eket kapcsoljunk össze, amelyek a lekérdezés részcéljainak ***páronként diszjunkt*** részhalmazait fedik.

30.36. állítás. *Adott Q lekérdezésre és nézetek $\mathcal{V}$, valamint MCL-ek $\mathcal{C}$ halmazára csak olyan $C_1, \ldots C_l$ MCL-ek kapcsolhatók össze Q nem redundáns átírásává, amelyekre teljesül, hogy*

F3. *$G_{C_1} \cup \cdots \cup G_{C_l}$ Q összes részcélját tartalmazza, és*
F4. *minden $i \neq j$-re $G_{C_i} \cap G_{C_j} = \emptyset$.*

Az, hogy csak páronként diszjunkt MCL-eket érdemes összekapcsolni, jelentősen lecsökkenti a keresési teret. A MCL-ÖSSZEKAPCSOLÓ eljáráshoz még egy jelölést be kell vezetnünk. A C MCL φ_C leképezése Q változóinak egy egész halmazát képezheti $h_C(V)$ ugyanazon változójára. E halmaz egy tetszőlegesen választott reprezentánsát kiválasztjuk, arra ügyelve, hogy ha a halmazban van Q fejében található változó, akkor egy olyat. $EC_{\varphi_C}(x)$ jelöli annak a halmaznak a reprezentáns változóját, amelyikbe x tartozik. A C MiniCon leírást EC_{φ_C}-vel

[4] $\varphi(x) = b$, $\varphi(y) = a$ eset hasonló.

kiegészítve a $(h_C, V(\tilde{Y}), \varphi_C, G_C, EC_{\varphi_C})$ ötösként kezeljük. Ha a $C_1, \ldots, C_k$ MCL-eket akarjuk összekapcsolni, és valamilyen $i \neq j$-re $EC_{\varphi_{C_i}}(x) = EC_{\varphi_{C_i}}(y)$ és $EC_{\varphi_{C_j}}(y) = EC_{\varphi_{C_j}}(z)$ teljesül, akkor az összekapcsolással kapott konjunktív átírásban x, y és z is ugyanarra a változóra lesz leképezve. Jelölje S_C azt az ekvivalencia relációt Q változóin, amelynek osztályai a φ_C által ugyanarra a változóra képezett elemek, azaz $xS_Cy \iff EC_{\varphi_C}(x) = EC_{\varphi_C}(y)$. $\mathcal{C}$ az MCL-készítő eljárás eredményeként kapott MCL-ek halmaza.

MCL-összekapcsoló($\mathcal{C}$)

```
1  Válasz ← ∅
2  for {C_1, ..., C_n} ⊆ C amelyre G_{C_1}, ..., G_{C_n} a Q részcéljainak partíciója
3      do Definiáljuk a Ψ_i leképezést az Ỹ_i-n a következőképpen:
4          if Q-ban van x változó melyre φ_i(x) = y
5              then Ψ_i(y) = x
6              else Ψ_i(y) az y egy új példánya
7      legyen S az S_{C_1} ∪ ··· ∪ S_{C_n} tranzitív lezártja
8                                          ▷ S ekvivalencia reláció Q változóin.
9      az S minden osztályához jelöljünk ki egy reprezentánst.
10     definiáljuk az EC leképezést a következőképpen:
11     if x ∈ Var(Q)
12         then EC(x) az S x-t tartalmazó osztályának reprezentánsa
13         else EC(x) = x
14     legyen Q' a Q'(EC(X̃)) ← V_{C_1}(EC(Ψ_1(Ỹ_1))), ..., V_{C_n}(EC(Ψ_n(Ỹ_n)))
15     Válasz ← Válasz ∪ {Q'}
16 return Válasz
```

Igaz az alábbi tétel.

30.37. tétel. *Adott konjunktív Q lekérdezésre és konjunktív nézetek $\mathcal{V}$ halmazára a* MiniCon *algoritmus által előállított konjunktív lekérdezések egyesítése a Q $\mathcal{V}$-t használó maximálisan tartalmazott átírása.*

A 30.37. tétel teljes bizonyítása meghaladja e fejezet kereteit. A 30-1. feladatban az Olvasóra bízzuk annak belátását, hogy az MCL-összekapcsoló eljárás eredményeként kapott konjunktív lekérdezések egyesítését Q tartalmazza.

Megjegyezzük, hogy a Vödör algoritmus, az Inverz-szabályok és a MiniCon algoritmus futási ideje legrosszabb esetben megegyező: $O(nmM^n)$, ahol n a lekérdezés részcéljainak száma, m a nézetek részcéljainak maximális száma és M a nézetek száma. Azonban gyakorlati futási eredmények azt mutatják, hogy nagyszámú nézet esetén (3–400 nézet) a MiniCon algoritmus lényegesen gyorsabb, mint a másik kettő.

Gyakorlatok

30.3-1. A 30.24. állítás és a 30.20. tétel felhasználásával bizonyítsuk be a 30.25. tételt.

30.3-2. Bizonyítsuk be a 30.26. lemma két állítását. *Útmutatás.* Az első állításhoz Q'-ben a $v_i(\tilde{Y}_i)$ nézetek helyére írjuk be a definíciójukat. Az így kapott Q'' lekérdezést minimalizáljuk a 30.19. tétel segítségével. A második bizonyítandó állításhoz használjuk a 30.24. állítást, amellyel bizonyítsuk be, hogy létezik h_i homomorfizmus a $v_i(\tilde{Y}_i)$ nézetet definiáló konjunktív leképezés testéből Q testébe. Lássuk be, hogy a $\tilde{Y}'_i = h_i(\tilde{Y}_i)$ választás megfelelő.

30.3-3. Bizonyítsuk be a 30.31. tételt, felhasználva, hogy a datalog programok minimális fixpontja egyértelmű.

Feladatok

30-1. MiniCon helyes
Bizonyítsuk be, hogy a MiniCon algoritmus helyes eredményt ad. *Útmutatás.* Elegendő belátni, hogy bármelyik, az MCL-összekapcsoló eljárás 14. sorában megadott Q' konjunktív lekérdezésre igaz, hogy $Q' \sqsubseteq Q$. Ez utóbbihoz készítsünk homomorfizmust Q-ról Q'-re.

30-2. $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ ***helyes***
Bizonyítsuk be, hogy a $(\mathcal{P}^-, \mathcal{V}^{-1})\downarrow$ logikai program eredményének minden sora benne van $\mathcal{P}$ eredményében. (A 30.32. tétel bizonyításának része.) *Útmutatás.* Legyen t olyan sor $(\mathcal{P}^-, \mathcal{V}^{-1})$ eredményében, amelyik nem tartalmaz függvény szimbólumot. Tekintsük t levezetési fáját. Ennek levelei nézet literálok, hiszen azok a $(\mathcal{P}^-, \mathcal{V}^{-1})$ program extenzionális relációi. Ha ezeket a leveleket elhagyjuk a levezetési fából, a maradék fa levelei már $\mathcal{P}$ *edb* relációi. Mutassuk meg, hogy az így kapott fa t levezetési fája a $\mathcal{P}$ datalog programban.

30-3. Datalog nézet
Ezzel a feladattal azt szeretnénk megindokolni, miért csak konjunktív nézet definíciókat tekintettünk. Legyen $\mathcal{V}$ nézetek halmaza, Q pedig lekérdezés. A nézetek adott $\mathcal{I}$ példánya esetén a t sor a Q lekérdezés ***biztos válasza,*** ha tetszőleges olyan $\mathcal{D}$ adatbázis példányra, amelyikre teljesül, hogy $\mathcal{I} \subseteq \mathcal{V}(\mathcal{D})$, $t \in Q(\mathcal{D})$ fenn áll.

a. Bizonyítsuk be, hogy ha a $\mathcal{V}$-beli nézetek datalog programmal vannak meghatározva, a Q lekérdezés konjunktív, és nem-egyenlőségi ($\neq$) atomokat tartalmazhat, az a kérdés, hogy a nézetek adott $\mathcal{I}$ példánya esetén egy t sor a Q lekérdezés biztos válasza-e, algoritmikusan eldönthetetlen. *Útmutatás.* Vezessük vissza rá a ***Post Megfeleltetési Problémát,*** ami a következő: Adott az $\{a, b\}$ ábécé feletti szavak két, $\{w_1, w_2, \ldots, w_n\}$ és $\{w'_1, w'_2, \ldots, w'_n\}$ halmaza. A kérdés az, hogy létezik-e olyan index sorozat $i_1, i_2, \ldots, i_k$ (ismétlések megengedettek), hogy

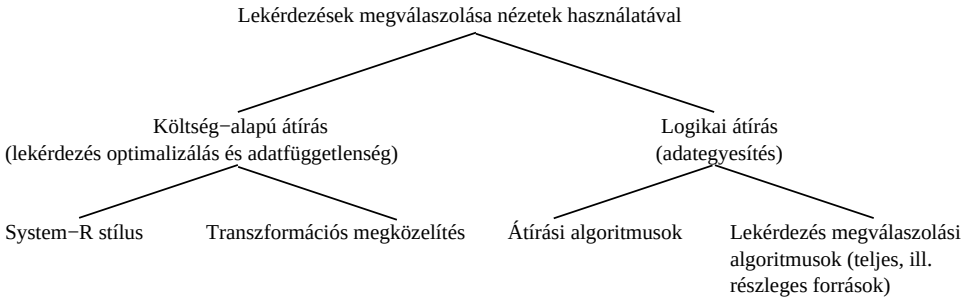
$$w_{i_1} w_{i_2} \cdots w_{i_k} = w'_{i_1} w'_{i_2} \cdots w'_{i_k} \ . \tag{30.87}$$

A Post Megfeleltetési Problémáról ismeretes, hogy algoritmikusan eldönthetetlen. Legyen a V nézet az alábbi datalog programmal adva:

$$\begin{array}{lcl} V(0,0) & \leftarrow & S(e,e,e) \\ V(X,Y) & \leftarrow & V(X_0,Y_0), S(X_0,X_1,\alpha_1), \ldots, S(X_{g-1},Y,\alpha_g), \\ & & S(Y_0,Y_1,\beta_1), \ldots, S(Y_{h-1},Y,\beta_h) \\ & & \text{ahol } w_i = \alpha_1 \ldots \alpha_g \text{ és } w'_i = \beta_1 \ldots \beta_h \\ & & \text{egy szabály minden } i \in \{1,2,\ldots,k\}\text{-ra} \\ S(X,Y,Z) & \leftarrow & P(X,X,Y), P(X,Y,Z) \ . \end{array} \tag{30.88}$$

Legyen továbbá Q a következő konjunktív lekérdezés.

$$Q(c) \ \leftarrow \ P(X,Y,Z), P(X,Y,Z'), Z \neq Z' \ . \tag{30.89}$$



30.6. ábra. Lekérdezés átírás használatának osztályozása.

Lássuk be, hogy a V nézet $\mathcal{I}$ példánya esetén, melyre $\mathcal{I}(V) = \{\langle e, e\rangle\}$ és $\mathcal{I}(S) = \{\}$, a $\langle c\rangle$ sor pontosan akkor lesz Q biztos válasza, ha a $\{w_1, w_2, \ldots, w_n\}$ és $\{w'_1, w'_2, \ldots, w'_n\}$ halmazokra a Post Megfeleltetési Problémának *nincs* megoldása.

b. Az *a.* pontban leírt kiszámíthatatlansági eredménnyel ellentétben, ha $\mathcal{V}$ konjunktív nézetek halmaza, és a Q lekérdezés a $\mathcal{P}$ datalog programmal adott, akkor tetszőleges t sorról könnyen eldönthető, hogy a Q lekérdezés biztos válasza-e adott $\mathcal{I}$ nézet példány esetén. Bizonyítsuk be, hogy a $(\mathcal{P}^-, \mathcal{V}^{-1})^{datalog}$ datalog program pontosan a Q biztos válaszában található sorokat adja eredményül.

Megjegyzések a fejezethez

A „lekérdezések megválaszolása nézetek használatával” feladat kezelését több szempontból lehet osztályozni. A 30.6. ábrán az osztályozás vázlata látható.

A legfontosabb választóvonal a különböző munkák közt, hogy a céljuk adategyesítés, vagy pedig lekérdezés optimalizálás és a fizikai adatfüggetlenség elérése. A két megközelítés közti különbség kulcsa a lekérdezéseket megválaszoló nézeteket használó algoritmus kimenete. Az első esetben, adott Q lekérdezéshez és nézetek $\mathcal{V}$ halmazához az eljárás célja olyan Q' lekérdezés előállítása, amelyik a nézetekre hivatkozik, és ekvivalens Q-val, vagy Q tartalmazza Q'-t. A második esetben az eljárásnak tovább kell lépnie, és egy (remélhetőleg) optimális végrehajtási tervet is elő kell állítania a Q megválaszolására a nézetek (és esetleg adatbázis relációk) használatával. Ebben az esetben csak ekvivalens átírásokat tekinthetünk.

A hasonlóság a két megközelítés között az, hogy mindkettő alapkérdése, hogy egy átírás vajon ekvivalens-e, vagy tartalmazza-e a lekérdezés. Azonban, amíg logikai helyesség elegendő az adategyesítés nézőpontból, addig lekérdezés optimalizálási szempontból nem, ott a *legolcsóbb,* a nézeteket használó végrehajtási tervet kell megtalálni. A nehézségek abból adódnak, hogy az optimalizáló algoritmusok olyan nézeteket is figyelembe kell vegyenek, amelyek ugyan nem járulnak hozzá az átírás logikai helyességéhez, de csökkentik a végrehajtási terv költségét. Ezért az adategyesítési algoritmusok helyességének indoklása főként logikai, míg az optimizálóké logikai és költség-alapú is. Másfelől, adategyesítési problémakörben alapvető adottság a nézetek nagy száma, amelyek a különböző adatforrásoknak

felelnek meg. Ezzel ellentétben, az optimalizálási feladatnál általában (de nem mindig!) feltesszük, hogy a nézetek száma a séma nagyságával összehasonlítható.

A lekérdezés optimalizálás téma munkái tovább oszthatók ***System-R stílusú,*** valamint transzformációs optimizálókra. Az előbbiekhez tartoznak Chaudhuri, Krishnamurty, Potomianos és Shim [62]; Tsatalos, Solomon, és Ioannidis [341] munkái. Az utóbbihoz Florescu, Raschid, és Valduriez [113]; Bello és társai [36]; Deutsch, Popa és Tannen [88], Zaharioudakis és társai [360], valamint Goldstein és Larson[138] cikkei.

Az adategyesítési munkák közül átírási algoritmusokkal foglalkoznak Yang és Larson [353]; Levy, Mendelzon, Sagiv és Srivastava [154]; Qian [292]; valamint Lambrecht, Kambhampati és Gnanaprakasam [219] cikkei. A vödör algoritmust Levy, Rajaraman és Ordille [151] vezette be. Az inverz szabályok eljárás Duschka és Genesereth [97, 98] munkája. A MiniCon algoritmust Pottinger és Halevy fejlesztették ki [286, 285].

Lekérdezés megválaszolási algoritmusokkal, illetve feladat bonyolultságával foglalkozik Abiteboul és Duschka [2]; Grahne és Mendelzon [145]; valamint Calvanese, De Giacomo, Lenzerini és Vardi [60].

A STORED rendszert Deutsch, Fernandez és Suciu dolgozták ki [87]. A szemantikus gyorstárolást Yang, Karlapalem és Li [354] tárgyalja. Az átírási feladat különböző kiterjesztéseivel [55, 114, 121, 216, 354] foglalkoznak.

A témakör összefoglalása található Abiteboul [1], Florescu, Levy és Mendelzon [112], Halevy [152, 153], valamint Ullman[343] munkáiban.

A fejezet témakörében magyar nyelven a datalog alapjai olvashatók Ullman és Widom [344] könyvében. Az NP-teljesség és algoritmikus eldönthetőség kérdéseit Ivanyos Gábor, Rónyai Lajos és Szabó Réka tankönyve tárgyalja [297]. Logikai programozásról magyar nyelven Peter Flach könyvében olvashatunk [110].

A szerzők munkáját részben támogatták a T034702, T037846T és a T042706 számú OTKA-szerződések.

31. Félig strukturált adatbázisok

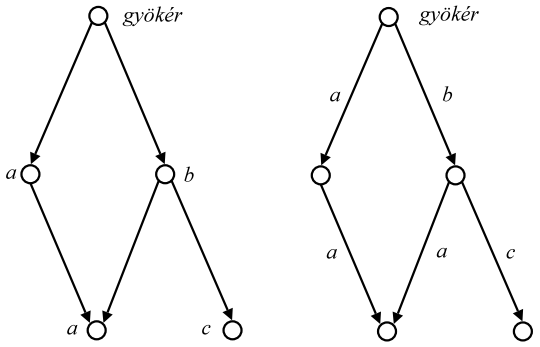
Az Internet elterjedése és az adatbázisok elméletének fejlődése kölcsönösen hatnak egymásra. Az Interneten megjelenő oldalak tartalmát sok esetben adatbázisrendszerek tárolják, másrészt az oldalak és a köztük kialakított kapcsolatok együttesen is tekinthetők egy olyan adatbázisnak, amelynek nincs a szokásos értelemben rögzített sémája. Az oldalak tartalmát és az oldalak közti kapcsolatokat maguk az oldalak írják le, ilyen értelemben csak félig strukturált adatokról beszélhetünk, melyeket legjobban irányított, címkézett gráfokkal lehet jellemezni. A félig strukturált adatok esetén az adatszerkezetek és a lekérdezések megadása során sokkal gyakrabban használunk rekurzív technikákat, mint a klasszikus relációs adatbázisok esetében. Ennek megfelelően kell az adatbázisok különböző problémaköreit, mint például a megszorításokat, függőségeket, lekérdezéseket, osztott tárolást, jogosultságokat, bizonytalanság kezelését, általánosítani. A félig strukturáltság új kérdések felvetését is jelenti. Mivel a klasszikus adatbázisoktól eltérően a lekérdezések nem minden esetben alkotnak zárt rendszert, azaz a lekérdezések egymás utáni alkalmazhatósága függ az eredmény típusától, ezért nagyobb jelentőséget kap a típusok ellenőrzésének problémaköre.

A relációs adatbázisok esetén az elméleti megalapozás szoros kapcsolatban áll a véges modellelmélettel. Félig strukturált esetben az automaták, azon belül is a faautomaták kapnak nagyobb hangsúlyt.

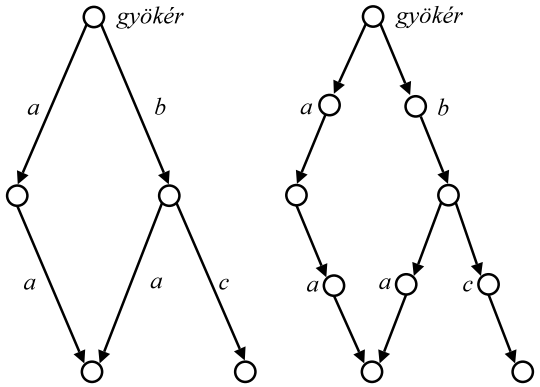
31.1. Félig strukturált adatok és az XML

A félig strukturált adatokon irányított, gyökérrel rendelkező címkézett gráfot értünk. A gyökér egy kitüntetett csúcs, amelybe nem mutat él. A gráf csúcsai azonosítóval megkülönböztetett objektumok. Az objektumok atomi vagy összetett típusúak. Az összetett típusú objektumot irányított élek kapcsolják egy vagy több objektumhoz. Az atomi típusú objektumokhoz társulnak az adatértékek. Kétféle modellt szokás használni, az egyikben a csúcsok címkézettek, a másikban az élek. Az utóbbi az általánosabb, mivel minden csúcscímkézett gráfnak egyértelműen megfeleltethető például az az élcímkézett gráf, amelyben minden él azt a címkét kapja, amilyen címkéje annak a csúcsnak van, ahová az él mutat. Ezzel olyan élcímkézett irányított gráfot kapunk, amelyre teljesül, hogy az egy csúcsba mutató éleknek ugyanaz a címkéje. Ezzel a megfeleltetéssel az élcímkézett gráfokra vonatkozó fogalmak, definíciók, állítások speciális esetként átfogalmazhatók csúcscímkézett gráfokra.

Ha élcímkézett gráfból akarunk csúcscímkézett gráfot kapni, akkor a szokásos megfeleltetés a következő. Ha egy (u, v) él címkéje c, akkor töröljük ezt az élt, és vezessünk be



31.1. ábra. Csúcscímkézett gráfnak megfelelő élcímkézett gráf.



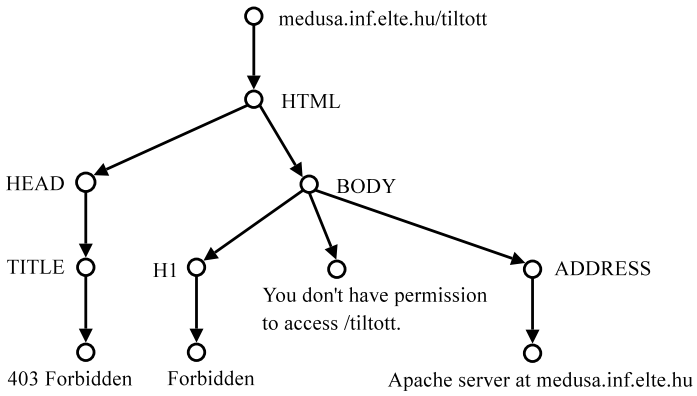
31.2. ábra. Élcímkézett gráf és a megfelelő csúcscímkézett gráf.

egy új w csúcsot, amelynek címkéje legyen c, majd húzzuk be az (u, w) és (w, v) éleket. Így egy n csúcsból, és m élből álló élcímkézett gráfból $m + n$ csúcsból, és $2m$ élből álló csúcscímkézett gráfot kapunk. Ezáltal a csúcscímkézett gráfokra kimondott algoritmusok és költségbecslések átírhatók élcímkézett gráfokra.

Mivel a gyakorlatban használt leírások inkább csúcscímkézett gráfokat használnak, ezért a fejezetben végig csúcscímkézett gráfokról lesz szó.

Az eredetileg dokumentumrendszerekhez kifejlesztett ***XML*** (e**X**tensible **M**arkup **L**anguage) nyelv egymásba ágyazott, rendezett, névvel ellátott elemek leírására szolgál, ily módon alkalmas a félig strukturált adatok közül a fák ábrázolására. Mivel a tágabban értelmezett XML nyelvben az elemek között hivatkozások is megadhatók, ezáltal tetszőleges félig strukturált adatok leírására is használható az XML nyelv.

Az alábbiakban a medusa.inf.elte.hu/tiltott címen található oldal leírását adtuk meg XML formában. A leírás szerkezeti jellemzői alapján, természetes módon kapható a 31.3. ábrán megadott csúcscímkézett fa.



31.3. ábra. A „tiltott” nevű XML fájlnak megfelelő gráf.

```
<HTML>
        <HEAD>
                <TITLE>403 Forbidden</TITLE>
        </HEAD>
        <BODY>
                <H1>Forbidden</H1>
                You don’t have permission to access /tiltott.
                <ADDRESS>Apache Server at medusa.inf.elte.hu </ADDRESS>
        </BODY>
</HTML>
```

Gyakorlatok

31.1-1. Adjunk meg egy csúcscímkézett gráfot, amely ennek a fejezetnek a szerkezetét és szövegformázásait reprezentálja.

31.1-2. Hány olyan különböző irányított, csúcscímkézett gráf adható meg, amelyben a csúcsok száma n, az élek száma m, a lehetséges címkék száma k? Ezek közül mennyi az izomorfia erejéig különböző? Milyen értékeket kapunk $n = 5, m = 7, k = 2$ esetére?

31.1-3. Tekintsünk egy fát, amelyben bármely csúcs gyerekeit különböző sorszámokkal látjuk el. Bizonyítsuk be, hogy a fa csúcsai megcímkézhetők (a_v, b_v) párokkal, ahol a_v és b_v természetes számok, úgy, hogy teljesüljenek a következők:

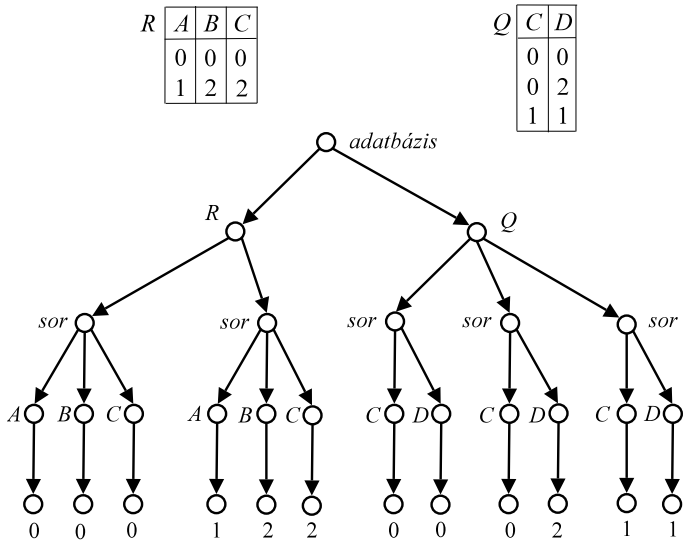
a. $a_v < b_v$, minden v csúcsra.

b. Ha v-nek az u leszármazottja, akkor $a_v < a_u < b_u < b_v$.

c. Ha u és v testvérek, és $sorszám(u) < sorszám(v)$, akkor $b_u < a_v$.

31.2. Sémák és szimulációk

A relációs adatbázisok esetében a sémák fontos szerepet töltenek be az adatok leírásában, lekérdezésében, a lekérdezések optimalizálásában, illetve a hatékonyságot növelő tárolási



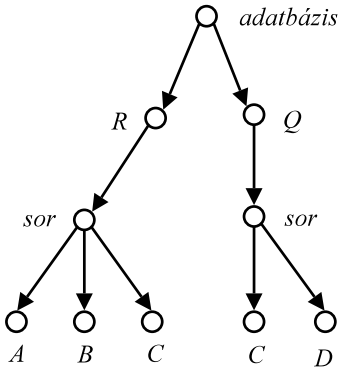
31.4. ábra. Egy relációs adatbázis megadása félig strukturált modellben.

eljárásokban. A félig strukturált esetben a sémát a gráfból kell visszanyerni. A séma korlátozza a gráf útvonalaihoz tartozó címkesorozatokat.

A 31.4. ábrán az $R(A, B, C)$ és $Q(C, D)$ relációs sémával rendelkező relációkat látjuk, illetve a nekik megfelelő félig strukturált leírásukat. A fa leveleinek címkéi a relációs sorok adatértékei. A gyökérből az adatértékekhez vezető irányított útvonalak a következő különböző címkesorozatokat tartalmazzák: *adatbázis.R.sor.A, adatbázis.R.sor.B, adatbázis.R.sor.C, adatbázis.Q.sor.C, adatbázis.Q.sor.D*. Ezt tekinthetjük a félig strukturált adatbázis sémájának. Vegyük észre, hogy a séma is egy gráf, ahogy ez a 31.5. ábrán látható. A két gráf diszjunkt egyesítése is gráf, amelyen az alábbi módon értelmezhetünk egy szimulációs leképezést. Ezzel teremtünk kapcsolatot az eredeti gráf és a sémának megfelelő gráf között.

31.1. definíció. *Legyen $G = (V, E, A, címke())$ csúcscímkézett irányított gráf, ahol V a csúcsok, E az élek, A a címkék halmaza, és $címke(v)$ a v csúcs címkéjével egyenlő. Jelölje $E^{-1}(v) = \{u \mid (u, v) \in E\}$ a V csúcsba mutató élek kiinduló csúcsainak halmazát. Egy s bináris reláció ($s \subseteq V \times V$) **szimuláció**, ha $s(u, v)$ esetén a következő 2 feltétel teljesül:*
i) $címke(u) = címke(v)$ és
ii) minden $u' \in E^{-1}(u)$ esetén létezik olyan $v' \in E^{-1}(v)$, melyre $s(u', v')$
*Egy **v csúcs szimulálja az u csúcsot,** ha van olyan s szimuláció, amelyre $s(u, v)$. Az **u és v csúcs hasonló,** $u \approx v$, ha u szimulálja v-t és v szimulálja u-t.*

Könnyű látni, hogy az üres reláció szimuláció, szimulációk egyesítése is szimuláció, mindig létezik maximális szimuláció, és végül a hasonlóság ekvivalencia reláció. Nem jelent lényegi változtatást, ha a definícióban $E^{-1}(u)$ helyett E-t írunk, mivel ez csak azt jelenti, hogy a gráfban az élek irányítását megfordítottuk.



31.5. ábra. A 31.4. ábrán megadott félig strukturált adatbázis sémája.

Azt mondjuk, hogy a D gráf szimulálja az S gráfot, ha létezik olyan $f : V_S \mapsto V_D$ leképezés, amelyre a $(v, f(v))$ reláció szimuláció a $V_S \times V_D$ halmazon.

Kétféle sémát szokás megkülönböztetni, alsó korlátot és felső korlátot. Ha a D adatgráf szimulálja az S sémagráfot, akkor azt mondjuk, hogy ***S alsó korlátja D-nek.*** Vegyük észre, hogy ez azt jelenti, hogy S-ben az irányított utakhoz tartozó összes címkesorozat D-ben is előfordul valamilyen út mentén. Ha S szimulálja D-t, akkor az ***S felső korlátja D-nek.*** Felső korlát esetén a D-ben előforduló címkesorozatok S-ben is előfordulnak.

Félig strukturált adatbázisok esetén fontos szerepet játszanak azok a sémák, amelyek maximális alsó korlátok, vagy minimális felső korlátok.

Az S és D gráfok közti éltartó leképezést morfizmusnak hívjuk. Vegyük észre, hogy az f akkor és csak akkor morfizmus, ha D szimulálja az S-et. Annak eldöntése, hogy létezik-e S-ről D-re morfizmus, NP-teljes probléma. Ezzel szemben az alábbiakban belátjuk, hogy a maximális szimuláció kiszámítása PTIME bonyolultságú.

Jelöljük $szim(v)$-vel azokat a csúcsokat, amelyek szimulálják az u-t. A maximális szimuláció kiszámítása ekvivalens az összes $szim(v)$ halmaz meghatározásával, ahol $v \in V$. Először a definíción alapuló naiv kiszámítást vesszük.

MAXIMÁLIS-SZIMULÁCIÓ-NAIV-MÓDON(G)

```
1  for minden v ∈ V
2      do szim(v) ← {u ∈ V | címke(u) = címke(v)}
3  while ∃ u, v, w ∈ V :  v ∈ E^{-1}(u)  ∧  w ∈ szim(u)  ∧  E^{-1}(w) ∩ szim(v) = ∅
4          do szim(u) ← szim(u) \ {w}
5  return {szim(u) | u ∈ V}
```

31.2. állítás. *A* MAXIMÁLIS-SZIMULÁCIÓ-NAIV-MÓDON *algoritmus* $O(m^2n^3)$ *idő alatt kiszámolja a maximális szimulációt, ha* $m \geq n$.

Bizonyítás. Abból indulunk ki, hogy mik lehetnek a $szim(u)$ elemei. Ha egy (v, u) él alapján a $szim(u)$ w eleme a definíció szerint nem szimulálja az u-t, akkor kivesszük a w-t a $szim(u)$

halmazból. Ilyenkor azt mondjuk, hogy a (v, u) él szerint élesítettük a $szim(u)$ halmazt. Ha már egyik $szim(u)$ halmazt sem lehet egyik él szerint sem élesíteni, akkor a $szim(u)$ minden eleme szimulálja az u-t. Az állítás igazolásához már csak azt kell észrevenni, hogy a **while** ciklus legfeljebb n^2 iterációból állhat. ■

Az algoritmus hatékonyságán speciális adatszerkezetek felhasználásával javíthatunk. Először vezessük be azt a $szim(u)$-nál bővebb *szim-jelölt*(u) halmazt, amelynek elemeiről meg akarjuk állapítani, hogy szimulálják-e az u-t.

MAXIMÁLIS-SZIMULÁCIÓ-JAVÍTVA(G)

```
 1  for minden v ∈ V
 2      do szim-jelölt(u) ← V
 3         if E^{-1}(v) = ∅
 4            then szim(v) ← {u ∈ V | címke(u) = címke(v)}
 5            else  szim(v) ← {u ∈ V | címke(u) = címke(v) ∧ E^{-1}(u) ≠ ∅}
 6  while ∃ v ∈ V : szim(v) ≠ szim-jelölt(v)
 7        do törlésre-jelölt ← E(szim-jelölt(v)) \ E(szim(v))
 8           for minden u ∈ E(v)
 9               do szim(u) ← szim(u) \ törlésre-jelölt
10           szim-jelölt(v) ← szim(v)
11  return {szim(u) | u ∈ V}
```

A javított algoritmus **while** ciklusa a következő invariáns tulajdonságokkal rendelkezik.

I_1: $\forall\, v \in V :\ szim(v) \subseteq$ *szim-jelölt*(v).
I_2: $\forall\, u, v, w \in V : (v \in E^{-1}(u) \ \wedge\ w \in szim(u)) \ \Rightarrow\ (E^{-1}(w) \cap$ *szim-jelölt*$(v) \neq \emptyset)$.

Amikor a $szim(u)$ halmazt egy (v, u) éllel élesítjük, akkor azt ellenőrizzük, hogy egy $w \in szim(u)$ elemnek van-e $szim(v)$-ben szülője. Az I_2 miatt $szim(v)$ helyett ezt elég a *szim-jelölt*(v) elemeire ellenőrizni, és ha egy $w' \in$ *szim-jelölt*$(v) \setminus szim(v)$ elemet egyszer már figyelembe vettünk, akkor véglegesen eltávolítjuk a *szim-jelölt*(v) halmazból.

Tovább javíthatjuk az algoritmust, ha a **while** ciklus iterációiban nem számoljuk ki újra a *törlésre-jelölt* halmazt, hanem dinamikusan tartjuk karban.

MAXIMÁLIS-SZIMULÁCIÓ-HATÉKONYAN(G)

```
1  for minden v ∈ V
2      do szim-jelölt(v) ← V
3         if E^{-1}(v) = ∅
4            then szim(v) ← {u ∈ V | címke(u) = címke(v)}
5            else  szim(v) ← {u ∈ V | címke(u) = címke(v) ∧ E^{-1}(u) ≠ ∅}
6      törlésre-jelölt(v) ← E(V) \ E(szim(v))
```

```
 7 while ∃ v ∈ V : törlésre-jelölt(v) ≠ ∅
 8       do for minden u ∈ E(v)
 9          do for minden w ∈ törlésre-jelölt(v)
10             do if w ∈ szim(u)
11                   then szim(u) ← szim(u) \ {w}
12                        for minden w'' ∈ E(w)
13                           do if E^{-1}(w'') ∩ szim(u) = ∅
14                                 then törlésre-jelölt(u) ← törlésre-jelölt(u) ∪ {w''}
15          szim-jelölt(v) ← szim(v)
16          törlésre-jelölt(v) ← ∅
17 return {szim(u) | u ∈ V}
```

A fenti algoritmus **while** ciklusa az alábbi invariánssal rendelkezik.

I_3: $\forall\, v \in V$: *törlésre-jelölt*$(v) = E($*szim-jelölt*$(v)) \setminus E($*szim*$(v))$.

Az algoritmus megvalósításához használjunk egy *számláló* $n \times n$-es tömböt. A *számláló*$[w'', u]$ értéke legyen egyenlő az $|E^{-1}(w'') \cap$ *szim*$(u)|$ nem negatív egész számmal. A számláló kezdeti értékeinek beállítása $O(mn)$ időben történik. Amikor w elemet eltávolítjuk a *szim*(u) halmazból, akkor a w összes w'' gyerekére csökkenteni kell a *számláló*$[w'', u]$ értéket. Ezzel elérjük, hogy a legbelső if feltételt konstans időben tudjuk ellenőrizni. Az algoritmus elején a *szim*(v) halmazok kezdeti értékeinek beállítása $O(n^2)$ időben történik, feltéve, hogy $m \geq n$. A *törlésre-jelölt*(v) halmazok beállítása összesen $O(mn)$ időt igényel. Tetszőleges v és w csúcsok esetén, ha $w \in$ *törlésre-jelölt*(v) igaz a **while** ciklus i-edik iterációjában, akkor minden j-edik iterációban ugyanez már hamis lesz, feltéve, hogy $j > i$. Ugyanis $w \in$ *törlésre-jelölt*(v)-ből következik, hogy $w \notin E($*szim*$(v))$, és $j > i$ esetén a *szim-jelölt*(v) értéke a j-edik iterációban részhalmaza a *szim*(v) i-edik iterációban vett értékének, valamint tudjuk, hogy teljesül az I_3 invariáns. Emiatt annak ellenőrzése, hogy $w \in$ *szim*(u), végrehajtható $\sum_v \sum_w |E(v)| = O(mn)$ idő alatt. A $w \in$ *szim*(u) ellenőrzés minden w és u csúcsra legfeljebb egyszer ad igaz értéket, mivel ha egyszer teljesül a feltétel, akkor a w-t eltávolítjuk a *szim*(u) halmazból, és többé már nem kerül oda vissza. Ebből következik, hogy a **while** ciklus külső **if** feltételének kiszámítása $\sum_v \sum_w (1 + |E(v)|) = O(mn)$ időben történik.

Ezzel beláttuk a következő állítást.

31.3. állítás. *A* Maximális-szimuláció-hatékonyan *algoritmus $O(mn)$ idő alatt kiszámolja a maximális szimulációt, ha $m \geq n$.*

Ha egy szimuláció inverze is szimuláció, akkor biszimulációról beszélünk. Az üres reláció biszimuláció, és mindig létezik maximális biszimuláció. A maximális biszimuláció hatékonyabban számolható ki, mint a szimuláció. A maximális biszimuláció a PT-algoritmussal $O(m \lg n)$ időben határozható meg. Élcímkézett gráfok esetén ugyanerre $O(m \lg(m+n))$ költség adódik.

Mint ahogy azt látni fogjuk, a biszimulációk fontos szerepet játszanak a félig strukturált adatbázisok indexelésében, mivel egy gráf biszimuláció szerinti hányados gráfja pontosan azokat a címkesorozatokat tartalmazza, mint a gráf. Megjegyezzük, hogy a gyakorlatban

szimuláció helyett még úgynevezett ***DTD*** leírásokat is szokás sémának tekinteni. A DTD reguláris nyelven megfogalmazott adattípus-definíciókat tartalmaz.

Gyakorlatok

31.2-1. Mutassunk példát arra, hogy a szimulációból nem következik a biszimuláció.

31.2-2. Irányított, nem feltétlen körmentes, csúcscímkézett G gráfra értelmezzünk egy *fásít*(G) műveletet a következőképpen. A művelet eredménye egy nem feltétlen véges G' gráf, melynek csúcsai a G gyökérből induló irányított útjai, az utak címkéi a hozzájuk tartozó címkesorozatok. A p_1 csúcsból húzzunk egy élt p_2-be, ha p_2 végpontját elhagyva p_1-et kapjuk. Bizonyítsuk be, hogy G és *fásít*(G) hasonlók a biszimulációra nézve.

31.3. Lekérdezések és indexek

A félig strukturált adatbázisban tárolt információt lekérdezések segítségével lehet kinyerni. Ehhez először rögzítjük a feltehető kérdések formáját, vagyis megadunk egy ***lekérdező nyelvet,*** majd definiáljuk a kérdések értelmét, vagyis a ***lekérdezés kiértékelését*** egy félig strukturált adatbázisra vonatkozóan. A hatékony kiértékeléshez általában indexeket használunk. Az ***indexelés*** lényege, hogy az adatbázisban tárolt adatokat valamilyen hasonlósági elven összevonjuk, azaz egy olyan indexet állítunk elő, amely tükrözi az eredeti adatok struktúráját. Az eredeti lekérdezést az indexben hajtjuk végre, majd az eredmény alapján megkeressük az indexértékeknek megfelelő adatokat az eredeti adatbázisban. Az index mérete általában jóval kisebb az eredeti adatbázisnál, ezáltal a lekérdezések gyorsabb végrehajtását lehet elérni. Megjegyezzük, hogy a klasszikus adatbázisok esetén használatos invertált lista típusú indexet integrálni lehet a következőkben bevezetett sématípusú indexekkel. Ez különösen akkor előnyös, mikor XML dokumentumokban kulcsszavak alapján keresünk.

Elsőként a reguláris kifejezéseket tartalmazó lekérdező nyelvvel, és az ehhez használatos indextípusokkal ismerkedünk meg.

31.4. definíció. *Legyen adva a* $G = (V, E, \text{gyökér}, \Sigma, \text{címke}, \text{azon}, \text{érték})$ *irányított, csúcscímkézett gráf, ahol* V *a csúcsok,* $E \subseteq V \times V$ *az élek,* Σ *a címkék halmaza,* Σ *tartalmaz két speciális címkét, a GYÖKÉR és ÉRTÉK címkéket. A* címke(v) *a* v *csúcs címkéje. Az* azon(v) *a* v *csúcs azonosítója, a gyökér egy olyan csúcs, melynek címkéje GYÖKÉR, és amelyből minden csúcs elérhető irányított úton keresztül. Ha* v *levél, azaz nem vezet ki él belőle, akkor a címkéje ÉRTÉK, és érték*(v) *a* v *levélhez tartozó adatérték. Úton mindig irányított utat értünk, azaz olyan* $n_0, \ldots, n_p$ *csúcsokból álló sorozatot, amelyben* n_i *csúcsból vezet él* n_{i+1}*-be, ha* $0 \le i \le p-1$*. Egy címkékből álló* $l_0, \ldots, l_p$ *sorozatot* ***címkesorozatnak*** *vagy másképpen* ***egyszerű kifejezésnek*** *hívunk. Az* $\mathbf{n_0, \ldots, n_p}$ ***út illeszkedik az*** $\mathbf{l_0, \ldots, l_p}$ ***címkesorozatra,*** *ha* címke(n_i) $= l_i$ *, minden* $0 \le i \le p$ *esetén.*

A reguláris kifejezéseket rekurzív módon definiáljuk.

31.5. definíció. *Legyen* $R ::= \varepsilon \mid \Sigma \mid \_ \mid R.R \mid R|R \mid (R) \mid R? \mid R^*$*, ahol* ***R reguláris kifejezés****, továbbá* ε *az üres kifejezés,* _ *tetszőleges címkét jelöl, . az egymás után következést,* | *a logikai vagy műveletet, ? az opcionális választást, * a véges ismétlést jelenti.* ***L(R)*** *jelölje az R által meghatározott címkesorozatokból álló reguláris nyelvet. Egy* ***n csúcs illeszkedik***

egy címkesorozatra, *ha van olyan gyökérből induló és n csúccsal végződő út, amely illeszkedik a címkesorozatra. Egy* ***n csúcs illeszkedik az R reguláris kifejezésre****, ha van olyan címkesorozat az $L(R)$ nyelvben, amelyre az n csúcs illeszkedik. Az R reguláris kifejezés által meghatározott* ***lekérdezés eredménye egy G gráfon*** *azoknak a csúcsoknak az $R(G)$ halmaza, amelyek illeszkednek az R kifejezésre.*

Mivel a reguláris kifejezések kiértékelésénél mindig gyökérből induló utakat keresünk, ezért a címkesorozatok első eleme mindig a GYÖKÉR, amit a rövidség kedvéért elhagyhatunk.

Megjegyezzük, hogy a reguláris kifejezéshez tartozó $L(R)$ nyelvek halmaza metszetre zárt, és az $L(R) = \emptyset$ probléma eldönthető probléma.

A lekérdezések eredménye kiszámolható az R reguláris kifejezésnek megfelelő nem determinisztikus A_R automata segítségével. Az algoritmus rekurzív megadása a következő.

NAIV-KIÉRTÉKELÉS(G, A_R)

1 *Bejárt* $\leftarrow \emptyset$ ▷ Ha s állapotban jártunk az u csúcsban, akkor (u, s) bekerül a *Bejárt* halmazba.
2 BEJÁR (*gyökér*(G), *kezdőállapot*(A_R))

BEJÁR(u, s)

1 **if** $(u, s) \in$ *Bejárt*
2 **then return** *eredmény*$[u, s]$
3 *Bejárt* $\leftarrow$ *Bejárt* $\cup \{(u, s)\}$
4 *eredmény*$[u, s] \leftarrow \emptyset$
5 **for** minden $s \xrightarrow{\epsilon} s'$ ▷ Ha s állapotból ϵ jelet olvasva s' állapotba kerülünk.
6 **do if** $s' \in$ *végállapot*(A_R)
7 **then** *eredmény*$[u, s] \leftarrow \{u\} \cup$ *eredmény*$[u, s]$
8 *eredmény*$[u, s] \leftarrow$ *eredmény*$[u, s] \cup$ BEJÁR(u, s')
9 **for** minden $s \xrightarrow{címke(u)} s'$ ▷ Ha s állapotból *címke*(u) jelet olvasva s' állapotba kerülünk.
10 **do if** $s' \in$ *végállapot*(A_R)
11 **then** *eredmény*$[u, s] \leftarrow \{u\} \cup$ *eredmény*$[u, s]$
12 **for** minden v, ahol $(u, v) \in E(G)$ ▷ Az u gyerekeire rekurzívan folytatjuk a bejárást.
13 **do** *eredmény*$[u, s] \leftarrow$ *eredmény*$[u, s] \cup$ BEJÁR(v, s')
14 **return** *eredmény*$[u, s]$

31.6. állítás. *Adott R reguláris lekérdezés és G gráf esetén az $R(G)$ kiszámítási költsége a G élei számának és az R-nek megfelelő véges nem determinisztikus automata állapotai számának polinomja.*

Bizonyítás. A bizonyítás vázlatosan a következő. Legyen az R-hez tartozó nem determinisztikus véges automata A_R. Jelölje $|A_R|$ az A_R állapotainak számát. Tekintsük az m élből álló G gráfnak a BEJÁR algoritmus szerinti szélességi bejárását a gyökérből kiindulva. A bejárás közben a csúcsból kiolvasott címke alapján jutunk az automata új állapotába, amelyet minden csúcshoz eltárolunk. Ha az automata elfogadó végállapotba került, akkor a csúcs

megoldás. A bejárás során időnként vissza kell lépnünk egy él mentén, hogy arra menjünk tovább, amerre még nem jártunk. Belátható, hogy a bejárás során minden állapotban legfeljebb egyszer kell egy élen áthaladni, vagyis az automata legfeljebb ennyi lépést hajt végre. Ez összesen $O(|A_R|m)$ lépés. Ezek alapján következik az állítás. ■

A G gráfban két csúcs ***nem különböztethető meg reguláris kifejezéssel,*** ha nincs olyan reguláris R, amelynek eredményében az egyik csúcs benne van, a másik pedig nincs benne. Nyilvánvaló, hogy ha két csúcs nem különböztethető meg, akkor a címkéjük megegyezik. Osztályozzuk a csúcsokat. Egy osztályba kerüljenek az azonos címkéjű csúcsok. Ezzel a csúcsok halmazának egy P partícióját állítottuk elő. Ez lesz az ***alappartíció.*** Az is könnyen végiggondolható, hogy ha két csúcs nem különböztethető meg, akkor ez a tulajdonság a szülőkre is öröklődik. Ebből következik, hogy a gyökérből a megkülönböztethetetlen csúcsokhoz vezető utakhoz tartozó címkesorozatok halmaza is megegyezik. Legyen minden n csúcsra $L(n) = \{l_0, \ldots, l_p \mid n$ illeszkedik az $l_0, \ldots, l_p$ címkesorozatra$\}$. Az n_1 és n_2 csúcsok akkor és csak akkor nem különböztethetők meg, ha $L(n_1) = L(n_2)$. Ha azok a csúcsok kerülnek egy osztályba, amelyek $L(n)$ értéke megegyezik, akkor a P partíciónak egy P' finomítását kapjuk. Erre a partícióra az is teljesül, hogy ha egy n csúcs szerepel egy R reguláris lekérdezés eredményében, akkor az n csúccsal egy osztályba tartozó minden csúcs is benne lesz a lekérdezés eredményében.

31.7. definíció. *Legyen adva egy $G = (V, E, gyökér, \Sigma, címke, azon, érték)$ gráf és legyen P a V egy olyan partíciója, amely az alappartíció finomítása, azaz az egy osztályba tartozó csúcsok címkéje megegyezik. Ekkor az $I(G) = (P, E', gyökér', \Sigma, címke', azon', érték')$ gráfot* ***indexnek*** *hívjuk. Az indexgráf csúcsai a P partíció osztályai, továbbá $(I, J) \in E'$ akkor és csak akkor, ha van olyan $i \in I$ és $j \in J$, melyekre $(i, j) \in E$. Ha $I \in P$, akkor $azon'(I)$ az I indexcsúcs azonosítója, és $címke'(I) = címke(n)$, ahol $n \in I$. A gyökér' a P partíciónak az az osztálya, amely a G gyökerét tartalmazza. Ha $címke(I) = ÉRTÉK$, akkor $érték'(I) = \{érték(n) \mid n \in I\}$.*

Ha adott a V halmaznak egy P partíciója, akkor $n \in V$ csúcs esetén ***osztály(n)*** jelölje a P partíciónak azt az osztályát, amelybe n tartozik. Indexek esetén az $I(n)$ jelölést is használhatjuk az $osztály(n)$ helyett.

Vegyük észre, hogy az indexek lényegében a csúcsok különböző partícióival azonosíthatók, így ha nem okoz félreértést, akkor a partíciókat is indexeknek hívjuk. Azok lesznek a jó indexek, amelyek kis méretűek, és a lekérdezések eredménye a gráfon és az indexen megegyezik. Az indexeket gyakran úgy adjuk meg, hogy a csúcsokon definiálunk egy ekvivalenciarelációt, és az indexnek megfelelő partíció az ekvivalencia osztályokból áll.

31.8. definíció. *Legyen P az a partíció, amelynek bármelyik I osztályára igaz, hogy $n, m \in I$ akkor és csak akkor, ha $L(n) = L(m)$. Ekkor a P alapján készített $I(G)$ indexet* ***naiv indexnek*** *hívjuk.*

Naiv index esetén a P partícióban szereplő I osztály minden n eleméhez ugyanaz az $L(n)$ nyelv tartozik, melyet $L(I)$-vel fogunk jelölni.

31.9. állítás. *Legyen I a naiv index egy csúcsa és R egy reguláris kifejezés. Ekkor $I \cap R(G) = \emptyset$ vagy $I \subseteq R(G)$.*

Bizonyítás. Legyen $n \in I \cap R(G)$ és legyen $m \in I$. Ekkor van olyan $l_0, \ldots, l_p$ címkesorozat az $L(R)$-ben, amelyre n illeszkedik, azaz $l_0, \ldots, l_p \in L(n)$. Mivel $L(n) = L(m)$ így m is illeszkedik erre a címkesorozatra, azaz $m \in I \cap R(G)$. ■

NAIV-INDEXES-KIÉRTÉKELÉS(G, R)

```
1 legyen I_G a G naiv indexe
2 Q ← ∅
3 for minden I ∈ NAIV-KIÉRTÉKELÉS(I_G, A_R)
4     do Q ← Q ∪ I
5 return Q
```

31.10. állítás. *A* NAIV-INDEXES-KIÉRTÉKELÉS *által előállított Q megegyezik $R(G)$-vel.*

Bizonyítás. Az előző állítás miatt egy I osztálynak vagy minden eleme beletartozik a lekérdezés eredményébe vagy egyik sem tartozik hozzá. ■

A naiv index használatával tehát a lekérdezést ki tudjuk értékelni, de a következő állítás szerint nem elég hatékonyan. Az állítást Stockmeyer és Meyer bizonyította be 1973-ban.

31.11. állítás. *A* NAIV-INDEXES-KIÉRTÉKELÉS *algoritmushoz szükséges I_G naiv index előállítása PSPACE-teljes probléma.*

A másik probléma a naiv index használatával, hogy az $L(I)$ halmazok különböző I esetén nem feltétlen diszjunktak, így ez a tárolásban redundanciát jelenthet.

A fentiek miatt a naiv index partíciójának olyan finomítását keressük, amelyet már hatékonyan tudunk előállítani, és amely segítségével $R(G)$-t továbbra is elő tudjuk állítani.

31.12. definíció. *Az $I(G)$ index **biztonságos**, ha bármilyen $n \in V$ és $l_0, \ldots, l_p$ címkesorozat esetén, melyekre teljesül, hogy n illeszkedik az $l_0, \ldots, l_p$ címkesorozatra a G gráfban, igaz, hogy osztály(n) illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $I(G)$ gráfban. Az $I(G)$ index **pontos**, ha az index bármely I osztálya és $l_0, \ldots, l_p$ címkesorozat esetén, melyekre teljesül, hogy I illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $I(G)$ gráfban, igaz, hogy tetszőleges $n \in I$ csúcs illeszkedik az $l_0, \ldots, l_p$ címkesorozatra a G gráfban.*

A biztonságosság azt jelenti, hogy az index alapján kiértékelt eredményben szerepelő csúcsok tartalmazzák a reguláris lekérdezés eredményét, vagyis $R(G) \subseteq R(I(G))$, a pontosság ennek a fordítottja, azaz az index alapján történő kiértékelés nem ad hamis eredményt, azaz $R(I(G)) \subseteq R(G)$. A biztonságosság és az index éleinek definícióiból rögtön következik az alábbi állítás.

31.13. állítás. *1. Minden index biztonságos.*
2. A naiv index biztonságos és pontos.

Ha I a G csúcsainak egy halmaza, akkor az $L(I)$ nyelvet, vagyis azt a nyelvet, amelybe tartozó címkesorozatokhoz az I elemei illeszkednek, eddig a G gráf alapján értelmeztük.

Ha ezt jelölni akarjuk, akkor az $L(I,G)$ jelölést használjuk. $L(I)$-t értelmezhetjük az $I(G)$ gráfban is, amelynek I egy csúcsa. Ekkor $L(I)$ helyett az $L(I,I(G))$ jelölést alkalmazzuk, mely az összes olyan címkesorozatot jelenti, amelyre az I csúcs illeszkedik az $I(G)$ gráfban. A biztonságos és pontos indexek esetén $L(I,G) = L(I,I(G))$, és ilyenkor egyszerűen $L(I)$-t írhatunk. Ebben az esetben az $L(I)$ kiszámítását az $I(G)$ alapján végezzük, mivel az $I(G)$ mérete általában kisebb a G méreténél.

Tetszőleges típusú indexgráfot lekérdezhetünk a NAIV-KIÉRTÉKELÉS algoritmussal. Ezután egyesítjük a talált indexcsúcsokat. Ha pontos indexet használtunk, akkor ugyanazt kapjuk, mintha az eredeti gráfot kérdeztük volna le.

INDEXES-KIÉRTÉKELÉS$(G, I(G), A_R)$

```
1 legyen I(G) a G indexe
2 Q ← ∅
3 for minden I ∈ NAIV-KIÉRTÉKELÉS(I(G), A_R)
4     do Q ← Q ∪ I
5 return Q
```

Először egy hatékonyan előállítható, biztonságos és pontos indexet fogunk megadni, amely a csúcsokon értelmezett hasonlóságon alapul. Így kapjuk az 1-indexet. Ennek nagy méretét csökkenthetjük, ha csak lokális hasonlóságot követelünk meg. Az így kapott A(k)-index esetén elveszítjük a pontosságot, vagyis az INDEXES-KIÉRTÉKELÉS alapján olyan eredményt is kaphatunk, amely nem szerepel az R reguláris lekérdezésre adott válaszban, ezért az eredményeket még tesztelnünk kell, hogy megőrizzük a pontosságot.

31.14. definíció. *Legyen $\approx$ egy olyan ekvivalencia reláció a V halmazon, amelyre teljesül, hogy $u \approx v$ esetén*
i) címke(u) = címke(v),
ii) ha az u' csúcsból vezet él az u csúcsba, akkor van olyan v' csúcs, amelyből vezet él a v csúcsba és $u' \approx v'$.
iii) ha a v' csúcsból vezet él az v csúcsba, akkor van olyan u' csúcs, amelyből vezet él az u csúcsba és $u' \approx v'$.
*A fenti ekvivalencia relációt **biszimulációnak** nevezzük. Egy gráf u és v csúcsa **biszimuláns** akkor és csak akkor, ha létezik olyan $\approx$ biszimuláció, amelyre $u \approx v$.*

31.15. definíció. *Legyen P az a partíció, amely egy biszimuláció ekvivalencia osztályaiból áll. A P partíció alapján definiált indexet **1-indexnek** nevezzük.*

31.16. állítás. *Az 1-index a naiv index finomítása. Ha a G gráfban az egy csúcsba mutató élek kezdőpontjai különböző címkékkel rendelkeznek, azaz $x \neq x'$ és $(x,y),(x',y) \in E$ esetén címke(x) $\neq$ címke(x'), akkor $L(u) = L(v)$ akkor és csak akkor, ha u és v biszimuláns.*

Bizonyítás. $u \approx v$ esetén *címke*(u) = *címke*(v). Az u csúcs illeszkedjen egy $l_0,\ldots,l_p$ címkesorozatra, és legyen u' az l_{p-1} címkéhez tartozó csúcs. Ekkor van olyan v', hogy $u' \approx v'$, és $(u',u),(v',v) \in E$. Az u' illeszkedik az $l_0,\ldots,l_{p-1}$ címkesorozatra, és indukció miatt ekkor v' is illeszkedik az $l_0,\ldots,l_{p-1}$ címkesorozatra, vagyis v illeszkedik az $l_0,\ldots,l_p$ címkesorozatra, tehát ha két csúcs ugyanabban az osztályban van az 1-index szerint, akkor ugyanabban az osztályban van a naiv index szerint is.

Az állítás második részéhez elég belátni, hogy a naiv index biszimulációnak felel meg. Tartozzon u és v egy osztályba a naiv index szerint. Ekkor $címke(u) = címke(v)$. Ha $(u', u) \in E$, akkor van olyan $l_0, \ldots, l_p$ címkesorozat, amelyhez tartozó két utolsó csúcs u' és u. Ekkor a szülőkre feltett különböző címkék miatt $L(u) = L' \cup L''$ diszjunkt felbontás, ahol $L' = \{l_0, \ldots, l_p \mid u'$ illeszkedik az $l_0, \ldots, l_{p-1}$ sorozatra, és $l_p = címke(u)\}$, valamint $L'' = L(u) \setminus L'$. Mivel $L(u) = L(v)$, ezért van olyan v', amelyre $(v', v) \in E$ és $címke(u') = címke(v')$. A szülők különböző címkéi miatt $L' = \{l_0, \ldots, l_p \mid v'$ illeszkedik az $l_0, \ldots, l_{p-1}$ sorozatra, és $l_p = címke(v)\}$, vagyis $L(u') = L(v')$ és indukció miatt $u' \approx v'$, tehát $u \approx v$. ■

31.17. állítás. *Az 1-index biztonságos és pontos.*

Bizonyítás. Ha x_p illeszkedik az $l_0, \ldots, l_p$ címkesorozatra a G gráfban az $x_0, \ldots, x_p$ csúcsok miatt, akkor az indexgráf definíciója miatt $osztály(x_i)$-ből vezet él $osztály(x_{i+1})$-be, $0 \le i \le p-1$, vagyis $osztály(x_p)$ illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $I(G)$ gráfban. A pontossághoz tegyük fel, hogy I_p illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $I(G)$ gráfban az $I_0, \ldots, I_p$ miatt. Ekkor van olyan $u' \in I_{p-1}$, $u \in I_p$, melyekre $(u', u) \in E$. Legyen $v \in I_p$ tetszőleges csúcs, ekkor van olyan v', melyre $u' \approx v'$, és $(v', v) \in E$, vagyis $v' \in I_{p-1}$. Indukciót alkalmazva következik, hogy v' illeszkedik az $l_0, \ldots, l_{p-1}$ címkesorozatra az $x_0, \ldots, x_{p-2}, v'$ csúcsok miatt, de akkor v illeszkedik az $l_0, \ldots, l_p$ címkesorozatra az $x_0, \ldots, x_{p-2}, v', v$ csúcsok miatt a G gráfban. ■

Ha azt a biszimulációt tekintjük, amely esetén minden csúcs különböző partícióba kerül, akkor az ehhez az 1-indexhez tartozó $I(G)$ gráf megegyezik a G gráffal, vagyis legrosszabb esetben az $I(G)$ mérete megegyezik a G méretével, és az $I(G)$ I csúcsaihoz még tárolnunk kell az I elemeit, amely az összes I-re összesen a G csúcsainak tárolását jelenti. A lekérdezések gyorsabb kiértékelése miatt minimális méretű 1-indexet, vagyis a legdurvább 1-indexet keressük. Ellenőrizhető, hogy x és y akkor és csak akkor tartoznak ugyanabba az osztályba a legdurvább 1-index szerint, ha x és y biszimuláns.

1-INDEXES-KIÉRTÉKELÉS(G, R)

1 legyen I_1 a G legdurvább 1-indexe
2 **return** INDEXES-KIÉRTÉKELÉS(G, I_1, A_R)

Az algoritmus első lépésében a legdurvább 1-indexet kell megadni. Ezt a feladatot visszavezethetjük a legdurvább stabil partíció megtalálására, amellyel a következő részben fogunk foglalkozni. Így az ehhez használható PT-algoritmus hatékony változatával a legdurvább 1-index $O(m \lg n)$ műveleti költséggel, $O(m + n)$ tárköltséggel található meg, ahol n a G csúcsainak, m az éleinek száma.

Az I_1 gráf biztonságossága és pontossága miatt elég a lekérdezést az I_1 gráfban kiértékelni, vagyis megkeresni azokat az indexcsúcsokat, amelyek illeszkednek az R reguláris kifejezésre. Ennek költsége a 31.6. állítás alapján az I_1 gráf méretének polinomja.

Az I_1 méretét a következő paraméterek segítségével becsülhetjük. Legyen p különböző címke a G gráfban és legyen ***k a G gráf átmérője,*** vagyis a leghosszabb irányított út hossza. (Az irányított útban nem szerepelhet kétszer ugyanaz a csúcs.) Ha a gráf fa, akkor az átmérő a fa mélységével egyenlő. Gyakran készítünk olyan weboldalakat, amelyek egy d mélységű fát alkotnak, majd minden oldalhoz hozzáadunk egy q elemből álló navigációs sort, vagyis

a gráf q darab kiválasztott oldalával összekötünk minden csúcsot. Belátható, hogy ekoor a kapott gráf átmérője legfeljebb $d + q(d-1)$. Gyakorlatban a gráf méretéhez képest a d és q általában nagyon kicsi. A következő állítás bizonyítása Milo és Suciu cikkében található meg.

31.18. állítás. *Legyen G gráfban legfeljebb p különböző címke. Legyen G átmérője kisebb, mint k. Ekkor tetszőleges biszimuláció által definiált I_1 1-index mérete felülről becsülhető a k és p függvényében, függetlenül a G méretétől.*

Gyakorlatok

31.3-1. Mutassuk meg a maximális szimuláció szerinti indexről, hogy finomításra nézve az 1-index és a naiv index közé esik. Adjunk példát arra, hogy mindkét tartalmazás valódi tartalmazás.

31.3-2. Jelölje $I_s(G)$ a maximális szimuláció szerinti indexet. Igaz-e, hogy $I_s(I_s(G)) = I_s(G)$?

31.3-3. Reprezentáljuk a G gráfot, és az R reguláris lekérdezésnek megfelelő automata állapotátmeneti gráfját relációs adatbázisban. Adjunk meg egy algoritmust relációs lekérdező nyelven, például PL/SQL-ben, amely $R(G)$-t számolja ki.

31.4. Stabil partíciók és a PT-algoritmus

A félig strukturált adatbázisok lekérdezéseinek hatékony kiértékeléséhez felhasznált indexstruktúrák többsége egy gráf csúcshalmazának valamilyen particionálásán alapul. Az indexek készítését gyakran vissza lehet vezetni a legdurvább stabil partíció előállítására.

31.19. definíció. *Legyen E bináris reláció egy véges V halmazon, azaz $E \subseteq V \times V$. Ekkor V a **csúcsok**, E az irányított **élek** halmaza. Legyen tetszőleges $S \subseteq V$ esetén $E(S) = \{y \mid \exists x \in S, (x,y) \in E\}$ és $E^{-1}(S) = \{x \mid \exists\, y \in S, (x,y) \in E\}$. Tetszőleges $S \subseteq V$ és $B \subseteq V$ esetén azt mondjuk, hogy **B stabil az S-re nézve,** ha vagy $B \subseteq E^{-1}(S)$ vagy $B \cap E^{-1}(S) = \emptyset$. Legyen P a V egy partíciója, azaz V felbontása diszjunkt halmazokra, más néven blokkokra. Azt mondjuk, hogy **P stabil S-re nézve**, ha P minden blokkja stabil S-re nézve. **P stabil a P' partícióra nézve,** ha P minden blokkja stabil a P' minden blokkjára nézve. Ha P stabil minden saját blokkjára nézve, akkor a P partíció **stabil**. Legyen P és Q a V két partíciója. **Q finomítása P-nek,** vagy másképpen **P durvább mint Q,** ha Q minden blokkja előáll a P valahány blokkjának egyesítéseként. **A legdurvább stabil partíció problémája** azt jelenti, hogy adott V, E és P esetén keressük a P legdurvább stabil finomítását, azaz olyan stabil finomítását P-nek, amely durvább P minden más stabil finomításánál.*

Megjegyezzük, hogy szokás úgy is definiálni a stabilitást, hogy B stabil az S-re nézve, ha vagy $B \subseteq E(S)$ vagy $B \cap E(S) = \emptyset$. Ez nem jelent lényeges különbséget, csak az élek irányításának megfordítását, vagyis E bináris reláció helyett az inverz E^{-1} bináris relációra nézve mondjuk ki a stabilitást, ahol $(x,y) \in E^{-1}$ akkor és csak akkor, ha $(y,x) \in E$. Ugyanis $(E^{-1})^{-1}(S) = \{x \mid \exists\, y \in S, (x,y) \in E^{-1})\} = \{x \mid \exists\, y \in S, (y,x) \in E\} = E(S)$.

Legyen $|V| = n$ és $|E| = m$. Be fogjuk látni, hogy a legdurvább stabil partíció problémájára mindig létezik egyértelmű megoldás, és megadható olyan algoritmus, amely a megoldást $O(m \lg n)$ futási időben találja meg, $O(m+n)$ tárigény mellett. Ezt az algoritmust R.

Paige és R. E. Tarjan tette közzé 1987-ben, ezért ezt ***PT-algoritmusnak fogjuk*** nevezni.

Az algoritmus lényege, hogy ha egy blokk nem stabil, akkor kettéhasítjuk úgy, hogy a részek már stabilak lesznek. Először egy naiv módszert adunk meg. A hasítási művelet tulajdonságai alapján ennek hatékonyságát növelhetjük azáltal, hogy az eljárást mindig a kisebbik félre folytatjuk.

31.20. definíció. *Legyen E bináris reláció V-n, $S \subseteq V$ és Q a V egy partíciója. Legyen* ***hasít***(S, Q) *a Q-nak az a finomítása, amelyet úgy kapunk, hogy Q minden olyan B blokkját kettévágjuk, amelybe $E^{-1}(S)$ valódi módon belemetsz, azaz ha $B \cap E^{-1}(S) \neq \emptyset$ és $B \setminus E^{-1}(S) \neq \emptyset$, akkor B helyett vegyük a partícióba a $B \cap E^{-1}(S)$ és $B \setminus E^{-1}(S)$ két blokkot. Azt mondjuk, hogy S a Q* ***hasítója****, ha hasít$(S, Q) \neq Q$.*

Vegyük észre, hogy Q nem stabil S-re akkor és csak akkor, ha S a Q hasítója.

A stabilitás és a hasítás a következő tulajdonságokkal rendelkezik, amelyek bizonyítását az Olvasóra bízzuk.

31.21. állítás. *Legyen S és T a V két részhalmaza, P és Q a V két partíciója. Ekkor teljesülnek a következők.*
1. A stabilitás öröklődik a finomítás során, azaz ha Q finomítása P-nek, és P stabil S-re nézve, akkor Q is stabil S-re nézve.
2. A stabilitás öröklődik az egyesítés során, azaz ha P stabil S-re és T-re nézve, akkor P stabil $S \cup T$-re nézve.
3. A hasít művelet monoton a második argumentumban, azaz ha P finomítása Q-nak, akkor hasít(S, P) finomítása hasít(S, Q)-nak.
4. A hasít művelet kommutatív a következő értelemben. Tetszőleges S, T és P esetén teljesül, hogy hasít$(S$,hasít$(T, P)) =$ hasít$(T$,hasít$(S, P))$, és P legdurvább olyan partíciója, amely egyaránt stabil S-re és T-re nézve is megegyezik hasít$(S$,hasít$(T, P))$-vel.

A naiv algoritmus során a P partícióból kiindulva finomítjuk a Q partíciót, amíg Q minden blokkjára nézve stabil nem lesz. A finomítási lépés során Q olyan S hasítóját keressük, amely Q valahány blokkjának egyesítéseként állítható elő. Megjegyezzük, hogy elég lenne Q blokkjai közül hasítót találni, de ez a kicsit általánosabb választás lehetőséget teremt a későbbiekben az algoritmus hatékonyabbá tételére.

NAIV-PT(V, E, P)

```
1  Q ← P
2  while Q nem stabil
3        do legyen S a Q olyan hasítója, amely Q valahány blokkjának egyesítése
4           Q ← hasít(S, Q)
5  return Q
```

Vegyük észre, hogy az algoritmus végrehajtása során nem használhatjuk fel kétszer ugyanazt az S halmazt, mivel finomításkor a stabilitás öröklődik, és a 4. lépésben kapott, finomított partíció S-re nézve stabil. A felhasznált S halmazok egyesítését sem lehet később újra felhasználni, mivel az egyesítés is örökli a stabilitást. Ugyanakkor az is nyilvánvaló, hogy egy stabil partíció stabil bármely olyan S-re nézve, amely a partíció valahány blokkjának egyesítése. Ezek alapján könnyen igazolhatók a következő állítások.

31.22. állítás. *A* Naiv-PT *algoritmus bármely lépésében a P legdurvább stabil finomítása a Q-ban tárolt aktuális partíciónak finomítása.*

Bizonyítás. A bizonyítás indukcióval történik arra nézve, hogy hányszor hajtjuk végre a ciklusmagot. A $Q = P$ eset triviális. Tegyük fel, hogy egy S hasító alkalmazása előtt Q-ra igaz az állítás. Legyen R a P legdurvább stabil finomítása. Mivel S a Q blokkjaiból áll, és indukció miatt R finomítása Q-nak, ezért S az R valahány blokkjának egyesítése. Az R stabil minden blokkjára és tetszőleges számú blokkjának egyesítésére nézve is, így R stabil S-re nézve, azaz $R = \mathit{hasít}(S, R)$. Másrészt a hasítás monotonitását kihasználva $\mathit{hasít}(S, R)$ finomítása $\mathit{hasít}(S, Q)$-nak, vagyis a Q aktuális értékének. ■

31.23. állítás. *A* Naiv-PT *algoritmus a P egyértelműen létező legdurvább stabil finomítását adja meg, miközben a ciklust legfeljebb $(n-1)$-szer hajtja végre.*

Bizonyítás. Nyilvánvaló, hogy Q blokkjainak száma legalább 1 és legfeljebb n. A hasítás alkalmazásával Q-nak legalább egy blokkját kettévágjuk, azaz Q blokkjainak száma határozottan nő. Ebből következik, hogy az algoritmus legfeljebb $(n-1)$-szer hajtja végre a ciklust. Megálláskor Q stabil finomítása P-nek, és az előző állítás miatt a P legdurvább stabil finomítása Q-nak finomítása, ami csak úgy lehet, hogy Q megegyezik a P legdurvább stabil finomításával. ■

31.24. állítás. *Ha a V minden x elemére tároljuk az $E^{-1}(\{x\})$ halmazt, akkor a* Naiv-PT *algoritmus költsége legfeljebb $O(mn)$.*

Bizonyítás. Az algoritmus megvalósításához az általánosság megszorítása nélkül korlátozhatjuk magunkat arra az esetre, amikor a gráfban nincsenek nyelők, azaz minden csúcsból indul ki él. Vagyis a V tetszőleges x elemére teljesül, hogy $1 \leq |E(\{x\})|$. Legyen adott ugyanis egy P partíció. P minden B blokkját vágjuk ketté. A B' halmazba tartozzanak B azon csúcsai, amelyekből legalább egy él indul ki. Ekkor $B' = B \cap E^{-1}(V)$. Legyen $B'' = B \setminus E^{-1}(V)$, azaz a B-ben található nyelők halmaza. A B'' halmaz tetszőleges S-re nézve stabil, hiszen $B'' \cap E^{-1}(S) = \emptyset$, így B''-t az algoritmus során sosem kell széthasítani. Így P helyett elég a B' blokkokból álló P'-t venni, amely a $V' = E^{-1}(V)$ halmaz partíciója. Nyilvánvaló, hogy P' legdurvább stabil finomításához hozzávéve a B'' blokkokat, P legdurvább stabil finomítását kapjuk. Ez azt jelenti, hogy az algoritmust megelőzi egy előkészítés, melyben P'-t állítjuk elő, majd egy utófeldolgozás, amelyben a kapott legdurvább stabil finomításhoz hozzávesszük a B'' blokkokat. Az előkészítés és utófeldolgozás költségét a következőképpen becsülhetjük. V' elemszáma nyilván legfeljebb m. Ha a V minden x elemére rendelkezésünkre áll $E^{-1}(\{x\})$, akkor az előkészítés és utófeldolgozás $O(m+n)$ költséggel jár.

Ezentúl tehát feltesszük, hogy V tetszőleges x elemére teljesül, hogy $1 \leq |E(\{x\})|$, amiből az is következik, hogy $n \leq m$. Mivel tároljuk az $E^{-1}(\{x\})$ halmazokat, ezért a Q partíció blokkjai közül $O(m)$ költséggel találhatunk egy hasító blokkot. Az előző állítással együtt ez azt jelenti, hogy $O(mn)$ költséggel megvalósítható az algoritmus. ■

Az algoritmust hatékonyabban is végrehajthatjuk, ha ügyesebben keresünk hasító halmazokat. A javított algoritmus alapötlete, hogy a P partíción kívül két másik partíciót is kezelünk, a Q mellett egy olyan X partíciót is, amelyre minden lépésben igaz, hogy a Q az X finomítása, és Q stabil az X minden blokkjára nézve. Kiindulásként legyen $Q = P$ és X az a partíció, amely egyetlen blokkot tartalmaz, a V halmazt. Az algoritmus finomító lépését addig ismételjük, amíg végül $Q = X$ teljesül.

PT(V, E, P)

1 $Q \leftarrow P$
2 $X \leftarrow \{V\}$
3 **while** $X \neq Q$
4 **do** legyen S az X-nek egy olyan blokkja, amely Q-nak nem blokkja,
és legyen S-ben B a Q-nak egy olyan blokkja, amelyre $|B| \leq |S|/2$
5 $X \leftarrow (X \setminus \{S\}) \cup \{B, S \setminus B\}$
6 $Q \leftarrow \mathit{hasít}(S \setminus B, \mathit{hasít}(B, Q))$
7 **return** Q

31.25. állítás. *A* PT*-algoritmus és a* Naiv-PT *algoritmus eredménye megegyezik.*

Bizonyítás. Kezdetben teljesül, hogy Q stabil finomítása P-nek az X blokkjaira nézve. Az 5. lépésben mindig az X egyik blokkját vágjuk ketté, így az X finomítását kapjuk. A 6. lépésben a Q hasításokkal történő finomításával elérjük, hogy a Q stabil legyen az X két új blokkjára nézve. A stabilitás tulajdonságaira vonatkozó 31.21. állítás, illetve a Naiv-PT algoritmus helyessége alapján következik, hogy a PT-algoritmus is a P egyértelműen létező legdurvább stabil finomítását határozza meg. ■

Bizonyos esetekben a 6. lépés két hasításából az egyik elhagyható. Ehhez elégséges, hogy az E az x változó függvénye legyen.

31.26. állítás. *Ha V minden x elemére $|E(\{x\})| = 1$, akkor a* PT*-algoritmus 6. lépése a $Q \leftarrow \mathit{hasít}(B, Q)$ lépésre cserélhető.*

Bizonyítás. Tegyük fel, hogy Q stabil egy olyan S halmazra nézve, amely Q valahány blokkjának egyesítése. Legyen B a Q-nak egy olyan blokkja, amely S-nek részhalmaza. Elég belátni, hogy $\mathit{hasít}(B, Q)$ stabil az $(S \setminus B)$-re nézve. Ehhez legyen B_1 a $\mathit{hasít}(B, Q)$ egy blokkja. Mivel a B szerinti hasítás eredménye B-re nézve stabil partíció, így vagy az teljesül, hogy $B_1 \subseteq E^{-1}(B)$ vagy az, hogy $B_1 \subseteq E^{-1}(S) \setminus E^{-1}(B)$. Kihasználva az $|E(\{x\})| = 1$ feltételt, első esetben $B_1 \cap E^{-1}(S \setminus B) = \emptyset$, második esetben $B_1 \subseteq E^{-1}(S \setminus B)$ következik, ami pont azt jelenti, hogy $(S \setminus B)$-re stabil partíciót kaptunk. ■

Megjegyezzük, hogy általában abból, hogy egy partíció stabil S-re és B-re nézve, nem lehet arra következtetni, hogy $(S \setminus B)$-re nézve is stabil a partíció. Ha ez teljesül, akkor a méretek felezése miatt az algoritmus végrehajtási költségét javítani tudjuk, mert csak B szerinti hasításokra van szükség.

A 6. lépés két hasítása általános esetben egy blokkot elvileg négy részre is felbonthat. A következő állítás szerint a második hasítás az első hasítással kettévágott blokk egyik felét

már nem változtatja meg, azaz a két hasítás maximum három részre bontást eredményezhet. Ezt kihasználva általános esetben is növelhető az algoritmus hatékonysága.

31.27. állítás. *Legyen Q stabil partíció az S-re nézve, ahol S a Q valahány blokkjának egyesítése, és legyen B a Q-nak olyan blokkja, amely S-nek részhalmaza. Legyen D a Q-nak olyan blokkja, amelyet hasít(B, Q) valódi módon felbont D_1 és D_2 részekre, úgy hogy ezek egyike sem üres halmaz. Tegyük fel, hogy a D_1 blokkot a hasít$(S \setminus B,$ hasít$(B, Q))$ valódi módon tovább bontja D_{11} és D_{12} nem üres halmazokra. Ekkor a következők teljesülnek.*
1. $D_1 = D \cap E^{-1}(B)$ és $D_2 = D \setminus D_1$ akkor és csak akkor, ha $D \cap E^{-1}(B) \neq \emptyset$ és $D \setminus E^{-1}(B) \neq \emptyset$.
2. $D_{11} = D_1 \cap E^{-1}(S \setminus B)$ és $D_{12} = D_1 \setminus D_{11}$ akkor és csak akkor, ha $D_1 \cap E^{-1}(S \setminus B) \neq \emptyset$ és $D_1 \setminus E^{-1}(S \setminus B) \neq \emptyset$.
3. A D_2 blokkot a hasít$(S \setminus B,$ hasít$(B, Q))$ nem bontja tovább.
4. $D_{12} = D_1 \cap (E^{-1}(B) \setminus E^{-1}(S \setminus B))$.

Bizonyítás. Az első két állítás a hasítás definíciójából következik. A harmadik állítás bizonyításához tegyük fel, hogy D_2 valódi felbontással keletkezett D-ből. Ekkor $D \cap E^{-1}(B) \neq \emptyset$, továbbá $B \subseteq S$ miatt $D \cap E^{-1}(S) \neq \emptyset$ is teljesül. A Q partíció minden blokkja, így D is stabil az S-re nézve, amiből az következik, hogy $D \subseteq E^{-1}(S)$. Mivel $D_2 \subseteq D$, ezért az első állítás miatt $D_2 \subseteq E^{-1}(S) \setminus E^{-1}(B) = E^{-1}(S \setminus B)$, vagyis D_2 stabil az $S \setminus B$ halmazra nézve, így az $S \setminus B$ szerinti hasítás nem vágja ketté a D_2 blokkot. Végül a negyedik állítás abból következik, hogy $D_1 \subseteq E^{-1}(B)$, és $D_{12} = D_1 \setminus E^{-1}(S \setminus B)$. ■

Jelölje *számláló*(x, S) azt a számot, ahány S-beli csúcsot x-ből el lehet érni, azaz legyen *számláló*$(x, S) = |S \cap E(\{x\})|$. Vegyük észre, hogy ha $B \subseteq S$, akkor $E^{-1}(B) \setminus E^{-1}(S \setminus B) = \{x \in E^{-1}(B) \mid$ *számláló*$(x, B) =$ *számláló*$(x, S)\}$.

A méretek feleződése miatt a V halmaz tetszőleges x eleme legfeljebb $\lg n + 1$ olyan különböző B halmazban szerepelhet, amelyet a PT-algoritmus közben finomításra használtunk. A következőkben meg fogjuk adni a PT-algoritmusnak egy olyan végrehajtását, amelyben egy adott B blokk szerinti finomítás meghatározása az algoritmus 5. és 6. lépésében $O(|B| + \sum_{y \in B} |E^{-1}(\{y\})|)$ költséggel jár. Ha ezt összegezzük az algoritmus során felhasznált összes B blokkra és az ilyen blokkok összes elemére, akkor azt kapjuk, hogy a Hatékony-PT bonyolultsága legfeljebb $O(m \lg n)$. Ahhoz, hogy megadjunk egy ilyen megvalósítást, az adatok ábrázolásához ügyesen kell kiválasztanunk az adatszerkezeteket.

- Az E halmaz minden (x, y) éléhez láncoljuk hozzá az x csúcsot, valamint minden y csúcshoz láncoljuk hozzá az $\{(x, y) \mid (x, y) \in E\}$ listát. Ezáltal az $E^{-1}(\{y\})$ halmaz végigolvasásának költsége az $E^{-1}(\{y\})$ méretével lesz arányos.
- Legyen a Q partíció az X partíció finomítása. Mindkét partíció minden egyes blokkját egy-egy rekorddal ábrázoljuk. Az X partíció S blokkját ***egyszerűnek*** nevezzük, ha a Q egyetlen blokkjából áll, egyébként pedig ***összetettnek*** hívjuk.
- Legyen C az X partícióban szereplő összetett blokkok listája. Kezdetben legyen $C = \{V\}$, mivel V a P blokkjainak egyesítése. Ha P csak egy blokkból áll, akkor P megegyezik a saját legdurvább stabil finomításával, így nincs szükség további számításokra.
- Az X partíció bármely S blokkja esetén legyen Q-*blokkok*(S) a Q partíció azon blokkjainak duplán láncolt listája, amelyek egyesítése az S halmaz. Ezenkívül az $E^{-1}(S)$ halmaz minden x eleméhez tároljuk a *számláló*(x, S) értéket, amelyre egy mutató mutat minden olyan (x, y) élből, amelyben y az S halmaznak eleme. Kezdetben minden x

csúcshoz a *számláló*$(x, V) = |E(\{x\})|$ érték tartozik, és minden (x, y) élhez készítsünk egy olyan mutatót, amely a *számláló*(x, V) értékre mutat.

- A Q partíció bármely B blokkja esetén legyen X-*blokk*(B) az X partíciónak az a blokkja, amelyben B szerepel. Legyen továbbá *méret*(B) a B számossága, valamint *elemei*(B) a B elemeinek duplán láncolt listája. Minden elemhez tartozzon egy mutató, amely a Q azon blokkjára mutat, amelyben ez az elem szerepel. A duplán láncolás segítségével egy elemet $O(1)$ időben tudunk törölni.

A 31.24. állítás bizonyítása alapján az általánosság korlátozása nélkül feltehetjük, hogy $n \leq m$. Ekkor belátható, hogy a fenti adatszerkezetek elkészítéséhez szükséges tárméret $O(m)$.

HATÉKONY-PT(V, E, P)

```
 1 if |P| = 1
 2    then return P
 3 Q ← P
 4 X ← {V}
 5 C ← {V}                                   ▷ C az X összetett blokkjainak listája.
 6 while C ≠ ∅
 7       do legyen S a C egy eleme
 8          legyen B az S első két eleme közül a kisebbik méretű
 9          C ← C \ {S}
10          X ← (X \ {S}) ∪ {{B}, S \ {B}}
11          S ← S \ {B}
12          if |S| > 1
13             then C ← C ∪ {S}
14          Az E halmaz azon (x, y) éleit végigolvasva, melyekre y a B eleme,
            állítsuk elő az E^{-1}(B) halmazt és ennek minden x elemére
            számoljuk ki a számláló(x, B) értéket.
15          Az E^{-1}(B) halmazt végigolvasva a Q minden D blokkjához határozzuk
            meg a D_1 = D ∩ E^{-1}(B) és D_2 = D \ D_1 halmazokat.
16          Az E halmaz azon (x, y) éleit végigolvasva, melyekre y a B eleme,
            állítsuk elő az E^{-1}(B) \ E^{-1}(S \ B) halmazt a számláló(x, B) = számláló(x, S)
            feltétel ellenőrzésével.
17          Az E^{-1}(B) \ E^{-1}(S \ B) halmazt végigolvasva a Q minden D blokkjához
            határozzuk meg a D_12 = D_1 ∩ (E^{-1}(B) \ E^{-1}(S \ B))
            és D_11 = D_1 \ D_12 halmazokat.
18          for Q minden olyan D blokkjára, melyre D_11 ≠ ∅, D_12 ≠ ∅ és D_2 ≠ ∅
19             do if D az X egyszerű blokkja
20                   then C ← C ∪ {D}
21                Q ← (Q \ {D}) ∪ {D_11, D_12, D_2}
22          Az E halmaz azon (x, y) éleit végigolvasva, melyekre y a B eleme,
            számoljuk ki a számláló(x, S) értéket.
23 return Q
```

31.28. állítás. *A* HATÉKONY-PT *algoritmus a P legdurvább stabil finomítását határozza meg.*

Az algoritmus műveleti költsége legfeljebb $O(m \lg n)$, tárköltsége legfeljebb $O(m+n)$.

Bizonyítás. Az algoritmus helyessége a PT-algoritmus helyességéből, valamint a 31.27. állításból következik. A felhasznált adatszerkezetek miatt a ciklus magját alkotó lépéssorozat műveleti költsége arányos a megvizsgált élek számával és a finomításhoz használt B blokk elemszámával, ami összesen $O(|B| + \sum_{y \in B} |E^{-1}(\{y\})|)$. Összegezzük ezt az algoritmus során finomításra felhasznált összes B blokkra és az ilyen blokkok összes elemére. Mivel a B mérete legfeljebb az S méretének fele, így a V halmaz tetszőleges x eleme legfeljebb $\lg n + 1$ különböző B halmazban szerepelhet. Ebből következik, hogy az algoritmus teljes műveleti költsége $O(m \lg n)$. Könnyen igazolható, hogy $O(m+n)$ méretű tár elég az algoritmushoz felhasznált adatszerkezetek tárolására, valamint arra, hogy az adatszerkezeteket az algoritmus futása alatt karbantartsuk. ■

Megjegyezzük, hogy egyes lépések összevonásával tovább lehetne javítani az algoritmust, de ez csak egy konstans tényezővel csökkentené a műveleti költséget.

Legyen $G^{-1} = (V, E^{-1})$ az a gráf, amit G-ből úgy kapunk, hogy minden él irányítását megfordítjuk. Vegyünk egy 1-indexet, amelyet egy $\approx$ biszimuláció határoz meg a G gráfban. Legyen I, J a biszimuláció két osztálya, azaz az $I(G)$ két csúcsa. A biszimuláció definíciójából következik, hogy $J \subseteq E(I)$ vagy $E(I) \cap J = \emptyset$. Mivel $E(I) = (E^{-1})^{-1}(I)$, ez azt jelenti, hogy J stabil az I-re nézve a G^{-1} gráfban. Tehát a G legdurvább 1-indexe a legdurvább stabil finomítása a G^{-1} gráf alappartíciójának.

31.29. következmény. *A legdurvább 1-index meghatározható a* Hatékony-PT *algoritmussal. Az algoritmus műveleti költsége legfeljebb $O(m \lg n)$, tárköltsége legfeljebb $O(m+n)$.*

Gyakorlatok

31.4-1. Bizonyítsuk be a 29.21. állítást.

31.4-2. A P partíció méret-stabil egy S halmazra nézve, ha a P minden B blokkjának tetszőleges x, y eleme esetén $|E(\{x\}) \cap S| = |E(\{y\}) \cap S|$. Méret-stabil egy partíció, ha az összes blokkjára nézve méret-stabil. Bizonyítsuk be, hogy tetszőleges partíció legdurvább méret-stabil finomítása kiszámítható $O(m \lg n)$ időben.

31.4-3. Az 1-index minimális, ha semelyik két egyforma címkéjű I, J csúcs nem vonható össze, mert létezik olyan K csúcs, amelyre nézve $I \cup J$ nem stabil. Mutassunk példát arra, hogy a minimális 1-index nem egyértelmű, és így különbözik a legdurvább 1-indextől.

31.4-4. Bizonyítsuk be, hogy aciklikus gráf esetén a minimális 1-index egyértelmű és megegyezik a legdurvább 1-indexszel.

31.5. A(k)-indexek

1-indexek esetén az egy osztályba tartozó csúcsok ugyanazokra a gyökérből induló címkesorozatokra illeszkednek. Ez azt jelenti, hogy az egy osztályba tartozó csúcsok az őseik alapján nem különböztethetők meg. Enyhítsük ezt a feltételt úgy, hogy csak lokális megkülönböztethetetlenséget kívánunk meg, azaz a legfeljebb k-adik szintig felmenő ősök alapján nem lehet megkülönböztetni az egy osztályba tartozó csúcsokat. Ezzel az 1-indexnél durvább, kevesebb osztályból álló indexet kapunk, vagyis az index mérete csökken, ami

csökkenti a lekérdezések kiértékelésének költségét. Az 1-index biztonságos és pontos volt, amit szeretnénk megőrizni, mert ez garantálja, hogy az index alapján kiértékelt lekérdezések eredménye ugyanaz, mintha az eredeti gráfon értékeltük volna ki a lekérdezést. Az A(k)-index esetén a biztonságosság érvényben marad, de a pontosság nem teljesül, ezért ezt a kiértékelő algoritmus módosításával kell elérnünk.

31.30. definíció. *$A \approx^k$* ***k-biszimuláció*** *a gráf V csúcsain az alábbi rekurzív definícióval meghatározott ekvivalencia reláció.*
i) $u \approx^0 v$ akkor és csak akkor, ha címke(u) = címke(v),
ii) $u \approx^k v$ akkor és csak akkor, ha $u \approx^{k-1} v$ és ha egy u' csúcsból vezet él az u csúcsba, akkor van olyan v' csúcs, amelyből vezet él a v csúcsba és $u' \approx^{k-1} v'$, illetve ez fordítva is igaz, azaz ha egy v' csúcsból vezet él az v csúcsba, akkor van olyan u' csúcs, amelyből vezet él a u csúcsba és $u' \approx^{k-1} v'$.
$u \approx^k v$ esetén azt mondjuk, hogy u és v ***k-biszimuláns****. Az* ***A(k)-indexhez*** *tartozó partíció osztályait a k-biszimuláció ekvivalenciaosztályai alkotják.*

Az elnevezésben az „A” az „approximatív” (közelítő) szóra utal.

Vegyük észre, hogy a $k = 0$ értékhez az alappartíció tartozik, és k értékének növelésével ezt finomítjuk, míg el nem jutunk a legdurvább 1-indexig.

Jelöljük $L(u,k,G)$-vel azokat a legfeljebb k hosszú címkesorozatokat, amelyekre u illeszkedik a G gráfban. Könnyen ellenőrizhetők az A(k)-index alábbi tulajdonságai.

31.31. állítás.
1. Ha u és v k-biszimuláns, akkor $L(u,k,G) = L(v,k,G)$.
2. Ha I az $A(k)$-index egy csúcsa és $u \in I$, akkor $L(I,k,I(G)) = L(u,k,G)$.
3. Az $A(k)$-index pontos a legfeljebb k hosszú egyszerű kifejezések esetén.
4. Az $A(k)$-index biztonságos.
5. A $(k+1)$-biszimuláció a k-biszimuláció (nem feltétlen valódi) finomítása.

Az A(k)-index a csúcsok k távolságú, gyökér felöli félkörnyezeteit hasonlítja össze, így az ezen kívül eső módosítások nem befolyásolják a csúcsok ekvivalenciáját, ahogy ezt az alábbi, könnyen belátható állítás mutatja.

31.32. állítás. *Tegyük fel, hogy a v csúcsból az x-be és y-ba vezető legrövidebb utak egyaránt k-nál több élből állnak. Ekkor egy u csúcsból v-be vezető él hozzáadása vagy törlése nem változtatja meg, hogy x és y k-biszimuláns vagy nem.*

Az A(k)-index készítéséhez a PT-algoritmus módosított változatát használjuk. Általánosan vizsgálhatjuk a legdurvább stabil finomítás közelítési problémáját.

31.33. definíció. *Legyen G=(V,E) irányított gráfban P a V partíciója. Legyen $P_0, P_1, \ldots, P_k$ partíciókból álló olyan sorozat, melyre $P_0 = P$, és P_{i+1} a P_i legdurvább olyan finomítása, amely stabil P_i-re nézve. Azt mondjuk, hogy a P_k partíció a* ***P legdurvább stabil finomításának k lépéses közelítése.***

Vegyük észre, hogy a P_i sorozat bármelyik tagja a P finomítása, és ha $P_k = P_{k-1}$, akkor P_k a P legdurvább stabil finomítása. Könnyen ellenőrizhető, hogy a PT-algoritmushoz hasonlóan a P legdurvább stabil finomításának tetszőleges közelítését mohó módon számol-

hatjuk ki, azaz, ha P_i valamelyik B blokkja nem stabil a P_{i-1} valamelyik S blokkjára nézve, akkor S-sel hasítsuk szét a B blokkot, azaz P_i helyett vegyük a *hasít*(S, P_i) partíciót.

NAIV-KÖZELÍTÉS(V, E, P, k)

```
1  P_0 ← P
2  for i ← 1 to k
3      do P_i ← P_{i-1}
4          for minden S ∈ P_{i-1}, amelyre hasít(S, P_i) ≠ P_i
5              do P_i ← hasít(S, P_i)
6  return P_k
```

Megjegyezzük, hogy a NAIV-KÖZELÍTÉS algoritmust is lehetne javítani a PT-algoritmushoz hasonlóan.

Az A(k)-index kiszámításához használhatjuk a NAIV-KÖZELÍTÉS algoritmust, csak azt kell észrevenni, hogy definíció alapján az A(k)-indexhez tartozó partíció a G^{-1} gráfban stabil az A$(k-1)$-indexhez tartozó partícióra nézve. Belátható, hogy az így előállított A(k)-index készítésének műveleti költsége $O(km)$, ahol m a G gráf éleinek száma.

A(k)-INDEXES-KIÉRTÉKELÉS(G, A_R, k)

```
1  legyen I_k a G A(k)-indexe
2  Q ← INDEXES-KIÉRTÉKELÉS(G, I_k, A_R)
3  for minden u ∈ Q
4      do if L(u) ∩ L(A_R) = ∅
5             then Q ← Q \ {u}
6  return Q
```

Az A(k)-index biztonságos, de csak a legfeljebb k hosszú egyszerű kifejezésekre nézve pontos, ezért az 4. lépésben a Q halmaz minden u elemére ellenőrizni kell, hogy tényleg kielégíti az R lekérdezést, és elhagyjuk az eredményből azokat, amelyek nem illeszkednek az R lekérdezésre. Azt, hogy egy adott csúcs kielégíti-e az R kifejezést, egy véges nem determinisztikus automatával lehet eldönteni a 31.6. állításban leírtakhoz hasonlóan, annyi változtatással, hogy az automatát most fordított irányban kell működtetni. Az ilyen ellenőrzések számát csökkenteni lehet a következő állítás alapján, melynek bizonyítását az Olvasóra bízzuk.

31.34. állítás. *Tegyük fel, hogy az A(k)-indexnek megfelelő I_k gráfban az I indexcsúcs illeszkedik egy olyan címkesorozatra, amelynek a végződése $s = l_0, \ldots, l_p$, $p \leq k-1$. Ha a G gráfban minden gyökérből induló, s'.s alakú címkesorozat kielégíti az R kifejezést, akkor az I minden eleme kielégíti az R kifejezést.*

Gyakorlatok

31.5-1. Jelölje $A_k(G)$ a G A(k)-indexét. Igaz-e, hogy $A_k(A_l(G)) = A_{k+l}(G)$?

31.5-2. Bizonyítsuk be a 31.31. állítást.

31.5-3. Bizonyítsuk be a 31.32. állítást.

31.5-4. Bizonyítsuk be a 31.34. állítást.

31.5-5. Igazoljuk, hogy a NAIV-KÖZELÍTÉS algoritmus valóban a legdurvább k lépéses stabil közelítést állítja elő.
31.5-6. Legyenek az $A = \{A_0, A_1, \ldots, A_k\}$ indexekből álló halmaz elemei rendre A(0)-, A(1)-, ..., A(k)-indexek. Az A *minimális,* ha az A_i bármelyik két elemét egyesítve az A_i nem marad stabil A_{i-1}-re nézve, $1 \leq i \leq k$. Bizonyítsuk be, hogy tetszőleges gráf esetén egyértelműen létezik olyan minimális A, melynek elemei legdurvább A(i)-indexek, $0 \leq i \leq k$.

31.6. D(k)- és M(k)-indexek

Az A(k)-index használata esetén a k értékét jól kell megválasztani. Túl nagy k esetén nagy lesz az index mérete, túl kicsi érték esetén a pontosság megőrzése céljából túl sokszor kell a talált megoldást ellenőrizni. Az egy osztályba tartozó csúcsok lokálisan hasonlóak, azaz a k távolságú környezetük, pontosabban a legfeljebb k hosszú hozzájuk vezető utak alapján nem különböztethetők meg. Minden csúcsra ugyanazt a k értéket használjuk, annak ellenére, hogy vannak csúcsok, amelyek kevésbé fontosak, mint más csúcsok. Például egyes csúcsok a gyakorlatban nagyon ritkán szerepelnek lekérdezések eredményében, és csak a rajtuk keresztülhaladó utak címkesorozatait vizsgáljuk meg. A kevésbé fontos csúcsokra nem érdemes nagyobb finomítást használni. Ez adja az ötletet a dinamikus D(k)-index használatához, amely különböző k értékeket rendel a csúcsokhoz, a lekérdezésektől függően. Feltételezzük, hogy adott a lekérdezéseknek egy halmaza. Ha például ezek között a lekérdezések között szerepel egy $R.a.b$ és egy $R'.a.b.c$ lekérdezés, ahol R, R' reguláris lekérdezés, akkor a b címkéjű csúcsok esetén legalább 1-biszimuláció, a c címkéjű csúcsok esetén legalább 2-biszimuláció szerinti felbontásra van szükség.

31.35. definíció. *Legyen $I(G)$ a G gráfhoz tartozó indexgráf, és minden I indexcsúcshoz tartozzon egy $k(I)$ nem negatív egész szám. Tegyük fel, hogy az I blokkba tartozó csúcsok $k(I)$-biszimulánsak. A $k(I)$ értékek elégítsék ki a következő feltételt: ha az $I(G)$ gráfban I-ből J-be vezet él, akkor $k(I) \geq k(J) - 1$. Az ilyen tulajdonságú $I(G)$ indexet* ***D*(k)*-indexnek*** *hívjuk.*

Az elnevezésben a „D” a „dinamikus” szóra utal. Vegyük észre, hogy az A(k)-index a D(k)-index speciális esete, mivel A(k)-index esetén bármely indexcsúcshoz tartozó elemek pontosan k-biszimulánsak. Mivel a címkék alapján történő osztályozás, vagyis az alappartíció A(0)-index, illetve az 1-index véges gráf esetén megegyezik valamilyen k-ra egy A(k)-indexszel, azért ezek is a D(k)-index speciális esetei. A D(k)-index, mint minden index, biztonságos, így elég rajta kiértékelni a lekérdezéseket. A pontosság biztosítására a válaszokat ellenőrizni kell. A következő állítás arról szól, hogy a lekérdezések egy részére a pontosság eleve garantált, így az ilyen lekérdezések esetén az ellenőrző lépés elhagyható.

31.36. állítás. *Legyen I_1, I_2, ..., I_s egy irányított út a D(k)-indexben, és tegyük fel, hogy $k(I_j) \geq j - 1$, ha $1 \leq j \leq s$. Ekkor címke(I_1), címke(I_2), ..., címke(I_s) címkesorozatra az I_s minden egyes eleme illeszkedik.*

Bizonyítás. A bizonyítás s-re vonatkozó indukción alapul. Az $s = 1$ eset triviális. Az indukciós feltevés miatt címke(I_1), címke(I_2), ..., címke(I_{s-1}) címkesorozatra az I_{s-1} minden

egyes eleme illeszkedik. Mivel az $I(G)$ gráfban az I_{s-1} csúcsból vezet él az I_s csúcsba, ezért van olyan $u \in I_s$ és $v \in I_{s-1}$, hogy G-ben vezet él v-ből u-ba. Ez azt jelenti, hogy u illeszkedik az $s-1$ hosszú címke(I_1), címke(I_2), ..., címke(I_s) címkesorozatra. Az I_s elemei legalább $(s-1)$-biszimulánsak, ezért I_s minden egyes eleme illeszkedik erre a címkesorozatra. ■

31.37. következmény. *A $D(k)$-index pontos egy $l_0, \ldots, l_m$ címkesorozatra nézve, ha az indexgráfban minden erre illeszkedő I csúcs esetén $k(I) \geq m$,*

A D(k)-index készítése során az alappartíciót, vagyis az A(0)-indexet fogjuk finomítani. A lekérdezések alapján kiinduló értékeket rendelünk az azonos címkéjű csúcsokat tartalmazó osztályokhoz. Tegyük fel, hogy összesen t különböző értéket használunk. Álljon a K_0 halmaz ezekből az értékekből, és legyenek a K_0 elemei $k_1 > k_2 > \ldots > k_t$. Ha a K_0 elemei nem elégítik ki a D(k)-indexben megadott feltételt, akkor a Súlymódosító algoritmussal a legnagyobb értékből kiindulva növeljük őket úgy, hogy kielégítsék a feltételt. Így az azonos címkéjű csúcsokhoz tartozó osztályok már megfelelő k értékkel fognak rendelkezni. Ezután az osztályokat hasításokkal elkezdjük finomítani, hogy az osztályhoz tartozó elemek k-biszimulánsak legyenek, és a hasítás minden tagjához ezt a k értéket rendeljük. Az eljárás közben a finomítással kapott partíciónak megfelelően az indexgráf éleit is fel kell frissíteni.

Súlymódosító(G, K_0)

```
1  K ← ∅
2  K_1 ← K_0
3  while K_1 ≠ ∅
4        do for minden I, ahol I az A(0)-index csúcsa és k(I) = max(K_1)
5              do for minden J, ahol J az A(0)-index csúcsa és J-ből vezet él I-be
6                    k(J) ← max(k(J), max(K_1) − 1)
7        K ← K ∪ {max(K_1)}
8        K_1 ← {k(A) | A az A(0)-index csúcsa} \ K
9  return K
```

Könnyen ellenőrizhető, hogy a Súlymódosító algoritmus műveleti költsége $O(m)$, ahol m az A(0)-index éleinek száma.

D(k)-index-készítő(G, K_0)

```
1  legyen I(G) a G gráfhoz tartozó A(0)-index, V_I az I(G) csúcsai,
   E_I az I(G) éleinek halmaza
2  K ← Súlymódosító(G, K_0)        ▷ A kezdeti súlyok módosítása.
                                     a D(k)-index feltétele szerint.
```

```
3  for k ← 1 to max(K)
4      do for minden I ∈ V_I
5          do if k(I) ≥ k
6              then for minden J, ahol (J, I) ∈ E_I
7                  do V_I ← (V_I \ {I}) ∪ {I ∩ E(J), I \ E(J)}
8                     k(I ∩ E(J)) ← k(I)
9                     k(I \ E(J)) ← k(I)
10                    E_I ← {(A, B) | A, B ∈ V_I, ∃ a ∈ A, ∃ b ∈ B, (a, b) ∈ E}
11                    I(G) ← (V_I, E_I)
12 return I(G)
```

Az algoritmus 7. lépésében egy hasítás műveletet végzünk. Ezzel elérjük, hogy a $(k-1)$-biszimuláció szerint ekvivalens elemeket tartalmazó osztályokat k-biszimuláció szerinti ekvivalencia osztályokra hasítjuk szét. A D(k)-INDEX-KÉSZÍTŐ algoritmusról belátható, hogy a műveleti költsége legrosszabb esetben $O(km)$, ahol m a G gráf éleinek száma, k pedig $max(K_0)$ értékkel egyenlő.

Előfordulhat, hogy a D(k)-index túlságosan finom felosztást eredményez, és nagy mérete miatt nem eléggé hatékony a használata. A túlfinomítás a következő okokra vezethető vissza. A D(k)-INDEX-KÉSZÍTŐ algoritmus az azonos címkéjű csúcsokhoz ugyanazt a k értéket rendeli, holott lehet, hogy ezeknek a csúcsoknak egy része a lekérdezések szempontjából kevésbé fontos, vagy gyakrabban fordul elő k-nál jóval rövidebb lekérdezések eredményében, ezért ezekre a csúcsokra kisebb finomság is elegendő lenne. A SÚLYMÓDOSÍTÓ algoritmus a csúcshoz tartozó k érték alapján a szülő értékét nem fogja csökkenteni, ha az nagyobb, mint $k-1$. Így ha ezek a szülők a gyakori lekérdezések alapján nem túl fontos csúcsok, akkor emiatt túlfinomítás fordulhat elő. A túlfinomítás elkerülésére vezetjük be az M(k)-indexet és az M*(k)-indexet, ahol az „M” a „mixed” („vegyes”) szóra utal, a „*” pedig arra, hogy nem egy indexet adunk meg, hanem fokozatosan finomodó indexekből álló véges hierarchiát. Az M(k)-index egy olyan D(k)-index, amelynek előállításához megadott algoritmus az egyforma címkéjű csúcsokat nem feltétlen sorolja ugyanolyan k-biszimulancia osztályokba.

Először vizsgáljuk meg, hogy egy $I(G) = (V_I, E_I)$ D(k)-indexet hogyan kell módosítani, ha az egyik I indexcsúcshoz tartozó kiindulási k_I súlyt megnöveljük. Ha $k(I) \geq k_I$, akkor az $I(G)$ változatlanul marad. Ellenkező esetben ahhoz, hogy a D(k)-index súlyokra vonatkozó feltételei teljesüljenek, az I őseihez tartozó súlyokat rekurzívan növelnünk kell, amíg a szülők már legalább $k_I - 1$ súllyal rendelkeznek. Végül az I szülők szerinti széthasításával a szülők súlyainál eggyel nagyobb, tehát legalább k_I lesz a kapott indexcsúcsok finomsága, azaz a hozzájuk tartozó elemek legalább k_I-biszimulánsak lesznek. Mindezt a következő SÚLYNÖVELŐ algoritmussal adjuk meg.

SÚLYNÖVELŐ($I, k_I, I(G)$)

```
1  if k(I) ≥ k_I
2     then return I(G)
3  for minden (J, I) ∈ E_I
4     do I(G) ← SÚLYNÖVELŐ(J, k_I − 1, I(G))
```

5 **for** minden $(J, I) \in E_I$
6 **do** $V_I \leftarrow (V_I \setminus \{I\}) \cup \{I \cap E(J), I \setminus E(J)\}$
7 $E_I \leftarrow \{(A, B) \mid A, B \in V_I, \exists\, a \in A, \exists\, b \in B, (a, b) \in E\}$
8 $I(G) \leftarrow (V_I, E_I)$
9 **return** $I(G)$

Könnyen ellenőrizhető az alábbi állítás, amelynek segítségével a későbbiekben egy lépésben tudjuk majd a megfelelő finomságot elérni, azaz nem kell egyesével növelni a finomságokat.

31.38. állítás. *$u \approx^k v$ akkor és csak akkor, ha $u \approx^0 v$ és ha egy u' csúcsból vezet él az u csúcsba, akkor van olyan v' csúcs, amelyből vezet él a v csúcsba és $u' \approx^{k-1} v'$, illetve ez fordítva is igaz, azaz ha egy v' csúcsból vezet él a v csúcsba, akkor van olyan u' csúcs, amelyből vezet él az u csúcsba és $u' \approx^{k-1} v'$.*

Jelölje *GYAK* a gyakori reguláris lekérdezések által meghatározott egyszerű kifejezések, azaz címkesorozatok halmazát. Azt akarjuk elérni, hogy az index annyira legyen csak finom, amennyi ahhoz kell, hogy a *GYAK* halmazhoz tartozó lekérdezésekre pontos legyen. Ehhez meghatározzuk a lényeges csúcsokat, és a D(k)-INDEX-KÉSZÍTŐ algoritmust úgy módosítjuk, hogy a finomítást növelő széthasításkor mindig elhagyjuk a nem lényeges csúcsokat, illetve a nem lényeges csúcsok őseit.

Legyen $R \in GYAK$ egy gyakori egyszerű lekérdezés. Az R-re illeszkedő csúcsok halmazát az indexgráfban S, az adatgráfban T jelölje, azaz $S = R(I(G))$ és $T = R(G)$. Az $I(G)$ indexgráfban egy I indexcsúcs finomságát jelölje $k(I)$, vagyis az I-hez tartozó csúcsok maximálisan $k(I)$-biszimulánsak.

FINOMÍT(R, S, T)

1 **for** minden $I \in S$
2 **do** $I(G) \leftarrow$ INDEXCSÚCSOT-FINOMÍT(I, hossz(R), $I \cap T$)
3 **while** $\exists\, I \in V_I$, melyre $k(I) <$ hossz(R) és I illeszkedik R-re
4 **do** $I(G) \leftarrow$ SÚLYNÖVELŐ(I, hossz(R), $I(G)$)
5 **return** $I(G)$

Az indexcsúcsok finomítását a következő algoritmussal adjuk meg. Először az I indexcsúcs lényeges szüleit finomítjuk rekurzívan. Ezután I-t a lényeges szülők alapján szétvágjuk úgy, hogy a kapott új részek finomsága k-val legyen egyenlő. A H halmazba kerülnek be az I szétvágott részei. Végül ezek közül összevonjuk azokat, amelyek nem tartalmaznak lényeges csúcsot, és az összevont halmaznál megtartjuk az I eredeti finomságát.

INDEXCSÚCSOT-FINOMÍT(I, k, *lényeges-csúcsok*)

```
1  if k(I) ≥ k
2     then return I(G)
3  for minden (J, I) ∈ E_I
4     do lényeges-szülők ← E^{-1}(lényeges-csúcsok) ∩ J
5        if lényeges-szülők ≠ ∅
6           then INDEXCSÚCSOT-FINOMÍT(J, k − 1, lényeges-szülők)
7  k-előző ← k(I)
8  H ← {I}
9  for minden (J, I) ∈ E_I
10    do if E^{-1}(lényeges-csúcsok) ∩ J ≠ ∅
11          then for minden F ∈ H
12                  do V_I ← (V_I \ {F}) ∪ {F ∩ E(J), F \ E(J)}
13                     E_I ← {(A, B) | A, B ∈ V_I, ∃ a ∈ A, ∃ b ∈ B, (a, b) ∈ E}
14                     k(F ∩ E(J)) ← k
15                     k(F \ E(J)) ← k
16                     I(G) ← (V_I, E_I)
17                     H ← (H \ {F}) ∪ {F ∩ E(J), F \ E(J)}
18 maradék ← ∅
19 for minden F ∈ H
20    do if lényeges-csúcsok ∩ F = ∅
21          then maradék ← maradék ∪ F
22               V_I ← (V_I \ {F})
23 V_I ← V_I ∪ {maradék}
24 E_I ← {(A, B) | A, B ∈ V_I, ∃ a ∈ A, ∃ b ∈ B, (a, b) ∈ E}
25 k(maradék) ← k-előző
26 I(G) ← (V_I, E_I)
27 return I(G)
```

A FINOMÍT algoritmus egy gyakori egyszerű kifejezés alapján finomítja az $I(G)$ index gráfot úgy, hogy egy indexcsúcsot nem feltétlen azonos finomságú részekre bont fel, és ezzel a túlfinomítást elkerüli. Ha az A(0)-indexből indulunk ki, és egymás után minden gyakori lekérdezésre elkészítjük a finomítást, akkor a gyakori lekérdezésekre nézve pontos indexgráfot kapunk. Ezt nevezzük M(k)-indexnek. A gyakori lekérdezések *GYAK* halmaza az idők folyamán változhat, ezért az indexet is dinamikusan módosítani kell.

31.39. definíció. *Az* ***M*(k)*-index*** *olyan $D(k)$-index, amelyet az alábbi* M(k)-INDEX-KÉSZÍTŐ *algoritmussal állítunk elő.*

M(k)-INDEX-KÉSZÍTŐ(G, *GYAK*)

```
1  I(G) ← a G gráfhoz tartozó A(0)-index
2  V_I ← az I(G) csúcsai
3  for minden I ∈ V_I
4     do k(I) ← 0
```

```
5  E_I ← az I(G) éleinek halmaza
6  for minden R ∈ GYAK
7      do I(G) ← FINOMÍT(R, R(I(G)), R(G))
8  return I(G)
```

Az M(k)-index a gyakori kifejezésekre nézve pontos index. Egy nem gyakori lekérdezés esetén a következőképpen járunk el. Az M(k)-index egyben D(k)-index is, ezért ha az $I(G)$ indexgráfban egy indexcsúcs illeszkedik egy R egyszerű lekérdezésre, és az indexcsúcs finomsága legalább akkora, mint az R hossza, akkor az indexcsúcs minden eleme illeszkedik az R lekérdezésre a G gráfban. Ha kisebb az indexcsúcs finomsága, akkor minden eleméről a NAIV-KIÉRTÉKELÉS alapján ellenőrizni kell, hogy megoldás-e a G gráfban.

Az M(k)-index használata esetén akkor a legkisebb a túlfinomítás, ha a gyakori egyszerű lekérdezések hossza megegyezik. Ha nagy eltérések fordulhatnak elő a gyakori lekérdezések hosszában, akkor előfordulhat, hogy a rövid lekérdezések számára túlságosan finom indexet használunk. Készítsük el azt a fokozatosan finomodó indexsorozatot, amellyel az A(0)-indexből indulva eljutunk az M(k)-indexhez úgy, hogy egy lépésben egy indexcsúcsot legfeljebb eggyel nagyobb finomságú részekre bontunk fel. Ha a teljes indexsorozatot ismerjük, akkor egy egyszerű lekérdezés kiértékeléséhez nem kell a legfinomabb, és ezért legnagyobb indexet használni, hanem csak a lekérdezés hosszának megfelelő finomságú indexet.

31.40. definíció. *Az* ***M***(k)**-index** *olyan I_0, I_1, ... I_k indexsorozat, amelyre a következő tulajdonságok teljesülnek:*

1. *Minden I_i index M(i)-index, ahol $i = 0, 1, \ldots, k$.*
2. *Az I_i indexben minden indexcsúcs finomsága legfeljebb i, ahol $i = 0, 1, \ldots, k$.*
3. *Az I_{i+1} az I_i finomítása, ahol $i = 0, 1, \ldots, k-1$.*
4. *Ha az I_i index J csúcsát az I_{i+1} indexben felbontjuk, és a felbontás egyik halmaza J', azaz $J' \subseteq J$, akkor $k(J) \leq k(J') \leq k(J) + 1$.*
5. *Legyen J az I_i index csúcsa, és $k(J) < i$. Ekkor $i < i'$ esetén, az $I_{i'}$ minden olyan J' indexcsúcsára, amelyre $J' \subseteq J$, teljesül, hogy $k(J) = k(J')$.*

A definícióból következik, hogy M*(k)-index esetén I_0 az A(0)-indexszel egyezik meg. Az utolsó tulajdonság azt mondja ki, hogy ha valamelyik indexcsúcs finomítása megáll, akkor már később sem fog változni a finomsága. Az M*(k)-index rendelkezik az M(k)-index jó tulajdonságaival, a felépítése is hasonló, azaz a gyakori lekérdezések alapján szükség esetén tovább finomítjuk az indexet úgy, hogy a gyakori lekérdezésekre pontos legyen, de most az egymás utáni durvább indexeket is megtartjuk és frissítjük, nem csak a legfinomabbat.

Az M*(k)-index ábrázolásánál ki lehet használni, hogy ha egy indexcsúcsot már nem bontunk tovább, akkor onnan kezdve nem kell minden későbbi indexben tárolni ezt a csúcsot, hanem elég hivatkozni rá. Hasonlóan az ilyen csúcsok közti éleket sem kell ismételten ábrázolni a indexsorozatban, hanem elég hivatkozni rájuk. Az M*(k)-index készítése az M(k)-INDEX-KÉSZÍTŐ algoritmushoz hasonló módon történhet. Az algoritmus részletes leírása

He és Yang cikkében található meg.

Az M*(k)-index segítségével többféle stratégiát is alkalmazhatunk a lekérdezések kiértékelésére. Legyen R egy gyakori egyszerű lekérdezés.

A legegyszerűbb stratégia, hogy a lekérdezés hosszával megegyező finomságú indexet használjuk.

M*(k)-INDEXES-NAIV-KIÉRTÉKELÉS(G, $GYAK$, R)

```
1  {I_0, I_1 ,..., I_k} ← a G gráfhoz tartozó M*(k)-index
2  h ← hossz(R)
3  return INDEXES-KIÉRTÉKELÉS(G, I_h, A_R)
```

A kiértékelés úgy is történhet, hogy a lekérdezés egyre hosszabb előtagjait értékeljük ki az előtag hosszával megegyező sorszámú index alapján. Az előtag kiértékeléséhez az eggyel rövidebb előtag kiértékelésénél megtalált csúcsok felbontásait vesszük a következő indexben, és ezekből indulva keresünk olyan éleket, amelyeknek a következő szimbólum a címkéje. Legyen $R = l_0, l_1, \ldots, l_h$ egyszerű gyakori lekérdezés, azaz $hossz(R) = h$.

M*(k)-INDEXES-FELÜLRŐL-LEFELÉ-KIÉRTÉKELÉS(G, $GYAK$, R)

```
1  {I_0, I_1, ..., I_k} ← a G gráfhoz tartozó M*(k)-index
2  R_0 ← l_0
3  H_0 ← ∅
4  for minden C ∈ E_{I_0}(gyökér(I_0))          ▷ A gyökér gyerekei az I_0 gráfban.
5      do if címke(C) = l_0
6           then H_0 ← H_0 ∪ {C}
7  for j ← 1 to hossz(R)
8      do H_j ← ∅
9         R_j ← R_{j-1}.l_j
10        H_{j-1} ← M*(k)-INDEXES-FELÜLRŐL-LEFELÉ-KIÉRTÉKELÉS(G,GYAK, R_{j-1})
11        for minden A ∈ H_{j-1}                ▷ Az A csúcs az I_{j-1} gráfnak csúcsa.
12            do if A = ∪B_m, ahol B_m ∈ V_{I_j}   ▷ Az A csúcs felbontása az I_j gráfban.
13                 then for minden B_m
14                          do for minden C ∈ E_{I_j}(B_m)   ▷ B_m minden gyerekére
                                                              az I_j gráfban.
15                             do if címke(C) = l_j
16                                  then H_j ← H_j ∪ {C}
17 return H_h
```

Az is lehet a stratégiánk, hogy először keresünk egy olyan részsorozatot az egyszerű lekérdezésnek megfelelő címkesorozatban, amelyre kevés csúcs illeszkedik, vagyis nagy a szelektivitása. A részsorozat hosszához tartozó indexben megkeressük az illeszkedő csúcsokat, majd az indexsorozat alapján megnézzük, hogy milyen csúcsokra bontjuk fel ezeket a csúcsokat a lekérdezés hosszához tartozó finomabb indexben. Végül ezekből a csúcsokból indulva megnézzük, hogy milyen csúcsok illeszkednek az eredeti lekérdezés maradék részére. Az ehhez a módszerhez tartozó M*(k)-INDEXES-ELŐSZŰRT-KIÉRTÉKELÉS algoritmus rész-

letes kidolgozását az Olvasóra bízzuk.

Gyakorlatok

31.6-1. Dolgozzuk ki részletesen az M$^*(k)$-INDEXES-ELŐSZŰRT-KIÉRTÉKELÉS algoritmust. Mi lesz az algoritmus költsége?

31.6-2. Bizonyítsuk be a 31.38. állítást.

31.6-3. Igazoljuk, hogy a SÚLYMÓDOSÍTÓ algoritmus műveletei költsége $O(m)$, ahol m az A(0)-index éleinek száma.

31.7. Elágazó lekérdezések

Reguláris lekérdezések segítségével a gráfból azokat a csúcsokat tudjuk kiválasztani, amelyekhez a gyökérből vezet egy olyan út, amelyen a címkék egy előre megadott reguláris mintára illeszkednek. Természetesnek tűnő általánosítás, hogy a csúcshoz vezető úton elhelyezkedő csúcsokra további feltételeket adjunk meg. Például kiköthetjük, hogy egy adott címkéjű csúcsból egy másik címkesorozattal legyen elérhető egy másik csúcs. Megfogalmazhatunk olyan feltételeket is, hogy egy adott címkéjű csúcshoz egy másik csúcsból valamilyen megadott címkesorozatú út vezessen. Ezekből a feltételekből többet is vehetünk, tagadásukat is használhatjuk, egymásba is ágyazhatjuk őket. A feltételek ellenőrzéséhez nemcsak előre kell lépegetni az élek irányításnak megfelelően, hanem időnként visszafele is. A következőkben megadjuk az elágazó lekérdezések nyelvének leírását, és bevezetjük az előre-hátra (forward-backward) indexeket. A összes elágazó lekérdezésre nézve biztonságos és pontos előre-hátra indexet FB-indexnek nevezzük. Az 1-indexhez hasonlóan ez általában túlságosan nagy méretű, ezért ehelyett inkább olyan FB(f, b, d)-indexet használunk, amely akkor lesz pontos, ha az elágazó lekérdezésben előre egyszerre legfeljebb f hosszan megyünk, visszafele legfeljebb b hosszan, és a feltételek egymásba ágyazása valójában legfeljebb d mélységű. A gyakorlatban előforduló esetekben az f, b és d értéke általában kicsi. Azokra a lekérdezésekre, amelyeknél valamelyik paraméter értéke nagyobb, mint az indexben szereplő megfelelő érték, egy ellenőrző lépést is be kell iktatni, vagyis az indexen kiértékeljük a lekérdezést, és a kapott indexcsúcsokban szereplő csúcsok közül csak azokat tartjuk meg, amelyek szintén kielégítik a lekérdezést.

Ha az n csúcsból irányított él vezet az m csúcsba, akkor ezt n/m vagy $m\backslash n$ jelöléssel adhatjuk meg. Ha az n csúcsból irányított úttal elérhető az m csúcs, akkor ezt $n//m$ vagy $m\backslash\backslash n$ formában jelöljük. (Eddig / helyett a . jelet használtuk, így // az _*, vagy röviden a * reguláris kifejezésnek felel meg.)

Címkesorozaton ezentúl olyan sorozatot értünk, amelyben elválasztójelek az előre-jelek (/, //) és a hátra-jelek (\, \\) lehetnek. Egy csúcssorozat illeszkedik egy címkesorozatra, ha az egymás utáni csúcsok rendre olyan viszonyban állnak egymással, amilyet a megfelelő elválasztójel meghatároz, és a csúcsok címkéi a címkesorozatban felsorolt címkék szerint követik egymást.

Előre-címkesorozatban csak előre-jelek, ***hátra-címkesorozatban*** csak hátra-jelek találhatók.

Az ***elágazó lekérdezéseket*** a következő nyelvtan definiálja.

elágazó_lekérdezés	::=	előre_címkesorozat [vagy_kifejezés] előre_jel elágazókifejezés
		| előre_címkesorozat [vagy_kifejezés]
		| előre_címkesorozat
vagy_kifejezés	::=	és_kifejezés or vagy_kifejezés
		| és_kifejezés
és_kifejezés	::=	elágazó_feltétel and és_kifejezés
		| nem_elágazó_feltétel and és_kifejezés
		| elágazó_feltétel
		| nem_elágazó_feltétel
nem_elágazó_feltétel	::=	not elágazó_feltétel
elágazó_feltétel	::=	feltétel_címkesorozat [vagy_kifejezés] elágazó_feltétel
		| feltétel_címkesorozat [vagy_kifejezés]
		| feltétel_címkesorozat
feltétel_címkesorozat	::=	előre_jel címkesorozat
		| hátra_jel címkesorozat

Az elágazó lekérdezésben egy adott címkéjű csúcsra vonatkozó feltétel akkor igaz, ha létezik olyan csúcssorozat, amely illeszkedik a feltételre. Például a *gyökér//a/b*[*\c//d* and not *\e/f*]*/g* lekérdezés azokat a *g* címkéjű csúcsokat keresi meg, amelyekhez a gyökérből el lehet jutni úgy, hogy a két utolsó előtti csúcs címkéje *a* és *b*, továbbá teljesül erre a *b* címkéjű csúcsra, hogy egyrészt létezik olyan *c* címkéjű szülője, amelynek van *d* címkéjű leszármazottja, másrészt nincs olyan *e* címkéjű gyereke, amelynek lenne *f* címkéjű szülője.

Ha egy elágazó lekérdezésből elhagyjuk az összes [ ] jelek közé zárt feltételt, akkor az elágazó lekérdezéshez tartozó ***főlekérdezést*** kapjuk. Az előbbi példában ez a *gyökér//a/b/g* lekérdezés. A főlekérdezés mindig egy előre-címkesorozatnak felel meg.

Az elágazó lekérdezéseknek természetes módon megfeleltethetünk egy irányított gráfot. A lekérdezésben szereplő címkesorozatnak ugyanolyan címkéjű csúcsokat feleltetünk meg, a / és a \ elválasztójel esetén az egymás utáni csúcsokat a megfelelő irányítású éllel kötjük össze, a // és a \\ esetén szintén a megfelelő irányított élt rajzoljuk be, és megcímkézzük az élt // vagy \\ címkével. Végül a logikai összekapcsolók jelét a hozzátartozó feltételek kezdő élének címkéjeként vesszük fel. Így előfordulhat, hogy egy élnek két címkéje is lesz például // és „and”. Vegyük észre, hogy a megadott nyelvtanból következően a kapott gráf nem tartalmazhat irányított kört.

Az így kapott fa alapján definiálhatjuk a lekérdezés bonyolultságának egy egyszerű mértékét. A főlekérdezés csúcsaihoz, és azokhoz a csúcsokhoz, amelyekből irányított út vezet a főlekérdezés valamely csúcsához, 0-t rendelünk. Ezután a 0 jelű csúcsokból irányított úttal elérhető eddig nem jelölt csúcsokhoz 1-et rendelünk. Azokhoz az eddig nem jelölt csúcsokhoz rendelünk 2-t, amelyekből elérhető 1 jelű csúcs. A 2 jelű csúcsokból elérhető nem jelölt csúcsokhoz 3-at rendelünk, és így tovább, a $2k$ jelű csúcsokból elérhetők, nem jelölt csúcsokhoz $2k + 1$-et rendelünk, ezután azokhoz a nem jelölt csúcsokhoz, amelyekből a $2k + 1$ jelű csúcsok elérhető, $2k + 2$-t rendelünk. A lekérdezéshez tartozó legnagyobb értékű csúcs értékét a ***fa mélységének*** nevezzük. A fa mélysége azt fejezi ki, hogy a lekérdezés kiértékelése során hányszor kell irányt váltani, azaz az élek irányításának megfelelően gyerekeket, vagy fordítva, szülőket kell keresni. Ugyanazt a lekérdezést a feltételek különböző mélységű beágyazásával többféle módon is megadhattuk volna, de belátható, hogy az így definiált érték független a felírás módjától, ezért nem a feltételek egymásba ágyazásának számával definiáltuk a lekérdezés bonyolultságát.

Az 1-index a beérkező útvonalak alapján osztályozta a csúcsokat, a biszimuláció alapján. A kiszámításhoz használt stabilitás fogalma az ***utód-stabilitás*** volt, azaz a gráf csúcsainak A halmaza ***utód-stabil*** a csúcsok egy B halmazára nézve, ha $A \subseteq E(B)$ vagy $A \cap E(B) = \emptyset$, ahol $E(B)$ a B-ből éllel elérhető csúcsok halmazát jelöli. Egy partíció utód-stabil, ha bármely két partíciós tag utód-stabil egymásra nézve. Az 1-index az azonos címkék alapján történő osztályozás legdurvább utód-stabil partíciója, amely a PT-algoritmussal számolható ki. Elágazó lekérdezések esetén az élek irányításával szemben is kell haladni, így az ellenkező irányultságú ***előd-stabilitás*** fogalmára is szükségünk lesz. A gráf csúcsainak A halmaza ***előd-stabil*** a csúcsok egy B halmazára nézve, ha $A \subseteq E^{-1}(B)$ vagy $A \cap E^{-1}(B) = \emptyset$, ahol $E^{-1}(B)$ azokat a csúcsokat jelöli, amelyekből a B valamelyik csúcsa elérhető.

31.41. definíció. *Az **FB-index** az alappartíció legdurvább olyan finomítása, amely egyszerre előd-stabil és utód-stabil.*

Vegyük észre, hogy ha a gráf éleinek irányítását megfordítjuk, akkor az előd-stabil partícióból utód-stabil partíció lesz, és fordítva, tehát a PT-algoritmus, illetve annak javításai használhatók a legdurvább előd-stabil partíció kiszámítására. A következő algoritmusban ezt használjuk fel. Kiindulunk az azonos címkéjű osztályozásból, kiszámítjuk az ehhez tartozó 1-indexet, megfordítjuk az élek irányítását, és ezt tovább finomítjuk úgy, hogy kiszámoljuk az ehhez tartozó 1-indexet. Amikor megáll az algoritmus, akkor a kiindulási partíciónak olyan finomítását kapjuk, amely előd-stabil és utód-stabil is egyszerre. Az Olvasóra bízzuk annak bizonyítását, hogy a legdurvább ilyen tulajdonságú partíciót kaptuk.

FB-INDEX-KÉSZÍTŐ(V, E)

```
1  P ← A(0)                          ▷ Az azonos címkéjű osztályozásból indulunk ki.
2  while P változik
3        do P ← PT(V, E^{-1}, P)     ▷ Kiszámítjuk az 1-indexet.
4           P ← PT(V, E, P)          ▷ Megfordítjuk az élek irányítását, és
                                     ▷kiszámoljuk az 1-indexet.
5  return P
```

A kétféle stabilitásból egyszerűen adódik az alábbi következmény.

31.42. következmény. *Az FB-index biztonságos és pontos az elágazó lekérdezésekre nézve.*

Az algoritmus bonyolultságát vissza lehet vezetni a PT-algoritmus bonyolultságára. Mivel a P mindig az előző partíció finomítása, ezért az a legrosszabb eset, ha a finomítás egyesével történik, azaz mindig az egyik partíciós tagból veszünk ki egy elemet, és tesszük egy egyelemű, önálló partíciós tagba. Így legrosszabb esetben a ciklust $O(n)$-szer hajtjuk végre. Tehát az algoritmus költsége legfeljebb $O(mn \lg n)$.

Azt a partíciót, amelyet úgy kapunk, hogy csak egyszer hajtjuk végre a ciklusmagot, ***F+B-indexnek*** hívjuk, ha kétszer hajtjuk végre a ciklusmagot ***F+B+F+B-indexről*** beszélünk, és így tovább.

Könnyen belátható az alábbi állítás.

31.43. állítás. *Az $F{+}B{+}F{+}B{+}\cdots{+}F{+}B$-index, ahol d-szer szerepel az $F{+}B$ tag, biztonságos és pontos a legfeljebb d mélységű elágazó lekérdezésekre nézve.*

Az FB-index alapján egy osztályba kerülő csúcsok elágazó lekérdezéssel nem különböztethetők meg egymástól. Általában ez túl erős megkötés, és ezért az FB-index mérete rendszerint nem sokkal kisebb, mint az eredeti gráf mérete. Nagyon hosszú elágazó lekérdezések a gyakorlatban ritkán fordulnak elő, ezért az A(k)-indexekhez hasonlóan csak lokális ekvivalenciát követelünk meg, de most ezt két paraméterrel jellemezzük attól függően, hogy az irányításnak megfelelő, vagy fordított iránynak megfelelő utak hosszát akarjuk korlátozni. A lekérdezés mélységét is előre korlátozhatjuk. Bevezetjük az FB(f, b, d)-indexet, amellyel az ilyen típusú feltételekkel korlátozott elágazó lekérdezések pontosan kiértékelhetők. A korlátozásoknak eleget nem tevő elágazó lekérdezéseket is az indexen értékeljük ki, csak ebben az esetben az eredményt még ellenőrizni is kell.

FB(f, b, d)-INDEX-KÉSZÍTŐ(V, E, f, b, d)

```
1  P ← A(0)                                  ▷ Az azonos címkéjű osztályozásból indulunk ki.
2  for i ← 1 to d
3      do P ← NAIV-KÖZELÍTÉS(V, E^{-1}, P, f)   ▷ Kiszámítjuk az A(f)-indexet.
4         P ← NAIV-KÖZELÍTÉS(V, E, P, b)       ▷ Megfordítjuk az élek irányítását, és
                                                ▷Kiszámoljuk az A(b)-indexet.
5  return P
```

Az algoritmus költsége az A(k)-index kiszámítási költsége alapján legfeljebb $O(dm\max(f, b))$, ami sokkal kedvezőbb, mint az FB-index kiszámítási költsége, és eredményül általában jóval kisebb indexgráfot kapunk.

A kapott indexre nyilván teljesül a következő állítás.

31.44. állítás. *Az FB(f, b, d)-index biztonságos és pontos azokra az elágazó lekérdezésekre, amelyekben az előre-sorozatok hossza legfeljebb f, a hátra-sorozatok hossza legfeljebb b, és a lekérdezéshez tartozó fa mélysége legfeljebb d.*

Speciális esetként kapjuk, hogy az FB(∞, ∞, ∞)-index megegyezik az FB-indexszel, az FB(∞, ∞, d)-index megegyezik az F+B+$\cdots$+F+B-indexszel, ahol az F+B tag d-szer szerepel, az FB($\infty, 0, 1$)-index megegyezik az 1-indexszel, és végül az FB($k, 0, 1$)-index megegyezik az A(k)-indexszel.

Gyakorlatok

31.7-1. Bizonyítsuk be, hogy a FB-INDEX-KÉSZÍTŐ algoritmus az alappartíció legdurvább előd-stabil és utód-stabil finomítását állítja elő.

31.7-2. Bizonyítsuk be a 31.44. állítást.

31.8. Az indexek frissítése

Az adatbázis-kezelésben általában három fontos szempontot tartunk szem előtt. Minél kisebb legyen a tároláshoz szükséges hely, minél gyorsabbak legyenek a lekérdezések, és minél gyorsabban lehessen az adatbázisban beszúrást, törlést, módosítást végrehajtani. Általában az egyik szempont szerinti jó megoldás rosszabb a másik szempontból. A tipikus lekérdezésekre vonatkozó indexek hozzáadásával növekszik a tárolás mérete, viszont cserébe

a lekérdezéseket az indexeken lehet kiértékelni, így gyorsabbak a lekérdezések. A gyakran módosuló, dinamikus adatbázisok esetén számolnunk kell azzal is, hogy nem csak az eredeti adatokon kell a módosítást végrehajtani, hanem az indexeket is megfelelőképpen kell megváltoztatni. A triviálisan pontos, de legköltségesebb módszer, hogy minden egyes módosítás esetén újra elkészítjük az indexeket. Érdemes olyan eljárásokat keresni, hogy a meglevő indexek minél kisebb módosításával jussunk el azokhoz az indexekhez, amelyek már a változásokat tükrözik.

Előfordulhat, hogy az indexet, vagy annak módosítását is megindexeljük. Egy index indexe az eredeti gráfnak is indexe, bár formailag indexcsúcsokat tartalmazó osztályokból áll, de egyesíthetjük az egy osztályba tartozó indexcsúcsok elemeit. Látható, hogy ezzel a megfeleltetéssel a gráf csúcsainak egy partícióját kapjuk, vagyis egy indexet.

A következőkben a félig strukturált adatbázisok módosításai közül azt vizsgáljuk részletesebben, mikor egy új gráfot kapcsolunk a gyökérhez, illetve mikor egy új élt adunk a gráfhoz. Mindkettő tipikus művelet, mivel ezekre van szükség egy új weboldal készítésekor, illetve egy új hivatkozás elhelyezésekor.

Tegyük fel, hogy $I(G)$ a G gráf 1-indexe. Legyen H olyan gráf, amelynek nincs G-vel közös csúcsa. Jelölje $I(H)$ a H 1-indexét. Legyen $F = G + H$, azaz F az a gráf, amelyet úgy kapunk, hogy a G és H gyökerét egyesítjük. Az $I(G + H)$-t szeretnénk előállítani $I(G)$ és $I(H)$ segítségével. Ebben segít a következő állítás.

31.45. állítás. *Legyen a G gráf 1-indexe $I(G)$, és legyen J az $I(G)$ tetszőleges finomítása. Ekkor $I(J) = I(G)$.*

Bizonyítás. Legyen u, v a G két csúcsa. Azt kell belátni, hogy u és v biszimuláns a G-ben az 1-index szerint akkor és csak akkor, ha $J(u)$ és $J(v)$ biszimuláns az $I(G)$ indexgráfban az $I(G)$ 1-indexe szerint. Legyen u és v biszimuláns a G-ben az 1-index szerint. Belátjuk, hogy megadható olyan biszimuláció, amely szerint $J(u)$ és $J(v)$ biszimuláns az $I(G)$-ben. Mivel az 1-index a legdurvább biszimulációnak megfelelő partíció, ezért a megadott biszimuláció az 1-indexnek megfelelő biszimulációnak finomítása, vagyis $J(u)$ és $J(v)$ az $I(G)$ 1-indexének megfelelő biszimuláció szerint is biszimulánsak. Legyen $J(a) \approx' J(b)$ akkor és csak akkor, ha a és b biszimuláns G-ben az 1-index szerint. Vegyük észre, hogy mivel J az $I(G)$ finomítása, ezért $J(a) \approx' J(b)$ esetén $J(a)$ és $J(b)$ minden eleme biszimuláns egymással G-ben. Ahhoz, hogy megmutassuk, hogy a $\approx'$ reláció biszimuláció, legyen $J(u')$ a $J(u)$ szülője, ahol u' az u_1 szülője, és u_1 a $J(u)$ eleme. Ekkor u_1, u és v biszimuláns G-ben, tehát van a v-nek olyan v' szülője, hogy u' és v' biszimuláns G-ben. Emiatt $J(v)$-nek $J(v')$ a szülője, és $J(u') \approx' J(v')$. Mivel a biszimuláció szimmetrikus, ezért a $\approx'$ reláció is szimmetrikus. Ezzel a bizonyítás egyik irányát beláttuk.

Legyen most $J(u)$ és $J(v)$ biszimuláns $I(G)$-ben az $I(G)$ 1-indexe szerint. A fentiekhez hasonlóan, elég belátni, hogy megadható olyan biszimuláció a G csúcsain, amely szerint u és v biszimulánsak. Legyen $a \approx' b$ akkor és csak akkor, ha $J(a) \approx J(b)$ az $I(G)$ 1-indexe szerint. A biszimuláció igazolásához legyen u' az u-nak szülője. Ekkor $J(u')$ is szülője $J(u)$-nak. Mivel $u \approx' v$ esetén $J(u)$ és $J(v)$ biszimulánsak, ezért van olyan $J(v'')$ szülője a $J(v)$-nek, hogy $J(u')$ és $J(v'')$ biszimuláns az $I(G)$ 1-indexe szerint, és v'' szülője a $J(v)$ valamely v_1 elemének. Mivel v és v_1 biszimulánsak, ezért v-nek van olyan v' szülője, amelyre igaz, hogy v' és v'' biszimuláns. Felhasználva a bizonyítás másik irányát, ebből következik, hogy $J(v')$ és $J(v'')$ biszimuláns az $I(G)$ 1-indexe szerint. A tranzitivitás miatt ekkor $J(u')$ és

$J(v')$ biszimulánsak az $I(G)$ 1-indexe szerint, tehát $u' \approx' v'$. Mivel a definiált $\approx'$ relációs szimmetrikus, így biszimulációt kaptunk. ■

Az állítás következményeként diszjunkt G, H gráfok esetén az $I(G + H)$-t az alábbi algoritmussal lehet előállítani.

GRÁFHOZZÁADÁS-1-INDEXE(G, H)

```
1  P_G ← A_G(0)                          ▷ P_G a címkék szerinti alappartíció.
2  P_H ← A_H(0)                          ▷ P_H a címkék szerinti alappartíció.
3  I_1 ← PT(V_G, E_G^{-1}, P_G)          ▷ I_1 a G 1-indexe.
4  I_2 ← PT(V_H, E_H^{-1}, P_H)          ▷ I_2 a H 1-indexe.
5  J ← I_1 + I_2                         ▷ Az 1-indexeket összekapcsoljuk a gyökerüknél.
6  P_J ← A_J(0)                          ▷ P_J a címkék szerinti alappartíció.
7  I ← PT(V_J, E_J^{-1}, PJ)             ▷ I a J 1-indexe.
8  return I
```

Az algoritmus költségének kiszámításánál feltesszük, hogy a G-hez tartozó $I(G)$ 1-index már a rendelkezésünkre áll. Ekkor az $I(G+H)$ készítésének összköltsége $O(m_H \lg n_H + (m_{I(H)}+m_{I(G)}) \lg(n_{I(G)}+n_{I(H)}))$, ahol n, m a megfelelő gráf csúcsainak, illetve éleinek számát jelöli.

Ahhoz, hogy az algoritmus jól működik, csak azt kell észrevenni, hogy a diszjunktság miatt az $I(G) + I(H)$ az $I(G + H)$-nak finomítása. Ebből az is következik, hogy $I(G) + I(H)$ biztonságos és pontos index, így ezt is használhatjuk, ha nem célunk a minimális index megkeresése. Ez különösen akkor hasznos, ha többször kell gráfot hozzáadni a már meglevő gráfhoz. Ilyenkor a ***lusta módszert*** használjuk, azaz nem páronként számoljuk ki a minimális indexet, hanem összeadjuk a tagok indexeit, és csak egyszer minimalizálunk.

31.46. állítás. *Legyen a G_i gráf 1-indexe $I(G_i)$, $i = 1, \ldots, k$, és a gráfoknak ne legyen közös csúcsuk. Ekkor a gyökereknél összekapcsolt gráfok $I(G_1 + \cdots + G_k)$ 1-indexére igaz, hogy $I(G_1 + \cdots + G_k) = I(I(G_1) + \cdots + I(G_k))$.*

A következőkben azt vizsgáljuk, hogy mi történik az indexszel, ha a gráfban behúzunk egy új élt. Egy ilyen műveletnek is jelentős lehet a hatása. Nem nehéz olyan gráfot konstruálni, hogy az indexe a gyökértől 2 távolságban két teljesen azonos részgráfot tartalmazzon, amelyek pont egy él hiánya miatt nem vonhatók össze. Ha ezt a kritikus élt húzzuk be, akkor a két részgráf összevonható, és az indexgráf mérete majdnem a felére csökken.

Tegyük fel, hogy a G gráfban egy új élt húzunk be u-ból v-be. Legyen az így kapott gráf G', azaz $G' = G + (u, v)$. Az $I(G)$ partíció legyen a G 1-indexe. Ha az $I(u)$-ból mutatott él az $I(v)$-be az $I(G)$-ben, akkor nem kell módosítani az indexgráfot, mert a stabilitás miatt az $I(v)$ elemeinek, azaz a v-vel biszimuláns összes csúcsnak létezik szülője az $I(u)$-ban, melynek elemei az u-val biszimulánsak. Vagyis $I(G') = I(G)$.

Ha az $I(u)$-ból nem vezetett él az $I(v)$-be, akkor be kellene húzni ezt az élt, de ezzel elromolhat az a tulajdonság, hogy $I(v)$ stabil az $I(u)$-ra nézve. Legyen Q az a partíció, amelyet úgy kapunk az $I(G)$-ből, hogy az $I(v)$-t kettévágjuk, az egyik tagba áttesszük a v-t, a többit pedig a másik tagba tesszük, és minden más partíciós tagot változatlanul hagyunk. A Q

meghatározza az éleit a szokásos módon, azaz ha az egyik partíciós tag valamely eleméből vezet él egy másik partíciós tag valamely elemébe, akkor a két partíciós tagot ugyanilyen irányítással összekötjük.

Legyen az X partíció az eredeti $I(G)$. Ekkor Q finomítása X-nek, és Q stabil az X-re nézve a G' alapján. Vegyük észre, hogy a PT-algoritmusban használt X és Q partíciókra pont ez az invariáns tulajdonság szerepelt. A 31.45. állítás szerint elég az $I(G')$ egy finomítását megtalálni. Ha a G' alappartíciójának egy tetszőleges stabil finomítását meg tudjuk találni, akkor mivel az 1-index a legdurvább stabil finomítás, így ez az $I(G')$-nek finomítása lenne. Az X az alappartíciónak, vagyis a címkék szerinti osztályozásnak a finomítása, és ugyanez igaz a Q-ra is. Tehát ha Q stabil, akkor készen vagyunk. Ha nem stabil, akkor stabilizálhatjuk a PT-algoritmussal, amelyet most a fenti X és Q partíciókkal indítunk. Először azokat a partíciós tagokat kell megvizsgálni, amelyek v-nek valamelyik gyerekét tartalmazzák, mert ezek nem biztos, hogy stabilak maradtak a kettévágással keletkezett két partíciós tagra nézve. A PT-algoritmus kettéhasítással stabilizálja ezeket, de emiatt ezek gyerekeit is meg kell vizsgálni, mert ezeknél is elromolhatott a stabilitás, és így tovább. Ezzel a stabilitás-terjesztő módszerrel végül egy stabil finomításhoz jutunk el. Mivel közben csak a v-ből elérhető csúcsokat járjuk be, így lehet, hogy nem a legdurvább stabil finomítást kaptuk. Ezzel beláttuk, hogy a következő algoritmus a $G + (u, v)$ gráf 1-indexét számolja ki.

ÉLHOZZÁADÁS-1-INDEXE$(G, (u, v))$

1 $P_G \leftarrow A_G(0)$ ▷ P_G a címkék szerinti alappartíció.
2 $I \leftarrow PT(V_G, E_G^{-1}, P_G)$ ▷ I a G 1-indexe.
3 $G' \leftarrow G + (u, v)$ ▷ Behúzzuk az (u, v) élt.
4 **if** $(I(u), I(v)) \in E_I$ ▷ Ha $I(u)$-ből $I(v)$-be mutatott él, akkor nem kell módosítani.
5 **then return** I
6 $I' \leftarrow \{v\}$ ▷ Kettévágjuk az $I(v)$-t.
7 $I'' \leftarrow I(v) \setminus \{v\}$
8 $X \leftarrow I$ ▷ X a régi partíció.
9 $E_I \leftarrow E_I \cup \{(I(u), I(v))\}$ ▷ Behúzunk egy élt $I(u)$-ból $I(v)$-be.
10 $Q \leftarrow (I \setminus \{I(v)\}) \cup \{I', I''\}$ ▷ $I(v)$-t kicseréljük I'-re és I''-re.
11 $E \leftarrow E_Q$ ▷ Meghatározzuk Q éleit.
12 $J \leftarrow PT(V_{G'}, E_{G'}^{-1}, P_{G'}, X, Q)$ ▷ A PT-algoritmust X-szel és Q-val indítva futtatjuk le.
13 $J \leftarrow PT(V_J, E_J^{-1}, P_J)$ ▷ J a legdurvább stabil finomítás.
14 **return** J

A gyakorlati tapasztalatok azt mutatják, hogy a 13. lépést elhagyhatjuk, mert a 12. lépésben előállított stabil finomítás már elég jó közelítése a legdurvább stabil finomításnak, alig 5% a méretük közti különbség.

A következőkben az FB-indexek és A(k)-indexek frissítésével foglalkozunk. Az FB-index az 1-indextől annyiban tér el, hogy két csúcs akkor kerül egy hasonlósági osztályba, ha nemcsak a bejövő utak, hanem a kimenő utak mentén is ugyanolyan címkesorozatokból álló halmazokat kapunk mindkét csúcsra. Láttuk, hogy az FB-index készítéséhez a PT-algoritmust kell kétszer futtatni, úgy, hogy másodszor a fordított irányítású gráfra kell alkalmazni az algoritmust. Az FB-index frissítése az 1-indexhez hasonlóan történik. A következő

állítás bizonyítása a 31.45. állítás bizonyításához hasonlóan történik, ezért ennek igazolását az Olvasóra bízzuk.

31.47. állítás. *Legyen a G gráf FB-indexe $I(G)$, és legyen J az $I(G)$ tetszőleges finomítása. Jelöljük $I(J)$-vel a J FB-indexét. Ekkor $I(J) = I(G)$.*

Az állítás következményeként diszjunkt G, H gráfok esetén a $G + H$ FB-indexét az alábbi algoritmussal lehet előállítani.

GRÁFHOZZÁADÁS-FB-INDEXE(G, H)

```
1  I1 ← FB-INDEX-KÉSZÍTŐ(VG, EG)                ▷ I1 a G FB-indexe.
2  I2 ← FB-INDEX-KÉSZÍTŐ(VH, EH)                ▷ I2 a H FB-indexe.
3  J ← I1 + I2                  ▷ Az FB-indexeket összekapcsoljuk a gyökerüknél.
4  I ← FB-INDEX-KÉSZÍTŐ(VJ, EJ)                 ▷ I a J FB-indexe.
5  return I
```

Az (u, v) él hozzáadásánál most arra is figyelnünk kell, hogy a stabilitás mindkét irányban elromolhat, ezért nemcsak az $I(v)$-t vágjuk ketté $\{v\}$-re és $(I \setminus \{v\})$-re, hanem az $I(u)$-t is, $\{u\}$-ra és $(I(u) \setminus \{u\})$-ra. X legyen a módosítás előtti partíció, és Q a szétvágásokkal kapott partíció. Az FB-INDEX-KÉSZÍTŐ algoritmus 3. lépésében szereplő PT-algoritmus hívását az X és Q partíciókkal indítjuk. A stabilizáláskor most nemcsak a v utódjait, hanem az u elődjeit is be fogjuk járni.

ÉLHOZZÁADÁS-FB-INDEXE(G, (u, v))

```
1  I ← FB-INDEX-KÉSZÍTŐ(VG, EG)                 ▷ I a G FB-indexe.
2  G' ← G + (u, v)                              ▷ Behúzzuk az (u, v) élt.
3  if (I(u), I(v)) ∈ EI      ▷ Ha I(u)-ből I(v)-be mutatott él, akkor nem kell módosítani.
4     then return I
5  I1 ← {v}                                     ▷ Kettévágjuk az I(v)-t.
6  I2 ← I(v) \ {v}
7  I3 ← {u}                                     ▷ Kettévágjuk az I(u)-t.
8  I4 ← I(u) \ {u}
9  X ← I                                        ▷ X a régi partíció.
10 EI ← EI ∪ {(I(u), I(v))}                     ▷ Behúzunk egy élt I(u)-ból I(v)-be.
11 Q ← (I \ {I(v), I(u)}) ∪ {I1, I2, I3, I4}    ▷ I(v)-t I1-re és I2-re, I(u)-t I3-ra
                                                   és I4-re cseréljük.
12 E ← EQ                                       ▷ Meghatározzuk Q éleit.
13 J ← FB-INDEX-KÉSZÍTŐ(VG', EG', X, Q)         ▷ Az FB-INDEX-KÉSZÍTŐ algoritmusban a PT-
                                                   algoritmust X-szel és Q-val indítva futtatjuk le.
14 J ← FB-INDEX-KÉSZÍTŐ(VJ, EJ)                 ▷ J a legdurvább előd-stabil és utód-stabil finomítás.
15 return J
```

Az A(k)-index frissítése az él hozzáadásakor eltér az eddigiektől. A gráf hozzáadásával még nincs baj, mivel igaz a következő állítás, amelynek igazolását az Olvasóra bízzuk.

31.48. állítás. *Legyen a G gráf $A(k)$-indexe $I(G)$, és legyen J az $I(G)$ tetszőleges finomítása. Jelöljük $I(J)$-vel a J $A(k)$-indexét. Ekkor $I(J) = I(G)$.*

Az állítás következményeként diszjunkt G, H gráfok esetén a $G + H$ $A(k)$-indexét az alábbi algoritmussal lehet előállítani.

GRÁFHOZZÁADÁS-A(k)-INDEXE(G, H)

```
1  P_G ← A_G(0)                          ▷ P_G a címkék szerinti alappartíció.
2  I_1 ← NAIV-KÖZELÍTÉS(V_G, E_G^{-1}, P_G, k)   ▷ I_1 a G A(k)-indexe.
3  P_H ← A_H(0)                          ▷ P_H a címkék szerinti alappartíció.
4  I_2 ← NAIV-KÖZELÍTÉS(V_H, E_H^{-1}, P_H, k)   ▷ I_1 a H A(k)-indexe.
5  J ← I_1 + I_2                         ▷ Az A(k)-indexeket összekapcsoljuk.
6  P_J ← A_J(0)                          ▷ P_J a címkék szerinti alappartíció.
7  I ← NAIV-KÖZELÍTÉS(V_J, E_J^{-1}, P_J, k)     ▷ I a J A(k)-indexe.
8  return I
```

Ha egy (u, v) élt adunk a gráfhoz, akkor az eddigiek szerint $I(v)$-ből leválasztjuk v-t egy $I' = \{v\}$ tagba, ezután az elromló k-stabilitásokat kell kijavítani a v leszármazottjait bejárva, de csak k távolságnyira. A probléma abból adódik, hogy az $A(k)$-index csak a k-biszimulációra tartalmaz információt, $(k-1)$-re nem. Például legyen v-nek gyereke a v_1 és legyen $k = 1$. Az 1-index szerinti stabilizáláskor a v_1-et le kell választanunk az osztályából, ha ugyanebben az osztályban van olyan elem, amelynek v nem szülője. Ez a feltétel az $A(1)$-index esetén túl erős, és ezért túl sok fölösleges szétvágást okoz. Ebben az esetben ugyanis csak akkor kell v_1-et leválasztani, ha van ebben az osztályban olyan elem, amelynek nincs olyan szülője, amely 0-biszimuláns, azaz azonos címkéjű lenne a v-vel. Emiatt ha az $A(k)$-indexet az eddigiek szerint frissítenénk él hozzáadásakor, akkor nagyon rossz közelítését kapnánk a módosításhoz tartozó $A(k)$-indexnek, ezért nem ezt az eljárást használjuk. Az alapötlet, hogy nem csak az $A(k)$-indexet tartjuk nyilván, hanem az összes $A(i)$-indexet, ahol $i = 0, \ldots, k$. Yi és társai megadnak egy algoritmust, amely ezen az elven működik, és a módosításnak megfelelő $A(k)$-indexet állítja elő. A megadott algoritmusok kis változtatással élek törlésére is használhatók, 1-index és $A(k)$-index esetén is.

Gyakorlatok

31.8-1. Bizonyítsuk be a 31.47. állítást.

31.8-2. Adjunk meg algoritmust arra, hogyan módosítsuk az indexet, mikor egy élt törlünk az adatgráfból. Különböző típusú indexeket vizsgáljunk. Mi lesz az algoritmus költsége?

31.8-3. Adjunk algoritmusokat arra, hogyan frissítsük a $D(k)$-indexet, ha az adatgráfot módosítjuk.

Feladatok

31-1. Megszorításokra vonatkozó implikációs probléma
Legyen R, Q reguláris kifejezés, x, y két csúcs. Az $R(x,y)$ predikátum jelentse azt, hogy x-ből elérhető y egy R-re illeszkedő címkesorozattal. Jelölje $R \subseteq Q$ a $\forall x(R(\text{gyökér}, x) \rightarrow Q(\text{gyökér}, x))$ megszorítást. $R = Q$, ha mindkét irányú tartalmazás teljesül. Jelöljön C egy megszorításokból álló véges halmazt, és c egy megszorítást.

a. Bizonyítsuk be, hogy a $C \models c$ implikációs probléma eldöntése 2-EXPSPACE probléma.

b. Jelölje $R \subseteq Q@u$ a $\forall v(R(u,v) \rightarrow Q(u,v))$ megszorítást. Bizonyítsuk be, hogy erre az osztályra nézve az implikációs probléma nem eldönthető.

31-2. Fák transzformációs távolsága
A csúcscímkézett fák közti *transzformációs távolság* legyen a minimális száma annak, ahány elemi művelettel az egyik fa a másikká átalakítható. Három elemi műveletet használhatunk, új csúcs hozzáadását, csúcs törlését, és címke átnevezését.

a. Bizonyítsuk be, hogy a T, T' fák transzformációs távolsága kiszámítható $O(n_T n_{T'} d_T d_{T'})$ időben, $O(n_T n_{T'})$ tárköltséggel, ahol n_T a fa csúcsainak száma, d_T a fa mélysége.

b. Legyen S, S' két fa. Adjunk meg egy algoritmust, amely az összes olyan (T, T') párt előállítja, ahol T, illetve T' szimulálja az S, illetve S' gráfot, és T, T' transzformációs távolsága kisebb egy előre megadott n egész számnál. (Ezt a műveletet *közelítő összekapcsolásnak* nevezzük.)

31-3. Osztott adatbázisok lekérdezése
Az osztott adatbázis egy olyan csúcscímkézett, irányított gráf, amelynek csúcsait m darab partícióba (szerverre) osztottuk. A különböző partíciók közti élek a *kereszthivatkozások*. A kommunikáció a szerverek közti üzenetszórással történik. Egy lekérdezést kiértékelő algoritmus *hatékony,* ha a kommunikációs lépések száma konstans, vagyis független az adatoktól és a lekérdezéstől, valamint a kommunikáció során átvitt adatok összmérete csak a lekérdezésre adott válasz méretétől és a kereszthivatkozások számától függ. Bizonyítsuk be, hogy osztott adatbázisok reguláris lekérdezésére megadható olyan hatékony algoritmus, amelyben a kommunikációs lépések száma 4, és az átvitt adatok mérete $O(n^2) + O(k)$, ahol n a lekérdezésre adott válasz mérete, és k a kereszthivatkozások száma. (*Útmutatás.* próbáljuk megfelelőképpen módosítani a NAIV-KIÉRTÉKELÉS algoritmust.)

Megjegyzések a fejezethez

A fejezet a félig strukturált adatbázisok világának azokat a területeit vizsgálta, amelyekben gráfok morfizmusait lehetett felhasználni. Így alapvetően a sémák, indexek készítését, használatát tárgyaltuk algoritmikus szemszögből. A félig strukturált adatbázisok, illetve az XML világa ennél jóval kiterjedtebb. A félig strukturált adatbázisok kialakulásának, aktuális témaköreinek, lehetséges további fejlődésének tömör összefoglalását adja Vianu [348] cikke.

A maximális szimuláció kiszámításával M. Henzinger, T. Henzinger és Kopke [161] cikke foglalkozik. Végtelen, de hatékonyan reprezentálható, úgynevezett effektív gráfokra is kiterjesztik a szimulációt, és belátják, hogy az ilyen gráfokra eldönthető, hogy két csúcs hasonló-e. Corneil és Gotlieb [72] cikke a hányados gráfokkal, és a gráfok izomorfizmusának eldöntésével foglalkozik. A relációs modellben használatos normálformákat terjeszti ki XML dokumentumokra Arenas és Libkin [22] cikke. Az általuk bevezetett XNF normálformáról kimutatják, hogy tetszőleges DTD átírható veszteségmentesen XNF normálformára.

Buneman, Fernandez és Suciu [56] tanulmányukban egy strukturális rekurzión alapuló lekérdező nyelvet, az UnQL-t vezetik be, ahol a használt adatmodell biszimulációval definiált. Gottlob, Koch és Pichler az XPath lekérdező nyelv osztályait vizsgálja bonyolultsági és párhuzamosíthatósági szempontból [142]. A bonyolultsági problémák áttekintéséhez Garey és Johnson klasszikus munkáját [127], valamint Stockmeyer és Meyer [323] cikkét ajánljuk.

A PT-algoritmus Paige és Tarjan [262]cikkében szerepel először. A biszimuláción alapuló 1-indexet Milo és Suciu részletesen tárgyalják [250], ezenkívül bevezetik a 2-indexet, és ennek általánosításaként a T-indexet.

Az A(k)-indexet Kaushik, Shenoy, Bohannon és Gudes [190] vezették be. A D(k)-index Chen, Lim és Ong [63] munkájában szerepelt először. A gyakori lekérdezéseken alapuló M(k)-index, illetve M*(k)-index He és Yang [160] eredménye. Az elágazó lekérdezésre vonatkozó FB-indexeket Kaushik, Bohannon, Naughton és Korth [192] vizsgálta először. Az 1-indexek, FB-indexek és A(k)-indexek módosítási algoritmusait Kaushik, Bohannon, Naughton és Shenoy [193] foglalta össze. Az itt tárgyalt eljárásokat javítja és általánosítja Yi, He, Stanoi és Yang munkája [358]. A lekérdezések szelektivitásának vizsgálatára Polyzotis és Garafalakis valószínűségi modellt használ [284]. A strukturális indexek és az invertált listák együttes alkalmazhatóságát javasolja Kaushik, Krishnamurthy, Naughton és Ramakrishnan [191].

Az XML gyakorlati felhasználásával foglalkozó művek TuckernevindexTucker, Alan B. kézikönyve [342], valamint a Khosrow-Pour által szerkesztett enciklopédia [197].

Magyar nyelven az XML elméletének még nincs irodalma, viszont a gyakorlati felhasználással több könyv is foglalkozik [35, 52, 247].

Irodalomjegyzék

[1] S. Abiteboul. Querying semi-structured data. In F. Afrati, P. Kolaitis (szerkesztők), *Proceedings of ICDT'97*, *Lecture Notes in Computer Science* 1186. kötete, 1–18. o. Springer-Verlag, 1997. 1483

[2] S. Abiteboul, O. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 254–263. o. ACM-Press, 1998. 1483

[3] P. Adriaans, D. Zantinge. *Data Mining*. Addison-Wesley Longman, 1996. Magyarul: *Adatbányászat*. Panem, Budapest, 2002. 1407

[4] R. Agrawal, H. Mannila, R., H. Toivonen, I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, 307–328. o., 1996. 1407

[5] R. Agrawal, R. Srikant. Mining sequential patterns. In P. S. Yu, A. L. P. Chen (szerkesztők), *Proceedings of the 11th International Conference on Data Engineering, ICDE*, 3–14. o. IEEE Computer Society, 1995. 1407

[6] R. Agrawal, R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, Carlo Zaniolo (szerkesztők), *Proceedings of the 20th International Conference Very Large Data Bases, VLDB*, 487–499. o. Morgan Kaufmann Publishers, 1994. 1406

[7] R. Agrawal, T. Imielinski, A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman, S. Jajodia (szerkesztők), *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 207–216. o., 1993. 1406

[8] M. Agrawal, N. Kayal, N. Saxena. PRIMES is in P. http://www.cse.iitk.ac.in/users/manindra/, 2002. 1093

[9] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974 (magyarul: *Számítógép-algoritmusok tervezése és analízise*. Műszaki Könyvkiadó, 1982. 831

[10] A. V. Aho, R. Sethi, J. D. Ullman. *Compilers, Principles, Techniques and Tools*. Addison-Wesley, 1986. 1009, 1010

[11] A. V. Aho, J. D. Ullman. *The Theory of Parsing, Translation and Compiling Vol. I.* Prentice-Hall, 1972. 958, 1009, 1010

[12] A. V. Aho, J. D. Ullman. *The Theory of Parsing, Translation and Compiling Vol. II.* Prentice-Hall, 1973. 958, 1009, 1010

[13] M. Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 10–19. o., 1998. 891, 892

[14] M. Ajtai, J. Komlós, E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983. 1053

[15] S. Albers, H. Bals. Dynamic TCP acknowledgement, penalizing long delays. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms*, 47–55. o., 2003. 1382

[16] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995. 1296

[17] M. Anderberg. *Cluster Analysis for Applications*. Morgan Kaufmann Publishers, 1973. 1434, 1435

[18] E. Andersen, K. Andersen. An implementation of the homogeneous algorithm. In T. Terlaky H. Frenk, K. Roos, S. Zhang (szerkesztők), *High Performance Optimization*, 197-232. o. Kluwer Academic Publishers, 2000. 1296

[19] E. Andersen, J. Gondzio, Cs. Mészáros, X. Xu. Implementation of interior point methods for linear programming. In T. Terlaky (szerkesztő), *Interior Point Methods of Mathematical Programming*, 189-252. o. Kluwer Academic Publishers, 1996. 1295

[20] M. Ankerst, M. Breunig, H. Kriegel, J. Sander. Optics: ordering points to identify the clustering structure. In *SIGMOD'99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, 49–60. o., 1999. 1435

[21] T. Apostol. *Introduction to Analytic Number Theory*. Springer, 1995. 1114

[22] M. Arenas, L. Libkin. A normal form for XML documents. In *Proceedings of the 21st Symposium on Principles of Database Systems*, 85–96. o., 2002. 1523

[23] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, 44:486–504, 1997. 1382, 1383

[24] A. Asteroth, C. Christel. *Theoretische Informatik*. Pearson Studium, 2002. 958

[25] H. Attiya, C. Dwork, N. A. Lynch, L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM*, 41:122–142, 1994. 1166

[26] H. Attiya, J. Welch. *Distributed Computing, Fundamentals, Simulations and Advanced Topics*. McGraw-Hill, 1998. 1166, 1167

[27] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985. 1166

[28] B. Awerbuch, Y. Azar S. Plotkin. Throughput-competitive online routing. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, 32–40. o., 1993. 1382

[29] Y. Azar. On-line load balancing. *Lecture Notes in Computer Science* 1442. kötete, 178–195. o. Springer-Verlag, 1998. 1383

[30] I. Bach. *Formális nyelvek (Formal Languages)*. Typotex, 2001. 959, 1010

[31] A. Bagyinszkiné Orosz. Petri-hálók I. ELTE, Numerikus és Gépi Matematika Tanszék, 1977. 1227

[32] A. Bagyinszkiné Orosz. Petri-hálók és formális nyelvek, 1979. Kandidátusi értekezés, Magyar Tudományos Akadémia. 1227

[33] R. Baker, G. Glyn. The Brun–Titchmarsh theorem on average. In *Proceedings of a Conference in Honor of Heini Halberstam (Progress in Mathematics)*, 138. kötet. Birkhäuser, 1996. 1113

[34] B. S. Baker, J. S. Schwartz. Shelf algorithms for two dimensional packing problems. *SIAM Journal on Computing*, 12:508–525, 1983. 1382

[35] C. F. Bates. *XML in Theory and Praxis*. John Wiley & Sons, 2003. Magyarul: *XML. Elmélet és gyakorlat*. Panem, 2004. 1523

[36] R. Bello, K. Dias, A. Downing, J. Freenan, T. Finnerty, W. D. Norcott, H. Sun, A. Witkowski, M. Ziauddin. Materialized views in Oracle. In *Proceedings of Very Large Data Bases'98*, 659–664. o., 1998. 1483

[37] E. Bender, R. Canfield. The asymptotic number of labeled graphs with given degree sequences. *Combinatorial Theory Series A*, 24:296–307, 1978. 1053

[38] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, 2002. 1434

[39] P. Berkhin, J. D. Becher. Learning simple relations: Theory and applications. In *Proceedings of the Second SIAM International Conference on Data Mining*, 2002. 1434

[40] P. Berman, J. Garay. Cloture votes: $n/4$-resilient distributed consensus in $t+1$ rounds. *Mathematical Systems Theory*, 26(1):3–19, 1993. 1166

[41] E. Best, R. Devillers, M. Koutny. *Petri Net Algebra*. Springer-Verlag, 2001. 1227

[42] E. Best, R. Hopkins, J. G. Hall. B(PN)2 – a basic Petri net programming notation. In A. B. M. Reeve, G. Wolf (szerkesztők), *Parallel Architectures and Languages Europe (PARLE'93)*, *Lecture Notes in Computer Science* 694. kötete, 379–390. o. Springer-Verlag, 1993. 1227

[43] R. Bhattacharjee, P. Pandey. Primality testing. Technical report, Indian Institute of Technology Kanpur, 2001. 1113

[44] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas of Communications*, 18(3):535–547, 2000. 1346

[45] J. Bánkfalvi, Zs. Bánkfalvi, G. Bognár. *A formális nyelvek szintaktikus elemzése*. Közgazdasági és Jogi Kiadó, 1978. 1010

[46] F. Bodon. Adatbányászati algoritmusok. http://www.cs.bme.hu/ bodon/magyar/index.htm (elektronikus kézirat), 2005. 1408

[47] F. Bodon. A fast APRIORI implementation. In B. Goethals, M. J. Zaki (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, *CEUR Workshop Proceedings* 90. kötete, 2003. 1407

[48] F. Bodon. Surprising results of trie-based fim algorithms. In B. Goethals, M. J. Zaki, R. Bayardo (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, *CEUR Workshop Proceedings* 126. kötete, 2004. 1407

[49] G. Bolch, S. Greiner, H. Meer, de, K. Trivedi. *Queueing Networks and Markov Chains, Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 1998. 1346

[50] C. Borgelt. Efficient implementations of apriori and eclat. In B. Bart, M. J. Zaki (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, *CEUR Workshop Proceedings* 90. kötete, 2003. 1407

[51] A. Borodin R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998. 1382

[52] N. Bradley. *The XML Companion (harmadik kiadás)*. Addison-Wesley, 2004. Magyarul: *XML kézikönyv*. Szak, 2005. 1523

[53] A. Z. Broder. On the Resemblance and containment of documents. In *SEQS: Sequences '91*, 1998. http://citeseer.ist.psu.edu/broder97resemblance.html. 1434

[54] J. G. Brookshear. *Theory of Computation – Formal Langauges, Automata, and Complexity*. The Benjamin/Cummings Publishing Company, 1989. 959

[55] P. Buneman. Semistructured data. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 117–121. o. ACM-Press, 1997. 1483

[56] P. Buneman, M. Fernandez, D. Suciu. UnQL: a query language and algebra for semistructured data based on structural recursion. *The International Journal on Very Large Data Bases*, 9(1):76–110, 2000. 1523

[57] D. Burdick, M. Calimlim, J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering*, 443–452. o., 2001. IEEE Computer Society. 1407

[58] J. E. Burns. A formal model for message passing systems. Technical Report91, Indiana University, 1980. 1166

[59] J. E. Burns, N. A. Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*, 107(2):171–184, 1993. 1166

[60] D. Calvanese, D. De Giacomo, M. Lenzerini, M. Varni. Answering regular path queries using views. In *Proceedings of the Sixteenth International Conference on Data Engineering*, 190–200. o., 2000. 1483

[61] J. Carroll, D. Long. *Theory of Finite Automata*. Prentice Hall, 1989. 959

[62] S. Chaudhury, R. Krishnamurty, S. Potomianos, K. Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*, 190–200. o., 1995. 1483

[63] Q. Chen, A. Lim, K. W. Ong. An adaptive structural summary for graph-structured data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 134–144. o., 2003. 1523

[64] Y. Cho, S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980. 1383

[65] M. Chrobak, L. Larmore. An optimal algorithm for k-servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991. 1382

[66] M. Chrobak, L. Larmore. The server problem and on-line games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 7. kötete, 11–64. o. American Mathematical Society, 1992. 1382

[67] M. Chrobak, H. J. Karloff, T. Payne, S. Vishwanathan. New results on the server problem. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991. 1382

[68] D. Comer, R. Sethi. The complexity of trie index construction. *Journal of the ACM*, 24(3):428–440, 1977. 1407

[69] K. D. Cooper, L. Torczon. *Engineering a Compiler*. Morgan Kaufman Publisher, 2004. 1010

[70] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 1990 (Magyarul: *Algoritmusok*. Műszaki Kiadó, 2003, negyedik kiadás). 831

[71] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 2004 (Második kiadás ötödik, javított utánnyomása. Magyarul: *Új algoritmusok*. Scolar Kiadó, 2003). 831

[72] D. G. Corneil, C. Gotlieb. An efficient algorithm for graph isomorphism. *Journal of the ACM*, 17(1):51–64, 1970. 1523

[73] M. Crane, A. Lemoine. *An Introduction to the Regenerative Method for Simulation Analysis*. Springer-Verlag, 1977. 1346

[74] J. Csirik, G. Woeginger. On-line packing and covering problems. *Lecture Notes in Computer Science* 1442. kötete, 147–177. o. Springer-Verlag, 1998. 1382

[75] J. Csirik, G. J. Woeginger. Shelf algorithms for on-line strip packing. *Information Processing Letters*, 63:171–175, 1997. 1382

[76] Z. Csörnyei. *Bevezetés a fordítóprogramok elméletébe I. (Introduction to the Theory of Compilers I.)*. Tankönyvkiadó, 1996. 1010

[77] Z. Csörnyei. *Bevezetés a fordítóprogramok elméletébe II. (Introduction to the Theory of Compilers II)*. ELTE Eötvös Kiadó, 1996. 1010

[78] Z. Csörnyei. *Fordítási algoritmusok (Algorithms of Compilers)*. Erdélyi Tankönyvtanács, 2000. 1010

[79] Gy. Dallos, Cs. Szabó. *Hírközlési csatornák véletlen hozzáférési módszerei (Random Access Methods in Communication Channels)*. Akadémiai Kiadó, 1984. 1330, 1346

[80] G. Dantzig. Impact of linear programming on computer development. *OR/MS Today*, 18(8):12–17, 1988. 1294

[81] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963. 1294

[82] Zs. Darvay. Belsőpontos módszerek a lineáris programozásban, 1997. Elektronikus egyetemi jegyzet, 64 oldal. 1297

[83] Zs. Darvay. Egy új prediktor-korrektor algoritmus a lineáris programozásban. *Alkalmazott Matematikai Lapok*, 22, 2004. Nyomdában. 1297

[84] S. C. Deerwester, S. Dumais, T. K. Landauer, G. W. Furnas, R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990. http://citeseer.ist.psu.edu/deerwester90indexing.html. 1434

[85] I. Deák. *Random Number Generators and Simulation*. Akadémiai Kiadó, 1990. 1345

[86] J. Demetrovics, J. Denev, R. Pavlov. *Bevezetés a számítástudományba (Introduction to Computer Science)*. Nemzeti Tankönyvkiadó, 2003 (2. kiadás). 959

[87] A. Deutsch, M. Fernandez, D. Suciu. Storing semistructured data with stored. In *Proceedings of SIGMOD'99*, 431–442. o., 1999. 1483

[88] A. Deutsch, L. Popa, D. Tannen. Physical data independence, constraints and optimization with universal plans. In *Proceedings of VLDB'99*, 459–470. o., 1999. 1483

[89] A. Deza, E. Nematollahi, R. Peyghami, T. Terlaky. The central path visits all the vertices of the Klee–Minty cube, 2004. AdvOl-Report#2004/11, McMaster University, Advanced Optimization Laboratory. 1296

[90] P. Dömösi, A. Fazekas, G. Horváth, Z. Mecsei. Formális nyelvek és automaták (Formal Languages and Automaton). Digitális kézirat, 2004. 959, 1010

[91] R. Dobrushin, S. Ortyukov. Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements. *Problems of Information Transmission (translated from Russian)*, 13(1):59–65, 1977. 1052, 1053

[92] R. Dobrushin, S. Ortyukov. Upper bound for the redundancy of self-correcting arrangements of unreliable elements. *Problems of Information Transmission (translated from Russian)*, 13(3):201–208, 1977. 1052, 1053

[93] D. Dolev, R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. 1166

[94] D. R. Dooly, S. A. Goldman, S. D. Scott. On-line analysis of the TCP acknowledgement delay problem. *Journal of the ACM*, 48:243–273, 2001. 1382, 1383

[95] Gy. Dósa, Y. He. Better online algorithms for scheduling with machine cost. *SIAM Journal on Computing*, 33(5):1035–1051, 2004. 1383

[96] D.-Z. Du, K-I. Ko. *Problem Solving in Automata, Languages, and Complexity*. John Wiley & Sons, 2001. 959

[97] O. Duschka, M. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 109–116. o. ACM-Press, 1997. 1483

[98] O. Duschka, M. Genesereth. Query planning in infomaster. In *Proceedings of ACM Symposium on Applied Computing*, 109–111. o. ACM-Press, 1997. 1483

[99] M. Ester, H. Kriegel, J. Sander, X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 226–231. o., 1996. AAAI Press. 1435

[100] G. Falin, J. Templeton. *Retrial Queues*. Chapman, 1997. 1345

[101] Gy. Farkas. A Fourier-féle mechanikai elv alkalmazásai. *Mathematikai és Természettudományi Értesítő*, 12:457–472, 1894. 1294

[102] Gy. Farkas. Paraméteres módszer fourier mechanikai elvéhez. *Mathematikai és Physikai Lapok*, 7:63–71, 1898. 1294

[103] Gy. Farkas. Theorie der einfachen ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124:1–27, 1898. 1294

[104] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, 1968, 3. kiadás (Magyarul: *Bevezetés a valószínűségszámításba*, Műszaki Könyvkiadó, 1978). 1346

[105] A. Fiat, Y. Rabani, Y. Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48:410–428, 1994. 1382

[106] A. Fiat, G. Woeginger (szerkesztők). *Online Algorithms. The State of Art*. Springer-Verlag, 1998. 1382

[107] C. N. Fischer, R. LeBlanc (szerkesztők). *Crafting a Compiler*. The Benjamin/Cummings Publishing Company, 1988. 1010

[108] M. J. Fischer, N. A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, 1982. 1166

[109] M. J. Fischer, N. A. Lynch, M. S. Paterson. Impossibility of distributed consensus with one faulty proces. *Journal of the ACM*, 32(2):374–382, 1985. 1166

[110] P. Flach. *Simply Logical: Intelligent Reasoning by Example (Wiley Series in Probability and Mathematical Statistics)*. Wiley & Sons, 1994 (magyarul: *Logikai programozás*, Panem, 2001). 1483

[111] R. Fleischer, M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000. 1382

[112] D. Florescu, A. Halevy, A. O. Mendelzon. Database techniques for the world-wide web: a survey. *SIGMOD Record*, 27(3):59–74, 1998. 1483

[113] D. Florescu, L. Raschid, P. Valduriez. Answering queries using oql view expressions. In *Workshop on Materialized views, in cooperation with ACM SIGMOD*, 627–638. o., 1996. 1483

[114] D. D. Florescu. *Search spaces for object-oriented query optimization*. PhD thesis, University of Paris VI, 1996. 1483

[115] F. Floris, B. Goethals, J. Bussche. A tight upper bound on the number of candidate patterns. In *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM '01)*, 155–162. o. IEEE Computer Society, 2001. 1407

[116] Z. Fülöp. *Formális nyelvek és szintaktikus elemzésük (Formal Languages and their Syntactical Analysis)*. Polygon, 2004 (második kiadás). 959, 1010

[117] C.-H. Foh, M. Zukerman. Performance analysis of the IEEE 802.11 MAC protocol. In *Proceedings of European Wireless*, 2002. 1346

[118] É. Fouvry. Théoréme de Brun-Titchmarsh: application au théoréme de Fermat. *Inventiones Mathematicae*, 79:383–407, 1985. 1113

[119] E. Fried. *Algebra I.* Nemzeti Tankönyvkiadó, 2000. 891

[120] E. Fried. *Algebra II.* Nemzeti Tankönyvkiadó, 2002. 891

[121] M. Friedman, D. S. Weld. Efficient execution of information gathering plans. In *Proceedings International Joint Conference on Artificial Intelligence*, 785–791. o., 1997. 1483

[122] R. Frisch. The logaritmic potential method for solving linear programming problems. Memorandum, University Institute of Economics, Oslo, 1955. 1295

[123] A. Fóthi, Z. Horváth. *Bevezetés a programozásba (Introduction to Programming)*. ELTE Informatikai Kar, 2005. Digitális tankönyv. 833

[124] I. Gaál. *Diophantine Equations and Power Integral Bases: New Computational Methods*. Birkhäuser Boston, 2002. 892

[125] R. Gallager. *Low-density Parity-check Codes*. The MIT Press, 1963. 1052

[126] H. Garcia-Molina, J. Seiferas. Elections in a distributed computing systems. *IEEE Transactions on Computers*, C-31(1):47–59, 1982. 1166

[127] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 832, 1523

[128] J. Gathen, von zur, J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999. 891, 892, 1113

[129] F. Gécseg, I. Peák. *Algebraic Theory of Automata*. Akadémiai Kiadó, 1972. 958

[130] P. Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1–2):45–267, 2001. See also www.arXiv.org/abs/math.PR/0003117 and the proceedings of the 1997 Symposium on the Theory of Computing. 1053

[131] P. Gács, A. Gál. Lower bounds for the complexity of reliable Boolean circuits with noisy gates. *IEEE Transactions on Information Theory*, 40(2):579–583, 1994. 1052

[132] P. Gács, L. Lovász. Khachiyan's algorithm for linear programming. *Mathematical Programming Study*, 14:61–68, 1981. 1294

[133] P. Gács, J. Reif. A simple three-dimensional real-time reliable cellular array. *Journal of Computer and System Sciences*, 36(2):125–147, 1988. 1053

[134] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer-Verlag, 1998. 1345

[135] D. Giammarresi, R. Montalbano. Deterministic generalized automata. *Theoretical Computer Science*, 215(1–2):191–208, 1999. 958

[136] B. Goethals, M. J. Zaki. Advances in frequent itemset mining implementations: Introduction to fimi03. In B. Goethals, M. J. Zaki (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, *CEUR Workshop Proceedings* 90. kötete, *CEUR Workshop Proceedings* 90. kötete. 1407

[137] M. Goldfeld. On the number of primes p for which $p + a$ has a large prime factor. *Mathematika*, 16:23–27, 1969. 1113

[138] J. Goldstein, P. A. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Optimizing queries using materialized views: a practical, scalable solution*, 331–342. o., 2001. 1483

[139] G. H. Golub, C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3. kiadás, 1996. 1434

[140] J. Gondzio. Multiple centrality corrections in a primal-dual method for linear programming. *Computational Optimization and Applications*, 6:147–156, 1996. 1296

[141] J. Gondzio, T. Terlaky. A computational view of interior point methods for linear programming. In J. Beasly (szerkesztő), *Advances in Linear and Integer Programming*, 393–481. o. Oxford University Press, 1995. 1296

[142] G. Gottlob, C. Koch, R. Pichler. The complexity of XPath query evaluation. In *Proceedings of the 22nd Symposium on Principles of Database Systems*, 179–190. o., 2003. 1523

[143] R. L. Graham. Bounds for certain multiprocessor anomalies. *The Bell System Technical Journal*, 45:1563–1581, 1966. 1382

[144] G. Grahne, J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In B. Goethals, M. J. Zaki (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, *CEUR Workshop Proceedings* 90. kötete, 2003. 1407

[145] G. Grahne, A. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proceedings of ICDT'99*, *Lecture Notes in Computer Science* 1540. kötete, 332–347. o. Springer-Verlag, 1999. 1483

[146] J. N. Gray. Notes on database operating system. In R. Bayer, R. M. Graham, G. Seegmuller (szerkesztők), *Operating Systems: An Advanced Course*, *Lecture Notes in Computer Science* 60. kötete, 393–481. o. Springer-Verlag, 1978. 1166

[147] D. Gries. *Compiler Construction for Digital Computers*. John Wiley & Sons, 1971. 1010

[148] D. Grune, H. Bal, C. J. H. Jacobs, K. Langendoen. *Modern Compiler Design*. John Wiley & Sons, 2000. 1010

[149] S. Guha, R. Rastogi, K. Shim. ROCK: A robust Clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000. 1435

[150] L. Györfi, I. Páli. *Tömegkiszolgálás informatikai rendszerekben (Queueing in Computer Systems)*. Műegyetemi Kiadó, 1996. 1345, 1346

[151] A. Halevy, A. Rajaraman, J. J. Ordille, D. Srivastava. Querying heterogeneous information sources using source descriptions. In *Proceedings of Very Large Data Bases*, 251–262. o., 1996. 1483

[152] A. Halevy. Logic based techniques in data integration. In J. Minker (szerkesztő), *Logic-based Artificial Intelligence*, 575–595. o. Kluwer Academic Publishers, 2000. 1483

[153] A. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10:270–294, 2001. 1483

[154] A. Halevy, A. Mendelzon, Y. Sagiv, D. Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 95–104. o. ACM-Press, 1995. 1483

[155] J. Han, M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufman Publisher, 1972 (magyarul: *Adatbányászat – Koncepciók és technikák*. Panem, 2004). 1407, 1434

[156] J. Han, J. Pei, Y. Yin. Mining frequent patterns without candidate generation. In W. Chen, J. Naughton, P. Bernstein (szerkesztők), *2000 ACM SIGMOD International Conference on Management of Data*, 1–12. o. ACM Press, 2000. 1407

[157] M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978. 958

[158] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975. 1434

[159] T. H. Haveliwala, A. Gionis, D. Klein, P. Indyk. Evaluating strategies for Similarity search on the web. In *Proceedings of the Eleventh International World Wide Web Conference, 2002*., 2002. http://citeseer.ist.psu.edu/haveliwala02evaluating.html. 1434

[160] H. He, J. Yang. Multiresolution indexing of XML for frequent queries. In *Proceedings of the 20th International Conference on Data Engineering*, 683–694. o., 2004. 1523

[161] M. R. Henzinger, T. A. Henzinger, P. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 453–462. o. IEEE Computer Society Press, 1995. 1523

[162] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001 (németül: *Einführung in Automatentheorie, Formale Sprachen und Komplexitätstheorie*, Pearson Studium, 2002). 2. kiadás. 958

[163] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979. 958

[164] P. Huard. Resolution of mathematical programming with nonlinear constraints by the method of centers. In J. Abadie (szerkesztő), *In Nonlinear Programming*, 207–219. o. North Holland Publ. House, 1967. 1295

[165] T. W. Hungerford. *Abstract Algebra: An Introduction*. Saunders College Publishers, 1990. 891

[166] R. W. Hunter. *Compilers, Their Design and Construction using Pascal*. John Wiley & Sons, 1985. 1010

[167] L. Hunyadvári, T. Manhertz. Automaták és formális nyelvek (Automaton and Formal Languages). Digitális kézirat, 2004. december 10. 959

[168] T. Illés, J. Peng, C. Roos, T. Terlaky. A strongly polynomial rounding procedure yielding a maximally complementary solution for $P*(\kappa)$ linear complementarity problems. *SIAM Journal on Optimization*, 11:320–340, 2000. 1296

[169] T. Illés, C. Roos, T. Terlaky. Polynomial affin scaling algoritms for $p_*(\kappa)$ linear complementarity problems. In E. Sachs T. Tichatschke P. Gritzmann, R. Horst (szerkesztő), *Recent Advances in Optimization*, *Lecture Notes in Economics and Mathematical Systems* 452. kötete, 119–137. o. Springer-Verlag, 1996. 1296

[170] T. Illés, M. Nagy. Mizuno–Todd–Ye típusú prediktor-korrektor algoritmus elégséges mátrixú lineáris komplementaritási feladatokra. *Alkalmazott Matematikai Lapok*, 22:29–49, 2004. 1297

[171] Cs. Imreh. Online strip packing with modifiable boxes. *Operations Research Letters*, 66:79–86, 2001. 1383

[172] Cs. Imreh, J. Noga. Scheduling with machine cost. In *Proceedings of APPROX'99*, *Lecture Notes in Computer Science* 1540. kötete, 168–176. o., 1999. 1383

[173] A. Inokuchi, T. Washio, H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, 13–23. o. Springer-Verlag, 2000. 1407

[174] A. Iványi. *Párhuzamos algoritmusok* (*Parallel Algorithms*). ELTE Eötvös Kiadó, 2003. 831, 833

[175] A. Iványi (szerkesztő). *Informatikai algoritmusok 1*. ELTE Eötvös Kiadó, 2004. Elektronikusan: ELTE Informatikai Kar. 831

[176] A. Jain, M. Murty, P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999. 1434

[177] A. K. Jain, R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988. 1434

[178] F. Jarre. Interior point methods for classes of convex programs. In T. Terlaky (szerkesztő), *Interior Point Methods of Mathematical Programming*, 255–296. o. Kluwer Academic Publishers, 1996. 1296

[179] L. Jereb, M. Telek (szerkesztők). *Sorbanállásos rendszerek (Queing Systems)*. BME Hiradástechnikai Tanszék jegyzete, 1999. 1346

[180] D. Johnson. Near-optimal bin packing algorithms. PhD értekezés, MIT Department of Mathematics, 1973. 1382

[181] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974. 1382

[182] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, R. L. Graham. Worst-case performance-bounds for simple one-dimensional bin packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974. 1382

[183] A. Járai. Számítógépes számelmélet. ELTE Informatikai Kara, digitális jegyzet: http://compalg.inf.elte.hu/~ajarai/konyvek.html, 1998. 1114

[184] A. Járai. *Bevezetés a matematikába*. ELTE Eötvös Kiadó, 2005. 1113

[185] D. B. Judin, A. S. Nemirovski. Informational complexity and effective methods for the solution of convex extremal problems (in Russian). *Ekonomika i Matematcheskie Metody*, 12:357–369, 1976. 1294

[186] S. Karlin, M. T. Taylor. *A First Course in Stochastic Processes*. Academic Press, 1975 (magyarul: *Sztochasztikus folyamatok*, Gondolat Kiadó, 1985). 1346

[187] A. R. Karlin, C. Kenyon, D. Randall. Dynamic TCP acknowledgement and other stories about $e/(e-1)$. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 502–509. o., 2001. 1382

[188] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. 1294

[189] L. Kaufman, P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990 (2. kiadás). 1434, 1435

[190] R. Kaushik, R. Krishnamurthy, J. F. Naughton, R. Ramakrishnan. Exploiting local similarity for indexing paths in graph-structured data. In *Proceedings of the 18th International Conference on Data Engineering*, 129–140. o., 2002. 1523

[191] R. Kaushik, R. Shenoy, P. F. Bohannon, E. Gudes. On the integration of structure indexes and inverted lists. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 779–790. o., 2004. 1523

[192] R. R. Kaushik, P. Bohannon, J. F. Naughton, H. Korth. Covering indexes for branching path queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, 133–144. o., 2002. 1523

[193] R. R. Kaushik, P. Bohannon, J. F. Naughton, H. Korth, P. Shenoy. Updates for structure indexes. In *Proceedings of Very Large Data Bases*, 239–250. o., 2002. 1523

[194] N. Kayal, N. Saxena. Towards a deterministic polynomial time test. Technical report, Indian Institute of Technology Kanpur, 2002. 1113

[195] D. Kelley. *Automata and Formal Languages*. Prentice Hall, 1995. 959

[196] L. Khacijan. A polynomial time algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979. 1294

[197] M. Khosrow-Pour (szerkesztő). *Encyclopedia of Information Science and Technology, Vol. 1, Vol. 2, Vol. 3, Vol. 4, Vol. 5*. Idea Group Inc., 2005. Az ELTE IK elektronikus könyvtárában mind az 5 kötet teljes anyaga elérhető. 833, 1407, 1434, 1523

[198] J. Kim, V. Vu. Generating random regular graphs. In *Proceedings of the Thirty Fifth ACM Symposium on Theory of Computing*, 213–222. o., 2003. 1053

[199] E. Klafszky, T. Terlaky. Az ellipszoid módszerről. *Szigma*, XX(2–3):195–208, 1987–88. 1297

[200] V. Klee, G. J. Minty. How good is the simplex algorithm? In O. Shisna (szerkesztő), *Inequalities III*, 159–175. o. Academic Press, 1972. 1294

[201] J. Kleinberg. An impossibility theorem for clustering. In S. Becker, S. Thrun, K. Obermayer (szerkesztők), *Proceedings of 15th Conference Neural Information Processing Systems, Advances in Neural Information Processing Systems* 15. kötete, 2002. 1434

[202] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, 1975 (Magyarul: *Sorbanállás – kiszolgálás: Bevezetés a tömegkiszolgálási rendszerek elméletébe*, Műszaki Könyvkiadó, 1979). 1345, 1346

[203] E. Klerk, de. *Aspects of Semidefinite Programming: Interior Point Algorithms and Selected Applications*. Kluwer Academic Publisher, 2002. 1296

[204] E. Klerk, de, C. Roos, T. Terlaky. *Nemlineáris optimalizálás*. BKÁE, Operációkutatás Tanszék, 2004. *Operációkutatás sorozat*, 5. kötet. 1297

[205] D. E. Knuth. *Fundamental Algorithms, The Art of Computer Programming* 1. kötete. Addison-Wesley, 1968 (3., javított kiadás 1997. Magyarul: *A számítógép-programozás művészete. 1. kötet. Alapvető algoritmusok*, Műszaki Könyvkiadó, 1993, 2. kiadás.). 831

[206] D. E. Knuth. *Seminumerical Algorithms, The Art of Computer Programming* 2. kötete. Addison-Wesley, 1969 (3. javított kiadás 1998. Magyarul: *A számítógép-programozás művészete. 2. kötet. Szeminumerikus algoritmusok*, Műszaki Könyvkiadó, 1993, 2. kiadás.). 831

[207] D. E. Knuth. *Sorting and Searching*, *The Art of Computer Programming* 3. kötete. Addison-Wesley, 1973 (3., javított kiadás 1997. Magyarul: *A számítógép-programozás művészete. 3. kötet. Keresés és rendezés*, Műszaki Könyvkiadó, 1994, 2. kiadás.). 831

[208] M. Kojima, S. Mizuno, A. Yoshise. A primal-dual interior point algorithm for linear programming. In N. Megiddo (szerkesztő), *Progress in Mathematical Programming: Interior Point and Related Methods*, 29–47. o. Springer-Verlag, 1989. 1295

[209] M. Kojima, N. Megiddo, T. Noma, A. Yoshise. *A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems*, *Lecture Notes in Computer Science* 538. kötete. 1991. 1296

[210] E. Koutsoupias, C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42:971–983, 1995. 1382

[211] D. C. Kozen. *Automata and Computability*. Springer-Verlag, 1995. 959

[212] L. Kozma, L. Varga. *A szoftvertechnológia elméleti kérdései (Theoretical questions of software technology*. ELTE Eötvös Kiadó, 2003. 1167

[213] Z. Kása. *Formális nyelvek és automaták (Automaton and Formal Languages)*. Farkas Gyula Egyesület a Matematikáért és Informatikáért, 2004. 1010

[214] M. Kuramochi, G. Karypis. Frequent subgraph discovery. In *Proceedings of the 1st IEEE International Conference on Data Mining*, 313–320. o., 2001. 1407

[215] A. V. Kuznetsov. Information storage in a memory assembled from unreliablecomponents. *Problems of Information Transmission (translated from Russian)*, 9(3):254–264, 1973. 1052

[216] C. T. Kwok, D. Weld. Planning to gather information. In *Proceedings of AAAI 13th National Conference on Artificial Intelligence*, 32–39. o., 1996. 1483

[217] L. Lakatos. On a simple continuous cyclic-waiting system. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae. Sectio Computatorica*, 14:105–113, 1994. 1345

[218] L. Lakatos. Equilibrium distributions for the $M|G|1$ and related systems. *Publicationes Mathematicae Debrecen*, 55:123–140, 1999. 1346

[219] E. T. Lambrecht, S. Kambhampati, S. Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceedings of 16th International Joint Conference on Artificial Intelligence*, 1204–1211. o., 1999. 1483

[220] L. Lamport. A new solution of dijkstra's concurrent programming problem. *Communications of the ACM*, 18(8):453–455, 1974. 1166

[221] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computers*, 5(1):1–11, 1987. 1166

[222] L. Lamport, R. Shostak M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. 1166

[223] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computer Statistical Data Analysis*, 19(2):191–201, 1995. 1434

[224] M. V. Lawson. *Finite Automata*. Chapman & Hall/CRC, 2004. 959

[225] A. K. Lenstra, H. W. Lenstra, Jr., L. Lovász. Factoring polynomials with integer coefficients. *Mathematische Annalen*, 261:513–534, 1982. 891

[226] S. Leonardi. On-line network routing. *Lecture Notes in Computer Science* 1442. kötete, 242–267. o. Springer-Verlag, 1998. 1382

[227] R. Lidl, H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1986. 891, 1113

[228] P. Linz. *An Introduction to Formal Languages and Automata*. Jones and Barlett Publishers, 2001. 959

[229] G. Liu, H. Lu, J. X. Yu, W. Wang, X. Xiao. AFOPT: An efficient implementation of pattern growth approach. In B. Goethals, M. J. Zaki (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, *CEUR Workshop Proceedings* 90. kötete, 2003. 1407

[230] M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002. 958

[231] K.. Louden. Compilers and interpreters. In A. B. Tucker (szerkesztő), *Handbook of Computer Science*, 99/1–99/30. o. Chapman & Hall/CRC, 2004. 1010

[232] L. Lovász, P. Gács. *Algoritmusok (Algorithms)*. Műszaki Könyvkiadó és Tankönyvkiadó, 1978 és 1987. 831

[233] L. Lovász. *Combinatorial Problems and Exercises*. Akadémiai Kiadó, 1979 (Magyarul: *Kombinatorikai problémák és feladatok*, Typotex, 1999, magyarul digitálisan Typotex, 2005). 833, 1434

[234] Y. Luke. *Mathematical Functions and their Approximations*. Academic Press, 1975. 1345

[235] O. B. Lupanov. On a method of circuit synthesis. *Izvestia VUZ (Radiofizika)*, 120–140. o., 1958. 1053

[236] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufman Publisher, 2001 (Ötödik kiadás. Magyarul: *Osztott algoritmusok*. Kiskapu Kiadó, 2002). 831, 1167

[237] N. A. Lynch, M. J. Fischer. On describing the behavior and implementation of distributed systems. *Theoretical Computer Science*, 13(1):17–43, 1981. 1166

[238] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967. 1434, 1435

[239] R. Mak. *Writing Compilers and Interpreters*. Addison-Wesley, 1991. 1010

[240] M. Manasse, L. McGeoch, D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990. 1382

[241] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill Book Co., 1974 (Magyarul: *Programozáselmélet*, Műszaki Könyvkiadó, 1981). 958

[242] H. Mannila, H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, 146–151. o. AAAI Press, 1996. 1407

[243] H. Mannila, H. Toivonen, I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD'95)*, 210–215. o. AAAI Press, 1995. 1407

[244] H. Mannila, H. Toivonen, I. Verkamo. Efficient algorithms for discovering association rules. In U. M. Fayyad, R. Uthurusamy (szerkesztők), *AAAI Workshop on Knowledge Discovery in Databases(KDD-94)*, 181–192. o., 1994. AAAI Press. 1406

[245] I. Maros, Cs. Mészáros. The role of the augmented system in interior point methods. *European Journal of Operations Research*, 107(3):720–736, 1998. 1296

[246] I. Maros. *Computational Techniques of the Simplex Method*. Kluwer Academic Publishers, 2003. 1294

[247] B. McLaughlin. *Java and XML*. O'Reilly, 2000. Magyarul: *XML kézikönyv*. Szak, 2005. 1523

[248] A. Meduna. *Automata and Languages: Theory and Applications*. Springer-Verlag, 2000. 959

[249] N. Megiddo. Pathways to the optimal set in linear programming. In N. Megiddo (szerkesztő), *Progress in Mathematical Programming: Interior Point and Related Methods*, 131–158. o. Springer-Verlag, 1989. 1295

[250] T. Milo, D. Suciu. Index structures for path expressions. *Lecture Notes in Computer Science* 1540. kötete, 277–295. o. Springer-Verlag, 1999. 1523

[251] S. Mizuno, Shinji, M. Todd, Y. Ye. On adaptive-step primal-dual interior-point algorithms for linear programming. *Mathematics of Operations Research*, 18(4):964–981, 1993. 1295

[252] R. N. Moll, M. A. Arbib, A. J. Kfoury. *An Introduction to Formal Language Theory*. Springer-Verlag, 1988. 959

[253] R. Motwani, P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. 1053

[254] Cs. Mészáros. Fast Cholesky factorization for interior point methods of linear programming. *Computers and Mathematics with Applications*, 31(4–5):49–54, 1996. 1296

[255] Cs. Mészáros. The BPMPD interior point solver for convex quadratic programming problems. *Optimization Methods and Software*, 11–12:431–449, 1999. 1296

[256] S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufman Publisher, 1997. 1010

[257] T. Murata. Petri nets, properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. 1227

[258] Y. Nesterov, A. Nemirovski. *Interior Point Polynomial Algorithms in Convex Programming*, *SIAM Studies in Applied Mathematics* 13. kötete. SIAM, 1994. 1296

[259] J. Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. Princeton University Press, 1956. 1052

[260] S. Owicki, D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6(4):319–340, 1976. 1166

[261] S. Owicki, L. Lamport. An axiomatic proof technique for parallel programs i. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, 1982. 1166

[262] R. Paige, R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987. 1523

[263] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. Pruning closed itemset lattices for association rules. In *Proceedings of the BDA French Conference on Advanced Databases*, 1998. 1407

[264] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999. 1407

[265] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, 398–416. o., 1999. 1407

[266] A. Pataricza. Petri-hálók. In A. Pataricza (szerkesztő), *Formális módszerek a számítástudományban*, 31–111. o. Typotex, 2004. 1227

[267] M. Pease, R. Shostak L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980. 1166

[268] J. Pei, J. Han, R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 21–30. o., 2000. 1407

[269] I. Peák. *Bevezetés az automaták elméletébe. Az automaták mint információátalakító rendszerek 1. (Introduction to the Theory of Automata. Automata as Systems for the Transformation of the Information).* Tankönyvkiadó, 1977. 959

[270] I. Peák. *Bevezetés az automaták elméletébe. Az automaták mint felismerő rendszerek 2. (Introduction to the Theory of Automata. Automata as Systems for the Recognition).* Tankönyvkiadó, 1978. 959

[271] I. Peák. *Bevezetés az automaták elméletébe. Automaták kompozíciói. Struktúratételek 3. (Introduction to the Theory of Automata. Composition of Automata. Structure Theorems).* Tankönyvkiadó, 1980. 959

[272] D. Pelleg, A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 277–281. o., 1999. 1435

[273] J. Peng, C. Roos, T. Terlaky. *A New Paradigm for Primal-Dual Interior-Point Algorithms*. Princeton University Press, 2002. 1296

[274] J. Peng, C. Roos, T. Terlaky. New complexity analysis of primal-dual newton method for linear optimization. *Annals of Operation Research*, 99:23–39, 2000. 1296

[275] Peptool. B(PN)^2 nyelv honlapja. http://theoretica.informatik.uni-oldenburg.de/ pep/, 2005. 1227

[276] G. Peterson, J. Fischer. Economical solutions for the critical section problem in distributed systems. In *Proceedings of the 9th ACM Symposium on Theory of Computing*, 91–97. o. IEEE Computer Society Press, 1977. 1166

[277] A. Pethő. Diofantoszi egyenletek effektív és explicit megoldása. Akadémiai doktori értekezés, MTA, 1990. 892

[278] C. Petri. *Kommunikation mit Automaten*. PhD thesis, Technical University Darmstadt, 1962. 1227

[279] N. Pippenger, G. Staomulis, J. N. Tsitsiklis. On a lower bound for the redundancy of reliable networks with noisy gates. *IEEE Transactions on Information Theory*, 37(3):639–643, 1991. 1052

[280] N. Pippenger. On networks of noisy gates. In *Proceeding of the 26th IEE FOCS Symposium*, 30–38. o., 1985. 1052

[281] N. Pippenger. Analysis of error correction by majority voting. In Silvio Micali (szerkesztő), *Randomness in Computation*. JAI Press, 1989. 1053

[282] T. Pittman. *The Art of Compiler Design, Theory and Practice*. Prentice Hall, 1992. 1010

[283] I. Pólik, T. Terlaky. Belsőpontos algoritmusok önreguláris távolságfüggvénnyel. *Alkalmazott Matematikai Lapok*, 22:163–184, 2004. 1297

[284] N. Polyzotis, M. N. Garofalakis. Statistical synopses for graph-structured XML databases. In *Proceedings of the 2002 ACM SIGMOD international Conference on Management of Data*, 358–369. o., 2002. 1523

[285] R. Pottinger. MinCon: A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2):182–198, 2001. 1483

[286] R. Pottinger, A. Halevy. A scalable algorithm for answering queries using views. In *Proceedings of Very Large Data Bases'00*, 484–495. o., 2000. 1483

[287] Párhuzamos folyamatok. B(PN)^2 ábrák. http://plc.inf.elte.hu/parh_foly, 2005. 1227

[288] A. Prékopa. *Valószínűségelmélet műszaki alkalmazásokkal (Probability Theory with Technical Applications)*. Műszaki Könyvkiadó, 1962. 1345

[289] A. Prékopa. *Lineáris programozás*. Bolyai János Matematikai Társulat, 1968. 1294

[290] A. Prékopa. Az optimalizáláselmélet kialakulásának a történetéről. *Alkalmazott Matematikai Lapok*, 4:165–191, 1978. 1294

[291] A. Prékopa. Farkas Gyula élete és munkásságának jelensége az optimalizálás elméletében. In S. Komlósi, T. Szántai (szerkesztők), *Új utak a magyar operációkutatásban: In memoriam Farkas Gyula*. Dialóg Campus Kiadó, 1999. 1294

[292] X. Qian. Query folding. In *Proceedings of International Conference on Data Engineering*, 48–55. o., 1996. 1483

[293] B Rácz. Nonordfp: An FP-growth variation without rebuilding the FP-tree. In B. Goethals, M. J. Zaki (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, *CEUR Workshop Proceedings* 126. kötete, 2004. 1407

[294] R. Reischuk, B. Schmelz, B.. Reliable computation with noisy circuits and decision trees—a general $n \log n$ lower bound. In *Proceedings of the 32-nd IEEE FOCS Symposium*, 602–611. o., 1991. 1052

[295] W. Reisig. *Elements of Distributed Algorithms, Modeling and Analysis with Petri Nets*. Springer-Verlag, 1998. 1227

[296] J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming*, 40:59–93, 1988. 1295

[297] L. Rónyai, G. Ivanyos, R. Szabó. *Algoritmusok* (*Algorithms*). Typotex, 1999. 831, 1435, 1483

[298] A. Rényi. *Probability Theory*. Akadémiai Kiadó/North Holland Publ. House, 1970 (Magyarul: *Valószínűségelmélet*, Tankönyvkiadó, Budapest, 1973). 1345

[299] C. Roos, T. Terlaky, J.-P. Vial. *Theory and Algorithms for Linear Optimization: An Interior Point Approach*. John Wiley & Sons, 1997. 1295, 1296

[300] G. Rozenberg, A. Salomaa. *Handbook of Formal Languages, Vols. I–III.* Springer-Verlag, 1997. 958

[301] Gy. Révész. *Bevezetés a formális nyelvek elméletébe (Introduction to the Theory of Formal Languages)*. Akadémiai Kiadó, 1979. 959, 1010

[302] P. Rózsa. *Lineáris algebra és alkalmazásai*. Műszaki Könyvkiadó, 1991 (3. kiadás). 1434

[303] T. L. Saaty. *Elements of Queueing Theory with Applications*. McGraw-Hill Book Co., 1961. 1345

[304] A. Salomaa. *Theory of Automata*. Pergamon Press, 1969. 958

[305] A. Salomaa. *Formal Languages*. Academic Press, 1987 (Második, módosított kiadás). 958

[306] M. Saunders, J. A. Tomlin. Solving regularized linear programs using barrier methods and KKT systems. Report SOL 96-4. Dept. of EESOR, Stanford University, 1996. 1296

[307] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, IT-29(1):23–35, 1983. 1166

[308] J. Sgall. On-line scheduling. *Lecture Notes in Computer Science* 1442. kötete, 196–231. o. Springer-Verlag, 1998. 1382

[309] C. Shannon. The synthesis of two-terminal switching circuits. *The Bell Systems Technical Journal*, 28:59–98, 1949. 1053

[310] G. Shedler. *Regeneration and Networks of Queues*. Springer-Verlag, 1987. 1346

[311] D. Shmoys, J. Wein, D. P. Williamson. Scheduling parallel machines online. *SIAM Journal on Computing*, 24:1313–1331, 1995. 1383

[312] I. E. Shparlinski. *Finite Fields: Theory and Computation The Meeting Point of Number Theory, Computer Science, Coding Theory, and Cryptography*. Kluwer Academic Publishers, 1999. 891

[313] M. Simon. *Automata Theory*. World Scientific Publishing Company, 1999. 959

[314] D. A. Simovoci, R. L. Tenney. *Theory of Formal Languages with Applications*. World Scientific Publishing Company, 1999. 959

[315] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997. 958

[316] D. Sleator R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985. 1383

[317] N. P. Smart. *The Algorithmic Resolution of Diophantine Equations*, *London Mathematical Society Student Text* 41. kötete. Cambridge University Press, 1998. 892

[318] Gy. Sonnevend. Az analytic center for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In A. Prékopa, J. Szelezsán, B. Straziczky (szerkesztők), *System Modelling and Optimization: Proceedings of the 12th IFIP-Conference*, *Lecture Notes in Control and Information Sciences* 84. kötete, 866–876. o. Springer-Verlag, 1985. 1295

[319] D. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the 27th ACM STOC Symposium*, 388–397. o., 1995 (Továbbá IEEE Transactions on Information Theory 42(6):1723–1732). 1052

[320] D. Spielman. Highly fault-tolerant parallel computation. In *Proceedings of the 37th IEEE Foundations of Computer Science Symposium*, 154–163. o., 1996. 1052

[321] G. Stewart, J. Sun. *Matrix Perturbation Theory*. Academic Press, 1990. 1434

[322] D. Stiliadis. *Traffic scheduling in packet switched networks: Analysis, Design, and Implementation*. PhD thesis, University of California Santa Cruz, 1996. 1346

[323] L. Stockmeyer, A. R. Meyer. Word problems requiring exponential time. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 1–9. o. ACM Press, 1973. 1523

[324] G. Stoyan, G. Takó. *Numerikus módszerek 2*. Typotex, 2002. Digitálisan: Typotex, 2005. 833

[325] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994. 959

[326] T. A. Sudkamp. *Languages and Machines*. Addison-Wesley, 1997. 959

[327] L. Szeidl. On the estimation of moment of regenerative cycles in a general closed central-server queueing network. In *Stability problems for stochastic models*, *Lecture Notes in Mathematics* 1233. kötete, 182–189. o., 1987. 1345, 1346

[328] L. Szeidl. Estimation of the moment of the regeneration period in a closed central-server queueing network. *Theory of Probability and Applications*, 33(2):309–313, 1988. 1345, 1346

[329] J. Sztrik. Bevezetés a sorbanállási elméletbe és alkalmazásaiba (introduction to queueing theory and its applications). http://irh.inf.unideb.hu/user/jsztrik/, 2004. 1345, 1346

[330] T. Takeaki. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In B. Goethals, M. J. Zaki, R. (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, *CEUR Workshop Proceedings* 126. kötete, 2004. 1407

[331] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 2004 (4. kiadás. Magyarul: *Számítógép-hálózatok*. Panem, 2004). 1346

[332] A. S. Tanenbaum, M. van Steen. *Distributed systems. Principles and Paradigms*. Prentice Hall, 2002 (Magyarul: *Elosztott rendszerek*. Panem, 2003). 1167

[333] M. G. Taylor. Reliable information storage in memories designed from unreliable components. *The Bell Systems Technical Journal*, 47(10):2299–2337, 1968. 1052

[334] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000 (2. kiadás). 1167

[335] T. Terlaky, T. Illés. Pivot versus interior point methods: Pros and cons. *Operational Research*, 140:170–190, 2002. 1296

[336] T. Terlaky. A Karmarkar típusú algoritmusokról. *Alkalmazott Matematikai Lapok*, 15:133–162, 1990–91. 1297

[337] T. Terlaky. A belsőpontos módszerek elméletének egy elemi tárgyalása. In S. Komlósi, T. Szántai (szerkesztők), *Új utak a magyar operációkutatásban: In memoriam Farkas Gyula*, 97–131. o., 1999. Dialóg Campus Kiadó. 1297

[338] L. Thieme. Algorithmic features of Eclat. In B. Goethals, M. J. Zaki, R. (szerkesztők), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, *CEUR Workshop Proceedings* 126. kötete, 2004. 1407

[339] J. Tomkó. *A Markov folyamatok elmélete és néhány operációkutatási vonatkozása (Theory of Markov Processes and some Related Aspects in Operation Research)*. Bolyai János Matematikai Társulat, 1968. 1345

[340] J-P. Tremblay, P. G. Sorenson. *Compiler Writing*. McGraw-Hill Book Co., 1985. 1010

[341] O. G. Tsatalos, M. C. Solomon, Y. Ioannidis. The GMAP: a versatile tool for physical data independence. *The VLDB Journal*, 5(2):101–118, 1996. 1483

[342] A. Tucker. *Handbook of Computer Science*. Chapman & Hall/CRC, 2004. 1523

[343] J. D. Ullman. Information integration using logical views. In *Proceedings of ICDT'97*, *Lecture Notes in Computer Science* 1186. kötete, 19–40. o. Springer-Verlag, 1997. 1483

[344] J. D. Ullman, J. Widom. *A First Course in Database Systems*. Prentice Hall, 1997 (Magyarul: *Adatbázis-rendszerek. Alapvetés*. Panem, Budapest, 1998). 1483

[345] A. van Vliet. Lower and upper bounds for on-line bin packing and scheduling heuristics. PhD értekezés, Erasmus University, Rotterdam, 1995. 1382

[346] A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Information Processing Letters*, 43:277–284, 1992. 1382

[347] L. Varga. *Rendszerprogramok elmélete és gyakorlata (Theory and Practice of System Programs)*. Akadémiai Kiadó, 1980. 1010

[348] V. Vianu. A Web Odyssey: from Codd to XML. In *Proceedings of the 20th Symposium on Principles of Database Systems*, 1–15. o., 2001. 1522

[349] W. Waite, G. Goos. *Compiler Construction*. Springer-Verlag, 1984. 1010

[350] J. Wang, J. Han, J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, 2003. 1407

[351] J. Wang (szerkesztő). *Encyclopedia of Data Warehousing and Mining, Vol. 1, Vol. 2*. Idea Group Inc., 2005. 1434

[352] H. Wolkowicz, R. Saigal, L. Vanderberhe (szerkesztők). *Handbook of Semidefinite Programming*. Kluwer Academic Publishers, 2000. 1297

[353] H. Z. Yang, P. A. Larson. Query transformation for PSJ-queries. In *Proceedings of Very Large Data Bases'87*, 245–254. o., 1987. 1483

[354] J. Yang, K., Karlapalem, Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of Very Large Data Bases'97*, 136–145. o., 1997. 1483

[355] A. C. C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27:207–227, 1980. 1383

[356] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. John Wiley & Sons, 1997. 1295, 1296

[357] Y. Ye, M. J. Todd, S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994. 1295

[358] K. Yi, H. He, I. Stanoi, J. Yang. Incremental maintenance of XML structural indexes. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 491–502. o., 2004. 1523

[359] N. Young. On-line file caching. *Algorithmica*, 33:371–383, 2002. 1382

[360] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, M. Urata. Answering complex SQL queries using automatic summary tables. In *Proceedings of SIGMOD'00*, 105–116. o., 2000. 1483

[361] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 71–80. o., 2002. ACM Press. 1407

[362] M. J. Zaki, C. Hsiao. Charm: an efficient algorithm for closed association rule mining. Technical report, RPI, 1999. 1407

[363] M. J. Zaki, M. Ogihara. Theoretical foundations of association rules. In *Proceedings of 3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'98)*, 1998. 1407

[364] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li. New algorithms for fast discovery of association rules. In D. Heckerman, H. Mannila, D. Pregibon, R. Uthurusamy, M. Park (szerkesztők), *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 283–296. o. AAAI Press, 1997. 1407

[365] X. J. Zhu, J. Peng, T. Terlaky, G. Zhang. On implementing self-regular proximity based feasible IPMs, 2004. AdvOl-Report#2004/3, McMaster University, Advanced Optimization Laboratory. 1295

[366] A. Vestjens. On-line machine scheduling, 1997. PhD értekezés, Eindhoven University of Technology. 1383

Tárgymutató

Névmutató

1. KÖTET TARTALOMJEGYZÉKE

Informatikai könyvek

Ebben a fejezetben az ELTE Eötvös Kiadó informatikai könyveinek tartalmát mutatjuk be.

Bevezetés a matematikába

Járai Antal (szerkesztő), Farkas Gábor, Fülöp Ágnes, Gonda János, Kovács Attila, Láng Csabáné, Székely Jenő: *Bevezetés a matematikába.* ELTE Eötvös Kiadó, Budapest, 2005. ISBN 963 463 729 9, 241 oldal.

Tartalomjegyzék

Párhuzamos algoritmusok

Iványi Antal: *Párhuzamos algoritmusok*. ELTE Eötvös Kiadó, Budapest, 2003. ISBN963 463 590 3, 335 oldal. Elektronikusan: ELTE IK, Budapest, 2005: http://elek.inf.elte.hu/.

Tartalomjegyzék

A szoftvertechnológia elméleti kérdései

Kozma László, Varga László: *A szoftvertechnológia elméleti kérdései*. ELTE Eötvös Kiadó, Budapest, 2003. ISBN 963 463 648 9, 370 oldal.

Tartalomjegyzék

Szoftvertechnológia és UML

Sike Sándor, Varga László: *Szoftvertechnológia és UML*. ELTE Eötvös Kiadó, Budapest, 2003 (második kiadás). ISBN 963 463 587 3, 352 oldal.

Tartalomjegyzék

Ipari és egyetemi ismertetők

Ebben a fejezetben a hazai informatikai felsőoktatás főszereplőit és ipari kapcsolataikat bemutató anyagokat gyűjtöttünk össze.

Először az **AITIA, Multiráció, NOKIA**, **SIEMENS PSE** és **Tateyama** cégek, majd a kolozsvári **Babes-Bolyai Tudományegyetem,** a **Budapesti Műszaki és Gazdaságtudományi Egyetem,** a **Budapesti Műszaki Főiskola,** a **Debreceni Egyetem,** az **Eötvös Loránd Tudományegyetem,** az **Eszterházy Károly Főiskola,** a **Pécsi Tudományegyetem** és a **Szegedi Tudományegyetem** ismertetője következik.

A magyar informatikai felsőoktatást is lényegesen átalakítja az alap (BSc), mester (MSc) és doktori (PhD) szintekből álló **lineáris képzés** bevezetése – az 1999-ben megkötött **Bolognai egyezmény** alapján

Az egyezménynek megfelelő alapszakok akkreditálására először 2004-ben került sor: a Debreceni Egyetem javaslatára a **programtervező informatikus,** a BME és a BMF közös javaslatára pedig a **mérnök-informatikus** szak megindítására nyílt lehetőség.

A programtervező informatikus szakon 2004-ben Debrecenben 99 hallgató kezdte meg tanulmányait, 98 pont volt a ponthatár. 2005-ben a DE (48 hely) mellett már az EKF (40 hely), az ELTE (180 hely), a PPKE (150 hely), a PTE (50 hely) és a SZTE (20 hely) is meghirdette a programtervező informatikus szakot. Így a felvehető – államilag támogatott, nappali tagozatos – hallgatók száma 488.

A mérnök-informatikus szakra 2004-ben a BMF 309, a Széchenyi István Egyetem 271 és a Veszprémi Egyetem 114 hallgatót vett fel. A ponthatár az államilag támogatott nappali képzésben a Budapesti Műszaki Főiskolán 102 pont, a Veszprémi Egyetemen 100 pont, Győrben 79 pont volt. 2005-ben a BMF (315 hely), a Széchenyi István Egyetem (252 hely) és a Veszprémi Egyetem (60 hely) mellett a BME (414 hely), a DE (70 hely), a ME (95 hely) és a SZTE (100 hely) is meghirdette a mérnök-informatikus szakot, melyre összesen 1306 hallgató vehető fel.

A BCE és a SZE közös javaslatára 2005-ben a gazdasági informatikus szakot is akkreditálták. A szakot 2005-ben a Budapesti Corvinus Egyetem (100 hely), a Nyugat-Magyarországi Egyetem (18 hely), a Széchenyi István Egyetem (27 fő), a Szegedi Tudományegyetem (20 fő) és a Veszprémi Egyetem (50 hely) hirdette meg. A keretszámok összege 215.

2005-ben országosan körülbelül 30 szakon és 100 helyen indul alapképzés – ami azt jelenti, hogy a bolognai egyezmény már a hazai felsőoktatásnak körülbelül az egyharmad részére kiterjed.

Az informatikai felsőoktatás többek között annak köszönheti, hogy minden más szakterületet legalább egy évvel megelőzve megkezdhette a bolognai egyezménynek megfelelő eurokonform képzést, hogy

- az **ELTE,** valamint a **DE** (akkor KLTE) és a **SZTE** (akkor JATE) már 1972-ben elindította a programozó matematikus és programtervező matematikus szakokból álló kétlépcsős informatikus képzést (az első évben 50 + 25 + 25 hallgatóval);

- 1994-ben Nagy-Britanniában a BMF (akkor KKMF) által gondozott főiskolai műszaki informatika szakot a brit BEng, a BME által gondozott egyetemi műszaki informatika szakot pedig a brit MEng szakkal egyenértékű szakként akkreditálták;

- az informatikai felsőoktatás képviselői a Neumann János Társaság **Felsőoktatási Szakosztályának** keretében folyamatosan egyeztetik terveiket, kölcsönösen támogatják egymást;

- az informatikai felsőoktatás képviselői az **Informatika a felsőoktatásban** című konferencián rendszeresen találkoznak, és a 2002. augusztus 18-án a konferencián elhangzott – **Első gondolatok a két ciklusú műszaki informatika képzésről** című – előadás hatására kibontakozó vita során gyorsan megegyeztek és közös javaslatot tettek (a következő konferencia 2005. augusztus 24–26-ig lesz Debrecenben, a 2005. április 30-i jelentkezési határidőig a résztvevők 252 előadással jelentkeztek).

Lázasan folyik az új informatikai mesterképzés (MSc) akkreditálásának előkészítése. Az a terv körvonalazódik, hogy az alapszakokhoz illeszkedően 3 mesterszak jön létre, és ezeken belül induló szakirányok (mesterszakonként 3–6) biztosítják a szűkebb szakterületek specialistáinak felkészítését. A mesterképzés először várhatóan 2006-ban indul néhány egyetemen, majd 2007-től további egyetemek is csatlakoznak.

A következő táblázatban összefoglaljuk a 2005/2006-os tanévben államilag támogatott informatikai alapszakokra felvehető nappali tagozatos hallgatók keretszámait.

Int.	Gazdasági inf.	Mérnök-inf.	Programtervező inf.	Összesen
BCE	100	–	–	100
BME	–	414	–	414
BMF	–	315	–	315
DE	–	70	48	48
EKF	–	–	40	40
ELTE	–	–	180	180
ME	–	95	–	95
NyME	18	–	–	18
PPKE	–	–	150	150
PTE	–	–	50	50
SZE	27	252	–	279
SZTE	20	100	20	140
VE	50	60		110
Összesen	215	1306	488	2009

Az új képzés mellett több helyen indulnak (valószínűleg utoljára) a korábbi szakok is. Ezek adatait összesíti a következő táblázat.

Int.	Gazd. inf., gazd. pr.mat.	Főisk. műsz. inf.+ e.m.i.	Pr.mat.+pr.terv.mat.	Összesen
BDF	–	30 + —	–	30
BME	–	25 + –	–	25
BMF	104	–	–	104
DE	–	–	– + 100	100
ELTE	–	–	– + 180	180
GDF	–	380 + –	–	380
ME	30	–	–	30
NyF	18	–	27 + –	27
PTE	–	285 + –	–	285
SZTE	55	– + 100	– + 100	255
VE	–	–	30 + –	30
Összesen	207	820	437	1464

AITIA Informatikai Rt.

1117 Budapest, Infopark sétány 1.

Tel: (1) 382-7580, Fax: (1) 382-7581

Email: **info@aitia.ai**

Honlap: **www.aitia.ai/**

Az AITIA mozaikszóként az **Artificial Intelligence** (Mesterséges Intelligencia), **Information Technology** (Információ Technológia) és az **Intelligent Agents** (Intelligens Ágensek) kifejezéseket foglalja magában. Ezek az AITIA Informatikai Rt. legfontosabb fejlesztési területei.

Célunk olyan, nagy szaktudást igénylő kutatás-fejlesztési, illetve innovatív feladatok elvégzése, melyben a cég munkatársainak – nemzetközi szinten is – kiemelkedő szaktudása érvényesülhet. Cégünk eredményei kiterjednek az **internettel, a telekommunikációval, a beszédtechnológiával és a mesterséges intelligenciával** kapcsolatos területekre.

Az AITIA rendszermegoldásai jelen vannak a magyar és nemzetközi szoftveripar, és távközlési ipar területein – pl. a nagy magyarországi mobilszolgáltató cégek mindegyike használja az AITIA Rt. fejlesztéseit.

Az AITIA Rt. 2002. decemberétől van jelen a magyar informatikai piacon. Jogelődje 1999-ben alakult Budapesten azzal a céllal, hogy a magyarországi kutatási-fejlesztési tevékenységekre összpontosítva XXI. századi termékekkel és innovatív eredményekkel járuljon hozzá az információs társadalom kialakulásához, fejlődéséhez. 2004. végén az Egyesült Államokban megalakult az **AITIA International Inc.,** melynek segítségével fejlesztéseink az amerikai piacra is eljuthatnak.

Az AITIA Rt. pályázati támogatásra és saját kutatási költségvetésére támaszkodva folytat alkalmazott informatikai kutatásokat. Folyamatosan és eredményesen részt veszünk az **Európai Uniós és hazai K+F pályázatokon, neves egyetemekkel (BME, ELTE)** és kutatóintézetekkel együttműködve.

Cégünk **stratégiai együttműködésre** törekszik a vezető magyar egyetemekkel. Számos közös projekt indult, melyekben az egyetemek és az AITIA Rt. konzorciumot alakított. Az AITIA Rt. szoros kapcsolatokat ápol több külföldi egyetemmel-kutatóintézettel is, annak érdekében, hogy a cég a high-tech megoldások alkalmazásával versenyelőnyét fenntartsa.

Az AITIA Rt. célja távközléshez, internethez és mesterséges intelligenciához kötődő kutatási eredmények létrehozása, illetve azok ipari és üzleti hasznosítása. Kutatási eredményeinket nagyrészt a termékfejlesztésben hasznosítjuk. Alapkutatási eredményeinket a cég tagjai a szakmai felsőoktatásban is alkalmazzák, kutatóink nemzetközi hírű egyetemeken ismertetik megoldásainkat és eredményeinket.

Budapest, 2005. február 24.

Tatai Gábor

MultiRáció Kft. és a MagyarOffice

MultiRáció Kft.

A társaságot magánszemélyek hozták létre 1992 őszén pénzügyi, gazdaságinformatikai elemző rendszerek (idősorok, előrejelzések) adaptálására, tervezésére, fejlesztésére és üzemeltetésére. Olyan feladatok, problémák megoldását is képes elvégezni, amelyek speciális ismereteket, például speciális informatikai vagy pénzügyi-számviteli, vagy akár matematikai, statisztikai felkészültséget, ismeretet igényelnek. A működési kört magas szintű webes szolgáltatások mellett 2001-től „dobozos" irodai szoftver fejlesztése is kiegészíti.

Az integrált irodai alkalmazások piaca

Magas munkakultúrájú országokban a vállalkozások és intézmények több, funkcionálisan egyenértékű, a felhasználóval az anyanyelvén kommunikáló irodai szoftver közül választhatnak a piaci kínálatból. Magyarországon a helyzet egyoldalú volt. A projekt ezen kívánt változtatni azzal, hogy magyar nyelven kommunikáló, a monopoltermékkel funkcionálisan egyenértékű (de akár a hazai igényeket annál jobban kielégítő) irodai szoftverrel bővíti a kínálatot. Közben az egy munkaállomás kialakítására vonatkozó szoftver/hardver beszerzési árhányadost jelentősen 1 alá szorítja!

MagyarOffice

A MagyarOffice integrált irodai szoftver átfogó megoldást nyújt mind állami intézmények, mind vállalkozások, mind magánszemélyek számára. Tartalmaz szöveg- és weblap-szerkesztőt, táblázatkezelőt, bemutató-készítőt és rajzolóprogramot, a magyar felhasználók igényeihez szabott sablonokkal, helyesírás ellenőrzővel, dokumentációval. Alkalmas sok más programcsomag fájljainak kezelésére, beleértve például a Word, Excel és PowerPoint fájlokat. Nemcsak Windows, hanem Linux és Solaris operációs rendszeren is futtatható, így kényelmes megoldást jelenthet a teljes egészében nyílt kódú programokra épülő rendszerek számára is.

A MagyarOffice-projekt: sikertörténet

A projekt reprezentatív (vállalati) mintán lefolytatott piackutatással indult 2001. nyarán, s az kedvező fogadtatást ígért egy magas funkcionalitású és kedvező árú szoftvernek. Mintegy 30 fős fejlesztő csapattal gyors ütemben megindult a fejlesztés, és az első változat már 2001. decemberében piacra került. 2004. decemberéig a megjelent 4 termékváltozatból a több mint 300 viszonteladó partner közreműködésével több mint 40.000 licenc talált gazdára. A Magyar Innovációs Alapítvány által kiírt XI., 2002. évi pályázaton a MultiRáció Kft. „A MagyarOffice irodai szoftvercsalád kifejlesztése és piaci bevezetése" című pályázata a Budapesti Kereskedelmi és Iparkamara Innovációs Díját nyerte el.

A termék tervezett jövőbeli jellemzői

Társaságunk az ELTE Informatikai Karával és a Szegedi Tudományegyetem Informatika Tanszékcsoport Szoftverfejlesztés Tanszékével együttműködve a következő két évben több mint 300 millió forintot fordít termékfejlesztésre, amely forrásnak a felét EU-s GVOP-s pályázati források biztosítják. Két fő fejlesztési irányt jelöltünk ki:

1. Az alap forráskód, az OpenOffice.org minőségbiztosítása: szoftverminőség-ellenőrző eljárások segítségével a bizonytalanságot, instabilitást eredményezhető kódrészletek feltárása, azonosítása után ezek javítása. Ezzel stabilabb, megbízhatóbb termék áll elő.

2. A szoftver grafikus felhasználói felületének tovább fejlesztése adaptivitásának megerősítésével.

SIEMENS **Siemens PSE Program- és Rendszerfejlesztő Kft.**

2003. október 1-jétől Siemens PSE Program- és Rendszerfejlesztő Kft., röviden: Siemens PSE Kft. elnevezéssel folytatja tevékenységét a Sysdata Kft. A vállalat a Siemens közép-európai vállalatcsoportjának, a **Program and System Engineering (PSE) csoport**nak a magyarországi képviselője. A tíz évvel ezelőtt alapított leányvállalat ma a legnagyobb létszámú és az egyik legsikeresebb hazai szoftverfejlesztő vállalkozás Magyarországon. A vállalat a névváltoztatással erősíteni és tudatosítani kívánja a kiemelkedően sikeres magyarországi leányvállalat Siemenshez tartozását, ugyanakkor fokozottabb támogatást kíván nyújtani a Siemens Rt. magyarországi üzleti tevékenységéhez.

A több mint 40 évvel ezelőtt alapított Siemens PSE jelenleg az Európa hét országában foglalkoztatott 5000 szakemberével évente mintegy 450 millió eurós forgalmat realizál. A Siemens PSE 2003-ig **Sysdata** néven, majd 2003. október 1-jétől Siemens PSE Kft-ként folytat kutatási és fejlesztési tevékenységet. A létszám az elmúlt évtizedben több mint tízszeresére növekedett, és jelenleg megközelíti az 500-at.

A fejlesztések legnagyobb részét a Siemens PSE hálózatában elvégzett **szoftverfejlesztési és kutatás-fejlesztési feladatok** jelentik. A magyarországi tevékenység között említhető számos telekommunikációs szoftver fejlesztése és karbantartása: Kiemelendő a Siemens intelligens hálózati (IN) alkalmazásai, valamint a **mobiltelefon-hálózatok bázisállomásait menedzselő rendszer** fejlesztése. Folyamatos megrendelést jelent a Siemens által fejlesztett távközlési berendezések és eszközök szoftvereinek biztosítása, például a Siemens vezetékes telefonközpontjához és telefonalközpontjához történő szoftverek fejlesztése. Növekvő arányt jelent a Siemens mobiltelefonjait működtető szoftverek fejlesztése. A fenti feladatok mellett egyre nagyobb jelentőséget kapnak a vállalati szektor számára készülő megoldások és az Internet-technológiára alapuló fejlesztések.

A világ sok országában hasznosítják a Siemens PSE magyar mérnökeinek munkáját, amely az utóbbi években egyre több magyarországi projektben is megjelenik. 2003-tól például milliók kerültek kapcsolatba a Matáv számára kifejlesztett automatizált **távszavazási rendszer**rel. Az Easy-Voting elnevezésű kiértékelő szoftvert a Matáv a kereskedelmi televíziók interaktív műsorainál, például a valóság-show-k és más, a nézők pillanatnyi véleményét tükröző műsoraihoz beérkezett szavazatok rögzítésére, kiértékelésére és a csatornákon keresztül a nézők azonnali tájékoztatására használja.

Technológiai értelemben a Siemens PSE a világ élvonalához tartozik: Siemens szoftverfejlesztési módszertanok használatával, Unix és Windows környezetben, elsősorban JAVA, C++ és C programnyelveken folynak a fejlesztések. Az objektum orientált- és Webes technológiák mellett egyre több IP alapú fejlesztést végzünk.

A Siemens PSE Magyarországon **2005-ben 50-70 új munkatárs**at keres szoftverfejlesztői, IT tanácsadói és más speciális munkakörökbe. Az aktuális álláshirdetések a vállalat Internet oldalán találhatók meg: **www.pse.siemens.hu/allas.**

Siemens PSE Kft. Személyzeti osztály Fax: (+36-1) 471-3011 palyazatok.pse@siemens.com
www.pse.siemens.hu/allas

1111 Budapest, Zenta u. 1.
Tel.: (1) 361 0344, Fax: (1) 361 4469
E-mail: **info@tateyama.hu**
Honlap: **http://www.tateyama.hu/**

A Tateyama Magyar Laboratórium Kft-t 1997-ben alapította a japán **TATEYAMA KAGAKU** csoport. A cég célja a magyar innovatív technológiák támogatása és kiaknázása japán technológia és irányítás segítségével. A **Tateyama Kft.** széles körben együttműködik több hazai egyetemmel, ösztöndíjakkal támogatja a tehetséges hallgatók, és tanáraik munkáját (pl. Budapesti Műszaki és Gazdaságtudományi Egyetem, Eötvös Loránd Tudományegyetem, Veszprémi Egyetem).

A **Tateyama Kft.** főleg japán piacra dolgozik. Megrendelőink közé tartozik a Sony és Matsushita Electric, és az európai piacra is nyitottak vagyunk.

Jelenleg a következő területeken dolgozunk:

- Kliens-szerver rendszerek, irodai alkalmazások, adatbázis alkalmazások tervezése és kivitelezése.
- Felhasználói és programozói felületek fejlesztése ipari robotokhoz, különös tekintettel felületszerelő gépekre, valamint PC alapú PLC/NC vezérlőre.
- Alkalmazások fejlesztése egy magyar találmányra, a 360 fokban látó PAL360 lencsére. Kamerás biztonsági megfigyelő és rögzítő rendszerek. PTZ és DOM kamerák vezérlése.
- Mozgásdetektálás, mozgáselemzés. Rendszámok felismerése, rögzítése. Járművek színének, típusának felismerése. Arcdetektálás, követés és felismerés. Alkalmazásfejlesztés e technológiák köré.

A Budapesti Műszaki és Gazdaságtudományi Egyetemről folyamatosan fogadjuk a laborgyakorlatukat nálunk letölteni kívánó hallgatókat. Az Eötvös Loránd Tudományegyetemről már fogadtunk ösztöndíjas hallgatókat, akik leginkább a következő témakörökben jeleskedtek:

- arcfelismerés,
- rendszámfelismerés,
- PAL360 lencséhez kapcsolódó algoritmusok.

Továbbiakban is várjuk azoknak az erős matematikai képességekkel rendelkező, a digitális képelemzés területén jártas hallgatóknak a jelentkezését, akik szeretnék tudásukat a gyakorlatban is alkalmazni.

Budapest, 2005. február 25.

BABEŞ-BOLYAI TUDOMÁNYEGYETEM
MATEMATIKAI ÉS INFORMATIKAI KAR

RO 400084 Cluj-Napoca, Str. Kogalniceanu 1
Tel: +40-264-405327, Fax: +40-264-591906,
Email: **maths@math.ubbcluj.ro**
Honlap: http://www.cs.ubbcluj.ro

A kolozsvári **BBTE Matematikai és Informatikai Karán** a következő szakokon folyik alapképzés: **matematika** (román és magyar nyelven), **informatika** (román, magyar és angol nyelven), **matematika-informatika** (román, magyar és német nyelven), valamint **alkalmazott matematika** (román nyelven). A magyar tagozaton évente mintegy 80–100 hallgató kezdi el tanulmányait (matematika szakon 10–15, informatika szakon 40–50, matematika-informatika szakon 30-40). Ebből 70 körüli az államilag támogatott helyek száma. Azok a hallgatók, akik elvégzik a pedagógiai modult is, tanári diplomát is szerezhetnek.

A magyar tagozaton a hallgatók minden tárgyat tanulhatnak magyarul, de a választható tárgyak esetében (amelyek az egyes tagozatokon különbözhetnek) választhatnak más nyelvű tárgyakat is.

A karon mesterképzés (MSc) is folyik román, magyar és angol nyelven. Magyar nyelven **számítógépes matematika** szak létezik. 2005-től bevezetjük a bolognai egyezménynek megfelelő 3+2-es rendszert, azaz a 3 éves alapképzést és a 2 éves mesterképzést. Jelenleg az alapképzés 4 év, a mesterképzés pedig további 1 év.

A magyar tagozat oktatói igyekeznek magyar nyelvű szakkönyveket is beszerezni a kari könyvtár számára. Így sikerült néhány példányt szerezni az *Algoritmusok* (Műszaki Könyvkiadó, 1997), az *Osztott algoritmusok* (Kiskapu Kiadó, 2002), a *Párhuzamos algoritmusok* (ELTE Eötvös Kiadó, 2003) és az *Új algoritmusok* (Scolar Kiadó, 2003) című tankönyvekből is.

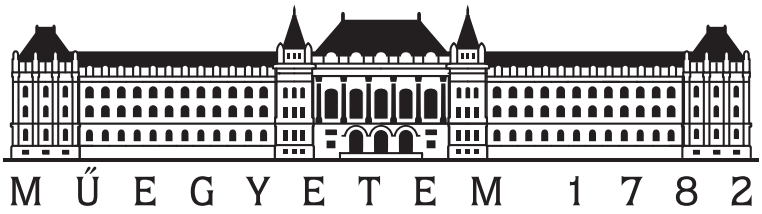
Az *Informatikai algoritmusok* című tankönyvet az informatikus alapképzés következő tárgyaihoz fogjuk használni: Algoritmika, Programozás, Adatszerkezetek, Algoritmusok elemzése és bonyolultsága, Numerikus módszerek, Operációkutatás, Számítógép-architektúrák, Operációs rendszerek, Mesterséges intelligencia, Az adatbázisok alapjai és Számítógépes grafika.

Kolozsvár, 2004. augusztus 18.

Dr. Kása Zoltán

egyetemi tanár

dékánhelyettes



BUDAPESTI MŰSZAKI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR

1117 Budapest, Magyar Tudósok krt. 2.
Tel: 463-1501, Fax: 463-3580
Email: **vikdhvez@vdk.bme.hu**

A **BME Villamosmérnöki és Informatikai Kar** alaptevékenységének meghatározó eleme az egyetemi mérnökképzés és az erre épülő doktori (PhD) képzés villamosmérnöki és műszaki informatika szakon. A mérnökképzésben évfolyamonként közel 500 villamosmérnöki, és nagyjából ugyanennyi informatikus hallgató vesz részt. A két egyetemi szintű szak oktatására az erős elméleti alapképzés, a mély szakmai ismereteket adó szakképzés, a széles területen gyakorlati tudást nyújtó laboratóriumi oktatás, és az önálló feladatok megoldására irányuló mérnöki gyakorlat a jellemző. A szakképzés során a villamosmérnöki hallgatók a beágyazott információs rendszerek, a híradástechnika, a távközlés és telematika, a számítógép technika, az irányítástechnika, a robotinformatika, valamint az elektronika és energetika területeinek valamelyikében mélyedhetnek el. Az informatikus hallgatók választható moduljai a rendszer- és szoftverfejlesztés, az infokommunikáció, a médiainformatika, az intelligens autonóm rendszerek és a gazdasági informatika.

A villamosmérnöki és informatikai szakterület doktori képzésében a hallgatók – a kar erős, nemzetközileg is elismert tudományos iskoláihoz kapcsolódva – alapkutatással és alkalmazott kutatással foglalkoznak, de emellett szervezett szakmai kurzusokon folytatják tanulmányaikat és részt vállalnak a kar oktatási tevékenységében is.

Az *Informatikai algoritmusok* című tankönyv többek között az informatika szaknak a Bevezetés a számításelméletbe, Formális nyelvek, Algoritmusok elmélete, Adatbázisok, Operációs rendszerek, Számítógépes grafika és képfeldolgozás, valamint Kódelmélet tárgyaihoz, a villamosmérnöki szaknak pedig az Informatika, Számítástudomány alapjai, Számítógépes grafika és animáció nevű tárgyaihoz használható jegyzetként, illetve hasznos kiegészítő olvasmányként.

Budapest, 2004. augusztus 16.

Dr. Arató Péter

egyetemi tanár
dékán

BUDAPESTI MŰSZAKI FŐISKOLA
NEUMANN JÁNOS
INFORMATIKAI FŐISKOLAI KAR

1034 Budapest, Bécsi út 96/b
E-mail: **titkarsag@nik.bmf.hu**
Honlap: **www.nik.bmf.hu**

A felsőoktatási reform egyik előhírnökeként Karunkon 2004 szeptemberében megindult a **Mérnök-informatikus alapképzés** (BSc képzés). Ezt a képzést Karunk igen kedvező helyzetben készíthette elő, mivel az elődintézményünk (Kandó Kálmán Villamosipari Műszaki Főiskola) által gondozott **Műszaki informatika szak** már 1994-ben nemzetközi akkreditációt szerzett, melynek értelmében a szakon szerzett oklevél egyenértékűnek minősült a brit BEng fokozattal (jó mérnöki gyakorlatot is tanúsító BSc fokozat).

Amikor Karunk az elsők között indít Mérnök-informatikus alapképzést, folytatja jogelőd intézményünk úttörő hagyományait, hiszen 1971-ben elsőként a Kandó Kálmán Villamosipari Műszaki Főiskolán indult el hazánkban Számítástechnikai szakképzés, 1987-ben az elsők között létesült Műszaki informatika szak, majd a felsőoktatási integráció során a Budapesti Műszaki Főiskolán 2000-ben elsőként létesült egy kizárólagosan informatikai profilú kar, a Neumann János Informatikai Főiskolai Kar. Így Mérnök-informatikus alapképzésünk tanterve és tantárgyprogramjai több mint harminc éves gyökerekre nyúlnak vissza.

Karunk nagy hangsúlyt helyez a képzés folyamatos korszerűsítésére annak érdekében, hogy végzett hallgatóink alapos és korszerű elméleti és gyakorlati ismereteket és kompetenciákat szerezzenek az informatika releváns területein, és képessé váljanak a diszciplína rohamos fejlődésének a követésére. E célunk megvalósítását segítik elő Kompetencia központjaink, melyeket a világ vezető informatikai cégeivel (CISCO, Intel, Microsoft, Oracle, Sun) közösen hoztunk létre, s melyek a folyamatos technológiai transzfer hatékony előmozdítói.

Mérnök-informatikus alapképzésünkben a választható szaktárgyakkal és szakirányokkal kellő teret kívánunk adni a folyamatosan megújuló, korszerű alkalmazási területek – mint például jelenleg **a vizuális programozás, az informatikai biztonság, a mobil informatika vagy az elektronikus kereskedelem** – oktatásának. A megadott szakirányok közül tanévenként a legkeresettebbek kapnak zöld utat.

Az *Informatikai algoritmusok* című tankönyv mérnök-informatikus hallgatóink értékes alapműve lesz a Programozás modul több tantárgya vonatkozásában is, így különösen az Imperatív objektumorientált programozás, a Programozási nyelvek I, II és a Haladó programozás tárgyak esetében.

Dr. Sima Dezső
egyetemi tanár, főigazgató

Budapest, 2005. március 11.

DEBRECENI EGYETEM
INFORMATIKAI KAR

4010 Debrecen, Pf. 12. E-mail: office@inf.unideb.hu
Telefon: (52) 489-100, Fax: (52) 416-857

DEBRECENI EGYETEM INFORMATIKAI KARA

A **Debreceni Egyetem Informatikai Kara** a Kelet-Magyarországi régióban az egyetlen akkreditált egyetemi szintű informatikai szakemberképzés gazdája. A kar öt tanszékén (Alkalmazott Matematika és Valószínűség-számítás, Információ Technológia, Komputergrafika és Könyvtárinformatikai, Számítógépes Rendszerek és Hálózatok, Számítógéptudomány) dolgozó 6 professzor, 13 docens (főmunkatárs), 21 adjunktus (tudományos munkatárs), 14 tanársegéd és 21 felsőfokú végzettségű számítástechnikai munkatárs jelentős, nemzetközileg is jegyzett szellemi potenciált képvisel. A Informatikai Karon jelenleg

- **programtervező informatikus** (nappali, 6 félév, alapképzési BSc szint);
- **programozó matematikus** (esti, 8 félév, főiskolai szint);
- **programtervező matematikus** (nappali, 10 félév, egyetemi szint);
- **informatikatanár** (nappali, levelező, 10/6 félév, egyetemi szint);
- **informatikus könyvtáros** (nappali, levelező, 10/6 félév, egyetemi szint)

szakokon folyik képzés.

A programtervező informatikus, programozó- és programtervező matematikus szakokon olyan komplex szakmai elméleti ismeretekkel rendelkező szakemberek képzése a cél, akik képesek a mindennapi élet által felvetett gyakorlati problémák tudományos igényű modellezésére, a megfelelő megoldási módszerek megkeresésére, illetve kidolgozására, képesek ilyen feladatok elvégzésére szerveződött csoportok szakmai irányítására, rendelkeznek a szakterületükön folytatható kutatásokhoz szükséges alapvető elméleti-, módszertani- és nyelvismeretekkel. A programtervező informatikus alapképzési BSc szakon a képzés Magyarországon először 2004-ben a Kar gondozásában indult. Ennek, az ún. Bolognai folyamatnak a keretében folyó képzésnek fontos szerepe lesz abban, hogy hazánk az Európai Unió tagjaként szervezetileg is felzárkózzék az európai felsőoktatási térséghez. A szakindítás az informatikai képzés kínálatában elsősorban nem mennyiségi, hanem minőségi változást eredményez:

- A nemzetközi mércéhez alkalmazkodó képzési rendszer és végzettség jelentősen növeli végzett hallgatóink esélyeit a nemzetközi munkamegosztásba történő bekapcsolódásra.
- A mesterképzés (MSc) bevezetése után világos helyzetet teremt a továbbtanulásnál.

A Kar által gondozott tanárszakokon a cél olyan magas szintű informatikai, illetve pedagógiai és általános műveltséggel rendelkező tanárok képzése, akik szakirányú és pedagógiai ismereteik alapján képesek az informatika tanítás minden szintjén feladatok ellátására. A leendő tanár képes lesz a hazai tanítás értékes hagyományainak és színvonalának megőrzésére, a képzésben fontos szerepet játszik továbbá a kitekintés e szakterület oktatásának európai hagyományaira is.

Informatikus könyvtáros szakon olyan felsőfokú szakemberek képzése a cél, akik tanulmányaik során elsajátították a könyvtár- és tájékoztatástudomány korszerű elméleti alapismereteit, továbbá az önálló gyakorlati alkalmazásukhoz szükséges ismereteket, beleértve az információkezelést, illetve az erre vonatkozó kutatások módszertanát is.

A Kar saját szakjain oktatott *hallgatók létszáma* jelentős, jelenleg 1700. Ezenfelül teljes egészében az Informatikai Kar gondozza más karok, így a Közgazdaságtudományi, Jogi, Műszaki Főiskolai Kar informatikai kurzusait is. Munkatársaink jelentős szerepet játszanak a hat programmal működő Matematika és Számítástudományok doktori (PhD) iskolában is évi 8–10 nappali és 10–20 levelező hallgatót beiskolázva.

A Kar kiterjedt nemzetközi kapcsolati lehetővé teszik, hogy évente több tucat hallgatónk vegyen részt külföldi részképzésben Európa és a világ jó nevű felsőoktatási intézményeiben.

Munkatársaink gyümölcsöző hazai szakmai együttműködésének egy szép példája az *Informatikai algoritmusok* című könyv elkészítése is, amelyet minden bizonnyal Debrecenben is haszonnal tanulmányoznak majd az érintettek.

A **Debreceni Egyetem** klasszikus értelemben vett egyetem. 18 karán és kari jogállású intézetében nemcsak elszigetelt szakemberképzés folyik, hanem a szakterületek együttműködésén, a sokszínűség kihasználásán alapuló értelmiségképzés. Pezsgő kulturális életével, szabadidős kínálatával ebben méltó partnere az egyetemnek Debrecen városa is.

Debrecen, 2004. augusztus. 16.

Dr. Pethő Attila
dékán

EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

1117 Budapest, Pázmány Péter sétány 1/C.

Tel: 381-2139, 381-2222 Fax: 381-2140

Honlap: **http://www.inf.elte.hu/**

Az ELTE Informatikai Karán olyan szakembereket képezünk, akik szilárd elméleti alapokra épülő, tudásuk fejlesztését hosszú távon biztosító ismereteiket felhasználva informatikai rendszerek fejlesztését, létrehozását, alkalmazását, bevezetését, működtetését, szervizelését önállóan és csoport tagjaként is képesek magas szinten ellátni. Rendelkeznek továbbá informatikai feladatok megoldásához szükséges modellalkotási képességekkel is. Az informatikai tanár szakon végzett hallgatóink a tanári mesterséget magas színvonalon sajátíthatják el karunkon.

Az **ELTE Informatikai Karán** jelenleg a nappali tagozaton 1780 **programtervező matematikus,** 246 **informatikatanár,** és 102 **térképész** hallgató tanul.

Az esti tagozaton **programozó matematikus** főiskolai alapszakon 396, **számítástechnika-tanár** főiskolai alapszakon 46, és a levelező tagozaton 260 informatika tanár hallgatónk van. Az Informatikai Doktori Iskolának 28 államilag támogatott hallgatója van.

A **programtervező informatikus** alapszakot a 2005/2006-os tanévben indítjuk először. A szak 180 helyére 1031 hallgató jelentkezett, közülük 137-en elektronikusan.

A Kar legfontosabb rendszeres rendezvényei az **Ipari Nap** (a következő az **ELTE Eötvös Napok** keretében lesz májusban), a **Neumann-nap** (2004. november 4-én volt a második) és a **Térinformatikai Világnap** programjában való részvétel (2004. november 17-én volt).

Az Informatikai Kar oktatói aktív szerepet játszottak az *Algoritmusok* (Műszaki Könyvkiadó, 1997), az *Osztott algoritmusok,* (Kiskapu Kiadó, 2002), a *Párhuzamos algoritmusok* (ELTE Eötvös Kiadó, 2003), a *Programozási nyelvek* (Kiskapu Kiadó, 2003), az *Új algoritmusok* (Scolar Kiadó, 2003) és az *Informatikai algoritmusok I* (ELTE Eötvös Kiadó, 2004) című tankönyvek megjelenésében. Az *Informatikai algoritmusok* című tankönyvet a programtervező informatikus alapképzés Bevezetés a matematikába, Programozás módszertani alapjai, Számításelmélet, Numerikus módszerek, Operációkutatás, Architektúrák és operációs rendszerek, Az adatbázisok elméleti alapjai és a Számítógépes grafika című tantárgyaihoz fogjuk tankönyvként használni.

A jelenlegi programtervező matematikus mesterképzésben a tankönyvet az Adatbázisrendszerek, Grafika, Hálózatok, Komputeralgebra, Mesterséges intelligencia, Operációkutatás, Párhuzamos rendszerek, Térinformatika sávok oktatásában hasznosítjuk.

2004-től kezdve az Oktatási Minisztérium támogatja elektronikus tankönyvek létrehozását. A pályázaton 4 informatikai és 6 matematikai könyv kapott támogatást. Ezek közül 4 elektronikus könyvnek – *Bevezetés a programozásba*), *Informatikai algoritmusok I.*), *Párhuzamos algoritmusok*, *Numerikus módszerek I.*) – a szerzői a Kar oktatói.

Budapest, 2005. március 16.

Dr. Kozma László
dékán

Eszterházy Károly Főiskola
Természettudományi Kar
MATEMATIKAI ÉS INFORMATIKAI INTÉZETE
3300 Eger, Leányka út 4. E-mail: **matinf@ektf.hu**
Telefon:06 (36) 520-400 / 4223 Fax: 06 (36) 520-478

ESZTERHÁZY KÁROLY FŐISKOLA
MATEMATIKAI ÉS INFORMATIKAI INTÉZETE

Az MM 51018/1989.-XVI. sz. ügyiratában Pusztai Ferenc az Akadémia egyetértésével 1989 szeptember 1-től a EKF előterjesztésére számítástechnika szakot alapított, amelynek alapján a főiskolánkon az 1989-90-es tanévben matematika-számítástechnika szakon az országban elsőként elindult a szakos tanárok képzése. A későbbi évek során valamennyi főiskola az engedélyezett tanterv és szakalapítás alapján elindította a **számítástechnika szakos tanárképzést.** Az eltelt több mint 10 évben közel 1000 számítástechnika szakos általános iskolai tanári diplomát adtunk ki nappali és esti tagozaton.

A felhalmozott óriási szakmai tapasztalat és a Debreceni Egyetem támogatásának eredményeként a főiskolák között először 2002/2003-as tanévben elindult a **programozó matematikus** szak első évfolyama nappali és esti tagozaton.

2005 őszén indítjuk a Bologna-folyamatnak megfelelő újonnan akkreditált **programtervező informatikus (BSc) alapszakot.**

Főiskolánk 2004-ben akkreditálta az informatikus könyvtáros egyetemi szakot, így a meglévő főiskolai szakok mellett már egyetemi szinten is tanulhatnak intézményünkben informatikát a hallgatók.

A Matematikai és Informatikai Intézetnek 12 főállású, teljes munkaidős közalkalmazotti munkaviszonnyal rendelkező minősített oktatója van, 1 éven belül további 2 fő, 3 éven belül további 4 fő szerzi meg a PhD tudományos fokozatot. Ezzel biztosított a minőségi informatikai képzés iránti elkötelezettség.

Az *Informatikai algoritmusok* című könyvet elsősorban a számítástechnika tanári szakon és a programtervező informatikus alapszakon fogjuk használni. 2004-ben az Oktatási Minisztérium által meghirdetett pályázat támogatja az elektronikus tankönyvek létrehozását. A pályázaton négy elektronikus informatikai könyv kapott támogatást, amelyből egy az *Informatikai algoritmusok*.

Az Eszterházy Károly Főiskola oktatóinak gondozásában készülő *C# programozási nyelv* című elektronikus tankönyv szintén a sikeres pályázók körébe tartozik.

Az intézettel és a képzésekkel kapcsolatos további információk elérhetők az interneten a **http://matinf.ektf.hu** címen.

Eger, 2004. szeptember 10.

Dr. Kovács Emőd

tanszékvezető főiskolai docens

PÉCSI TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR
MATEMATIKAI ÉS INFORMATIKAI INTÉZET

H-7624 Pécs, Ifjúság útja 6.
Telefon/Fax: +36 (72) 503-615,
E-mail: matinf@ttk.pte.hu

PÉCSI TUDOMÁNYEGYETEM MATEMATIKAI ÉS INFORMATIKAI INTÉZETE

A **PTE Természettudományi Karán** hosszabb ideje folyik **számítástechnika szakos tanárképzés,** elsősorban a technika, illetve matematika tanárszakkal együtt szakpárban. Az elmúlt évek jelentős fejlesztéseinek eredményeként önálló informatikus szak indítására került sor a 2003/2004 tanévben, ekkor indult el a TTK-n a **programozó matematikus** szak első évfolyama. Ezeknek a szakoknak a gazdája a **Matematikai és Informatikai Intézet,** amely ugyanakkor ellátja a Karon az informatika ismeretanyagának oktatását közismereti tantárgyként, illetve szaktárgyként a többi természettudományi szakos hallgatók számára.

Ebben az évben két új informatika orientált szak akkreditálására került sor a Természettudományi Karon – a 2004/2005 tanévtől kezdve elindul a képzés a **fizikus informatikus** és **könyvtáros informatikus** szakokon. A két szak közül az első az informatika komoly elméleti alapozását is megköveteli.

Jelenleg a **programtervező informatikus** alapképzési szak akkreditációs anyagának elkészítése van folyamatban, s az akkreditációs eljárásra történő benyújtása ez év szeptemberében fog megtörténni. A szak indítása a 2005/2006 tanévben várható.

Az *Informatikai algoritmusok* című könyvet (hasonlóan az *Új algoritmusok* című és más korábban elkészült könyvekhez) elsősorban a programtervező informatikus alapképzési szakon fogjuk használni tankönyvként. Várhatóan fontos segítséget fog jelenteni több tantárgy oktatásában: Számításelmélet, Operációkutatás, Adatbázisok elméleti alapjai, Számítógépes grafika, Numerikus módszerek.

A könyv számos fejezete hasznos oktatási anyagot képez a TTK-n jelenleg folyó matematika tanár és az ebben az évben induló fizikus informatikus képzéshez, továbbá a PTE más informatikai irányú képzéseihez is.

Pécs, 2004. július 15.

Dr. Szeidl László
egyetemi tanár
igazgató

SZEGEDI TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR
INFORMATIKAI TANSZÉKCSOPORT

6720 Szeged, Árpád tér 2.
Levelezési cím: 6701 Szeged, Postafiók 652.
Telefon: +36 62 546396, Fax: +36 62 546-397
E-mail: depart@inf.u-szeged.hu

SZEGEDI TUDOMÁNYEGYETEM INFORMATIKAI TANSZÉKCSOPORTJA

A Szegedi Tudományegyetem Természettudományi Kara 1921-ben kezdte meg működését, amikor Szeged városa befogadta a Kolozsvárról áttelepült tudományegyetemet. A Természettudományi Kar oktatási és kutatási tevékenységét tudományterületi alapon felosztotta intézetei, tanszékcsoportjai között. Hat tanszékcsoport működik, amelyek mindegyike részben önálló kutató és oktató intézetként is felfogható.

A Szegedi Tudományegyetem képzési rendszerén belül az Informatikai Tanszékcsoport öt, az informatika tudományághoz tartozó szaknak a gazdája: **programozó matematikus szak, programtervező matematikus szak, közgazdasági programozó matematikus szak, műszaki informatikai szak** és **informatika tanár szak.** Az Informatikai Tanszékcsoport 1990-ben alakult, ekkor vette át a nagy múltú szegedi informatikusképzés gondozását a TTK Matematikai Tanszékcsoportjától. A öt tanszéket *(Alkalmazott Informatika Tanszék, Képfeldolgozás és Számítógépes Grafika Tanszék, Számítástudomány Alapjai Tanszék, Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszék, Szoftverfejlesztés Tanszék)* magában foglaló Informatikai Tanszékcsoport oktató- és kutatómunkáját az MTA-SZTE Mesterséges Intelligencia Tanszéki Kutatócsoport is segíti. A hallgatóink előtt is nyitva álló intézeti könyvtárunk folyamatosan bővül, könyvállománya meghaladja az 5000 kötetet és közel 200 tudományos folyóirat is elérhető. A Tanszékcsoport a gyakorlati oktatásra kiválóan felszerelt számítógépes kabinettermeket üzemeltet, amelyekben a hallgatók a tantervi órákon kívül is dolgozhatnak és számukra a szabad Internet-hozzáférés is biztosított.

A hallgatóink bekapcsolódhatnak a tanszékeken folyó kutatásokba, és ezt végzés után folytathatják PhD képzés keretében. A tudományos munka eredményeként rendszeresen színvonalas Országos Tudományos Diákköri dolgozatok születnek. A Tanszékcsoport lehetőség szerint támogatja a hallgatók külföldi részképzését. Ennek érdekében külföldi partnerekkel több nemzetközi projektben vesz részt a tanszékcsoport, melynek eredményeként számos hallgató élhet a külföldi részképzés lehetőségével.

Szeged város pezsgő sport és kulturális élete ragyogó lehetőségeket nyújt a tanulás mellett pihenni, kikapcsolódni és szórakozni vágyó hallgatóinknak.

Szeged, 2004. szeptember 2.

Dr. Csirik János
tanszékcsoportvezető egyetemi tanár