Iványi Antal

PÁRHUZAMOS ALGORITMUSOK

ELTE Eötvös Kiadó

Iványi Antal

PÁRHUZAMOS ALGORITMUSOK

ELTE Eötvös Kiadó, Budapest

Ez az elektronikus tankönyv az Oktatási Minisztérium támogatásával, a Szakkönyv- és Tankönyvtámogatási Pályázat keretében készült.

Lektorok: Fóthi Ákos, Horváth Zoltán, Kása Zoltán, Pécsy Gábor, Szűcs László

Kiadja az
ELTE Eötvös Kiadó
1051 Budapest, Szerb utca 1.
Telefon: 411-6740
Fax: 485-52-26

Honlap: http://www.elte.hu/szervezet/eotvos_kiado.html
Elektronikus cím: eotvoskiado@ludens.elte.hu

Felelős kiadó: Pándi András

Előszó

Ez az elektronikus tankönyv az ELTE programtervező matematikus szakának I/2B. sávjában a *Párhuzamos algoritmusok* tantárgy, valamint az Informatikai Doktori Iskolában a *Párhuzamos és osztott algoritmusok elemzése* tantárgy keretében tartott előadások anyagát tartalmazza.

A könyv hat fejezetből (*Bevezetés, Párhuzamos gép, Rács, Kocka, Szinkronizált hálózat, Hagyományos és elektronikus irodalom*) és nyolc további részből (*Jelölések, Angol szakkifejezések, Magyar szakkifejezések, Irodalomjegyzék, Lelőhelyjegyzék, Névmutató, Tárgymutató, Megoldások*) áll.

A második fejezetet Horváth Zoltán, a harmadikat Szűcs László, a negyediket Pécsy Gábor, a könyv többi részét Kása Zoltán lektorálta. Fóthi Ákos sokat segített a könyv felépítésének és stílusának kialakításában.

Köszönöm a könyv lektorainak a sok hasznos észrevételt. Lényeges segítséget jelentett, hogy a lektorok nem csak a saját részüket nézték át. Kollégáim közül köszönöm Balogh Ádám (ELTE), Dózsa Gábor (SZTAKI), Fábián Mária (ELTE), Fóthi Ákos (ELTE), Miletics Edit (Széchenyi István Egyetem), Pethő Attila (Debreceni Egyetem), Rét Anna (Műszaki Könyvkiadó), Scharnitzky Viktorné (ELTE), Sima Dezső (BMF), Simon Péter (ELTE) és Szili László (ELTE) javaslatait. A korábbi változatok alapján vizsgázó hallgatók közül pedig Balázs Gábor (ELTE), Baksa Klára (ELTE), Dévai Gergely (ELTE), Hajdu Tamás (ELTE), Hegyessy Tamás (ELTE), Hermann Péter (ELTE), Kapinya Judit (ELTE), Kovács Péter (ELTE), Metykó Beáta

(ELTE) és Szalai Róbert (ELTE) segítettek.

A könyv ábráit Locher Kornél rajzolta. A könyv kézirata a LaTeX kiadványszerkesztővel készült, amelyet az elmúlt években fejlesztettünk ki Belényesi Viktorral és Locher. Az irodalomjegyzéket Iványi Anna tette élővé.

A párhuzamos algoritmusok szakirodalma nagyon gyorsan fejlődik. Ezért a könyv

http://people.inf.elte.hu/tony/books/paralg

című *honlapján* folyamatosan karbantartjuk a *külföldi és hazai hivatkozások* aktuális listáját (hiperszöveg formájában, így a honlap látogatói kattintással közvetlenül elérhetik az idézett művek szerzőinek honlapját, és olyan elektronikus könyvtárakat, ahonnan az idézett cikkek letölthetők). Ugyanitt a beérkezett észrevételek alapján a könyv szövegében javasolt *változtatások* listáját is karbantartjuk.

Az *elektronikus címek aláhúzása* azt jelzi (itt az előszóban, az irodalomjegyzékben és a lelőhelyjegyzékben), hogy a könyv honlapján lévő PDF és DVI változatban a címek élnek, a PS változatban pedig kék színnel ki vannak emelve.

Kérem a könyv Olvasóit, hogy észrevételeiket és javaslataikat írják meg a

tony@inf.elte.hu

címre. Különösen köszönöm a témával kapcsolatos új feladatokat.

Budapest, 2005. január 26.

Iványi Antal

1. Bevezetés

A számítógépes feladatmegoldás természetes és hagyományos világa a *soros adatfeldolgozás*. A különböző alkalmazási területek nagy teljesítményigényének és környezetünk párhuzamos jellegének hatására azonban rohamosan fejlődik a párhuzamos feldolgozás is.

Párhuzamos adatfeldolgozás. Egy feladat párhuzamos megoldása ugyanúgy 3 lépésből áll, mint a soros megoldás. Először meg kell érteni és pontosan *meg kell fogalmazni* a feladatot – ez a lépés hasonló a soros feldolgozás első lépéséhez. Mivel a soros adatfeldolgozás számára ismert feladatokat oldunk meg, a problémák megértése – az eddigi tapasztalatok szerint – az olvasók többsége (például harmadéves programtervező és informatika szakos hallgatók) számára nem jelent gondot.

A második lépésben választunk egy ismert architektúrát (esetleg tervezünk egy újat) és ahhoz *tervezünk egy megoldási algoritmust*. Ez vagy új algoritmus, vagy egy soros algoritmus párhuzamosított változata. Könyvünk tárgya a párhuzamos adatfeldolgozásnak ez a része, azaz a *párhuzamos algoritmusok tervezése és elemzése*.

Végül a harmadik lépésben az algoritmust a meglévő programozási módszerek valamelyikével *párhuzamos program formájában megvalósítjuk* (itt is szóba jön új programozási módszer alkalmazása).

Párhuzamos algoritmusok. Ha egy feladatot együttműködő processzorok segítségével oldunk meg, akkor a számítási idő rendszerint csökken, mi-

vel bizonyos műveletek egyidejűleg elvégezhetők. Ennek a csökkenésnek az ára a nagyobb beruházásigény és az elvégzendő munka mennyiségének növekedése.

A párhuzamos algoritmusokat különböző szempontok szerint szokták osztályozni.

Az egyik szempont a processzorok működésének ***időzítése.*** Eszerint vannak összehangoltan dolgozó, ún. ***szinkronizált processzorok*** és egymástól függetlenül dolgozó, ún. ***aszinkron processzorok.***Emellett vannak ***hibrid*** megoldások, ahol a processzorok részben összehangoltan dolgoznak.

Egy másik osztályozási szempont az együttműködő processzorok közötti információcsere módja, azaz a ***kommunikációs modell.*** Ez az információcsere történhet a közös memória segítségével és/vagy üzenetek küldésével és fogadásával.

Fontos a ***rendszer megbízhatóságával*** kapcsolatos felfogásunk: hibátlan működést tételezünk fel vagy megengedünk bizonyos típusú hibákat (például a processzorok vagy az adatátviteli vonalak meghibásodását).

Lényeges az az ***architektúra,*** amelyre algoritmusainkat tervezzük. A második fejezetben röviden áttekintjük a párhuzamos számítógépek főbb típusait, az elemzésekhez azonban a számítási modelleknek nevezett ***absztrakt számítógépeket*** használjuk.

A párhuzamos algoritmusok közül csak a szinkronizált modelleken megvalósíthatókat tárgyaljuk. Eközben feltételezzük, hogy a processzorok, adatátviteli vonalak, közös és saját memória – azaz a rendszer minden eleme – megbízhatóan működnek.

Előismeretek. A tárgyalt anyag nagy része az informatikai képzés alapvető kurzusait (adatszerkezetek, algoritmusok, analízis, diszkrét matematika, programozás) sikeresen elvégző hallgatók számára érthető. A véletlenített algoritmusok elemzése a binomiális eloszlás néhány tulajdonságán alapul, ezért ezekben az alfejezetekben a valószínűségszámítási ismeretekre is tá-

maszkodunk. Ezeket a részeket a címekben és a tartalomjegyzékben csillaggal (∗) jelöltük. A feladatok egy részének megoldásához hasznosak az operációkutatással, optimalizálással és szimulációval kapcsolatos ismeretek.

A fejezetek egymásra épülése. Bár helyenként felhasználunk a könyv korábbi részeiben bemutatott algoritmusokat, a bevezető első fejezet elolvasása után a többi fejezet egymástól függetlenül is érthető. Ha azonban az Olvasót például a konvex burok hiperkockán való meghatározása érdekli, célszerű a párhuzamos gépen és a rácson alkalmazott algoritmusokkal is megismerkedni (a tartalomjegyzék és a tárgymutató segít a *visszalapozásban*).

Tartalom. A könyv hat fejezetből és hét további részből áll.

Az első fejezetben (*Bevezetés*) előkészítjük a további anyagot: megadjuk az alapvető meghatározásokat, ismertetjük a felhasznált számítási modelleket és pszeudokódot, foglalkozunk a rekurzív algoritmusokkal és a rekurzív egyenletekkel, a véletlenített algoritmusokkal, az alsó korlátokkal és az anomáliával.

A további négy fejezetben párhuzamos algoritmusokat ismertetünk és elemzünk – az algoritmusok megvalósítására használt számítási modellek (és az azok alapjául szolgáló architektúrák) szerint csoportosítva: a szinkronizált processzorok kapcsolatát párhuzamos közvetlen hozzáférésű gépek (*Párhuzamos gép*), rácsok (*Rács*), kockák (*Kocka*) és tetszőleges gráfok (*Szinkron hálózat*) segítségével adjuk meg.

A hatodik fejezetben (*Hagyományos és elektronikus irodalom*) a könyv írásához felhasznált és az egyes témák részletesebb megismeréséhez ajánlott – nyomtatott és elektronikus formában, magyar és idegen nyelveken hozzáférhető – dokumentumok tartalmát és lelőhelyi adatait foglaljuk össze.

Módszertan. A könyv felépítésével és az alkalmazott tipográfiai eszközökkel igyekeztünk megkönnyíteni az anyag megértését. A szövegben gyakran alkalmaztunk magyarázattal ellátott ábrákat és példákat. Az egyes feje-

zetek végén gyakorlatok és feladatok vannak. A gyakorlatok a fejezet anyagának jobb megértését segítik elő és az anyag ismeretében rendszerint gyorsan megoldhatók. A feladatok megoldása önálló gondolkodást és esetenként több matematikai ismeretet igényel.

Az algoritmusok elemzését nem törtük meg hivatkozásokkal, viszont a hatodik fejezetben témakörönként összefoglaltuk az elsősorban ajánlott szakkönyvek és cikkek adatait, és alternatív megoldásokra is utaltunk.

A könyv végén összefoglaltuk az alkalmazott jelöléseket (*Jelölések*), és megadtuk az angol szakkifejezések (*Angol szakkifejezések*) magyar, valamint a magyar szakkifejezések (*Magyar szakkifejezések*) angol megfelelőjét. A bizonyítások végét ■, a példák végét ♠ jelzi. A definíciókban az új fogalom nevét ***kék félkövér dőlt betűkkel*** írtuk. A sorszámozott képletek tördelésénél az Olvasók kényelmét szolgáló amerikai stílust követtük (azaz minden relációjel új sorba kerül – az ilyen képletek sorfolytonosan olvasandók). Mivel a vesszőnek gyakran van a szövegben nyelvtani szerepe, a számokban tizedespontot használunk. A gyakorlatok egy részének megoldását megadtuk.

Az irodalomjegyzékben együtt adjuk meg a hazai és külföldi forrásokat. Az idegen nyelvű szakirodalomból csak a klasszikus és a viszonylag friss műveket soroljuk fel. A magyar nyelvű anyag összeállítása során – az algoritmusok elemzésével foglalkozó műveket tekintve – teljességre törekedtünk. Az elektronikus formában elérhető dokumentumoknál megadtuk a hálózati címet is. Minden dokumentumnál feltüntettük azoknak a szövegrészeknek az azonosítóját, amelyekből hivatkozás van az adott dokumentumra (pl. 2.7. a megfelelő alfejezetre, 113 pedig az irodalomjegyzék 113-as sorszámú elemében lévő hivatkozásra utal).

A névmutató a könyvben előforduló neveket tartalmazza – teljességre törekedve.

A tárgymutatóban *dőlt számok* jelzik az egyes fogalmak definiálásának helyét. Az előfordulási helyek felsorolásánál megelégedtünk a lényegesnek

tartott helyekre való utalással. A gyakorlatokban, feladatokban, ábrákban előforduló fogalmakat az oldalszám melletti rövidítések (mint *gy, fe, áb*) jelzik.

1.1. Alapfogalmak

A párhuzamos algoritmusok tárgyalása a soros algoritmusokra épül, ezért a szokásos fogalmak mellett a soros algoritmusokkal kapcsolatos definíciókat is megadjuk.

Az algoritmusok elemzése – a helyesség bizonyítása mellett – a végrehajtáshoz szükséges erőforrások (ez számítógépen megvalósított algoritmus esetében lehet processzoridő, memóriakapacitás – számítási modellen futó algoritmus esetében pedig memóriarekesz, kommunikációs üzenet, műveleti lépés) mennyiségének meghatározását is jelenti. Gyakran nem tudjuk vagy nem akarjuk az igényelt erőforrás mennyiségét pontosan megadni. Ilyenkor megelégszünk az igény nagyságrendjének jellemzésével.

Ennek a jellemzésnek a jól bevált eszközei az Ω, O, Θ, o és ω jelölések. Mivel az igények általában nemnegatívak, ezért az alábbi meghatározásokban mindenütt feltesszük, hogy az f és g függvények a pozitív egészek halmazán vannak értelmezve, az $f(n)$ és $g(n)$ függvényértékek pedig nemnegatív valós számok.

Az O jelöléssel aszimptotikus felső, az Ω jelöléssel aszimptotikus alsó korlátot tudunk adni, míg a Θ jelöléssel pontosan meg tudjuk adni a vizsgált függvény aszimptotikus viselkedését.

$O(g(n))$ (ejtsd: ***nagy ordó***) azon $f(n)$ függvények halmazát jelenti, amelyekhez léteznek olyan c pozitív valós és n_0 pozitív egész állandók, hogy

$$\text{ha } n \geq n_0,\ f(n) \leq cg(n)\ . \tag{1.1}$$

$\Omega(g(n))$ (ejtsd: ***nagy omega***) azon $f(n)$ függvények halmazát jelenti, ame-

lyekhez léteznek olyan c pozitív valós és n_0 pozitív egész állandók, hogy

$$\text{ha } n \geq n_0, \text{ akkor} f(n) \geq cg(n) \ . \tag{1.2}$$

$\Theta(g(n))$ (ejtsd: ***nagy teta***) azon $f(n)$ függvények halmazát jelenti, amelyekhez léteznek olyan pozitív valós c_1, c_2 és pozitív egész n_0 állandók, hogy

$$\text{ha } n \geq n_0, \text{ akkor } c_1 g(n) \leq f(n) \leq c_2 g(n) \ . \tag{1.3}$$

Az elemzésekben arra törekszünk, hogy Θ típusú becsléseket adjunk, amihez azonos argumentumfüggvényt tartalmazó O és Ω típusú becslésekre van szükség. Ha egy becslésben hangsúlyozni akarjuk, hogy a becslés nem éles, akkor hasznos a o és a ω jelölés.

$o(g(n))$ (ejtsd: ***kis ordó***) azon $f(n)$ függvények halmazát jelenti, melyekre teljesül, hogy minden pozitív valós c állandóhoz megadható egy pozitív egész n_0 úgy, hogy

$$\text{ha } n \geq n_0, \text{ akkor} f(n) \leq cg(n) \ . \tag{1.4}$$

$\omega(g(n))$ (ejtsd: ***kis omega***) azon $f(n)$ függvények halmazát jelenti, melyekre teljesül, hogy minden pozitív valós c állandóhoz megadható egy pozitív egész n_0 úgy, hogy

$$\text{ha } n \geq n_0, \text{ akkor } f(n) \geq cg(n) \ . \tag{1.5}$$

Ha $f(n) = o(g(n))$, akkor

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \ , \tag{1.6}$$

és ha $f(n) = \omega(g(n))$, akkor

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \ . \tag{1.7}$$

Az 1.1. táblázatban összefoglaltuk a függvények növekedésével kapcsolatban leggyakrabban használt jelöléseket és kifejezéseket. A táblázatban

Sorszám	Növekedési korlát képlettel	Korlát típusa szóval
1	$\Theta(1)$	konstans
2	$\Theta(\lg^\star n)$	majdnem konstans
3	$o(\lg n)$	szublogaritmikus
4	$\Theta(\lg n)$	logaritmikus
5	$\Theta(\lg^{\Theta(1)} n)$	polilogaritmikus
6	$\omega(\lg n)$	szuperlogaritmikus
7	$o(n)$	szublineáris
8	$\Theta(n)$	lineáris
9	$\omega(n)$	szuperlineáris
10	$\Theta(n^2)$	négyzetes
11	$\Theta(n^3)$	köbös
12	$o(n^{\Theta(1)})$	szubpolinomiális
13	$\Theta(n^{\Theta(1)})$	polinomiális
14	$\omega(n^{\Theta(1)})$	szuperpolinomiális

1.1. táblázat Függvények növekedésének leggyakoribb korlátai.

szereplő korlátok tetszőleges erőforrással kapcsolatban használhatók – elemzéseinkben azonban elsősorban lépésszámokra vonatkoznak.

A táblázatban a *szub* kifejezést *kisebb,* a *szuper* kifejezést pedig *nagyobb* értelemben használtuk. Érdemes megemlíteni, hogy szokás a *kisebb vagy egyenlő,* illetve a *nagyobb vagy egyenlő* értelmű használat is.

A könyvben a szuperpolinomiális kifejezés szinonimájaként fogjuk használni az *exponenciális* jelzőt. Ezzel az ***exponenciális futási időt*** a matematikában szokásosnál tágabban definiáltuk: ha egy algoritmus futási ideje felülről nem korlátozható polinommal, akkor exponenciálisnak nevezzük.

1.2. Hatékonysági mértékek

Az algoritmusok elemzése során az igényelt erőforrások mennyiségét ***abszolút*** és ***relatív*** mennyiségekkel jellemezzük.

Ezeknek a mennyiségeknek a célszerű megválasztása attól is függ, hogy az algoritmus ***konkrét gépen*** vagy ***absztrakt gépen*** (számítási modellen) fut.

Jelöljük $W(n, \pi, \mathcal{A})$-vel, illetve $W(n, \pi, p, \mathcal{P})$-vel azt az időt (azoknak a lépéseknek a számát), amely alatt a π problémát az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus – (utóbbi p processzort felhasználva) – n méretű feladatokra legrosszabb esetben megoldja.

Hasonlóképpen jelöljük $B(n, \pi, \mathcal{A})$-vel, illetve $B(n, \pi, p, \mathcal{P})$-vel azt az időt (azoknak a lépéseknek a számát), amely alatt a π problémát az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus (utóbbi p processzort felhasználva) – n méretű feladatokra legjobb esetben megoldja.

Jelöljük $N(n, \pi)$-vel, illetve $N(n, \pi, p)$-vel azoknak a lépéseknek a számát, amelyekre az n méretű π feladat megoldásához mindenképpen szüksége van bármely soros, illetve bármely párhuzamos algoritmusnak – utóbbinak akkor, ha legfeljebb p processzort vehet igénybe.

Tegyük fel, hogy minden n-re adott a π feladat n méretű konkrét előfordulásainak $D(n, \pi)$ eloszlásfüggvénye. Ekkor legyen $E(n, \pi, \mathcal{A})$, illetve $E(n, \pi, p, \mathcal{P})$ annak az időnek a várható értéke, amennyi alatt a π problémát n méretű feladatokra megoldja az $\mathcal{A}$ soros, illetve a $\mathcal{P}$ párhuzamos algoritmus – utóbbi p processzort felhasználva.

A tankönyvekben az elemzések során gyakori az a feltételezés, hogy az azonos méretű bemenetek előfordulási valószínűsége azonos. Ilyenkor ***átlagos futási időről*** beszélünk, amit $A(n, \mathcal{A})$-val jelölünk.

A W, B, N, E és A jellemzők függnek attól a számítási modelltől is, amelyen az algoritmust megvalósítjuk. Az egyszerűség kedvéért feltesszük, az algoritmus egyúttal a számítási modellt is egyértelműen meghatározza.

Amennyiben a szövegkörnyezet alapján egyértelmű, hogy melyik problémáról van szó, akkor a jelölésekből π-t elhagyjuk.

Ezek között a jellemzők között érvényesek az

$$N(n) \leq B(n, \mathcal{A}) \tag{1.8}$$
$$\leq E(n, \mathcal{A}) \tag{1.9}$$
$$\leq W(n, \mathcal{A}) \tag{1.10}$$

egyenlőtlenségek. Hasonlóképpen a párhuzamos algoritmusok jellemző adataira az

$$N(n, p) \leq B(n, \mathcal{P}, p) \tag{1.11}$$
$$\leq E(n, \mathcal{P}, p) \tag{1.12}$$
$$\leq W(n, \mathcal{P}, p) \tag{1.13}$$

egyenlőtlenségek teljesülnek. Az átlagos futási időre pedig a

$$B(n, \mathcal{A}) \leq A(n, \mathcal{A}) \tag{1.14}$$
$$\leq W(n, \mathcal{A}), \tag{1.15}$$

illetve

$$B(n, \mathcal{P}, p) \leq A(n, \mathcal{P}, p) \tag{1.16}$$
$$\leq W(n, \mathcal{P}, p) \tag{1.17}$$

áll fenn.

Hangsúlyozzuk, hogy ezek a jelölések nem csak futási időre, hanem az algoritmusok más jellemzőire – például memóriaigény, küldött üzenetek száma – is alkalmazhatók.

Ezután relatív jellemzőket, úgynevezett ***hatékonysági mértékeket*** definiálunk.

A gyorsítás (vagy relatív lépésszám) azt mutatja, hányszor kisebb a párhuzamos algoritmus futási ideje a soros algoritmus futási idejénél.

A $\mathcal{P}$ párhuzamos algoritmusnak az $\mathcal{A}$ soros algoritmusra vonatkozó ***gyorsítása***

$$g(n, \mathcal{A}, \mathcal{P}) = \frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} \ . \tag{1.18}$$

Ha a gyorsítás egyenesen arányos a felhasznált processzorok számával, akkor lineáris gyorsításról beszélünk. Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = \Theta(p) \ , \tag{1.19}$$

akkor $\mathcal{P}$ A-ra vonatkozó gyorsítása ***lineáris.***

Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = o(p) \ , \tag{1.20}$$

akkor $\mathcal{P}$ A-ra vonatkozó gyorsítása ***szublineáris.***

Ha a $\mathcal{P}$ párhuzamos és az $\mathcal{A}$ soros algoritmusra

$$\frac{W(n, \mathcal{A})}{W(n, p, \mathcal{P})} = \omega(p) \ , \tag{1.21}$$

akkor $\mathcal{P}$ A-ra vonatkozó gyorsítása *szuperlineáris.*

A párhuzamos algoritmusok esetében fontos jellemző az $m(n, p, \mathcal{P})$ ***munka,*** amit a futási idő és a processzorszám szorzatával definiálunk. Akkor is ez a szokásos definíció, ha a processzorok egy része csak a futási idő egy részében dolgozik:

$$m(n, p, \mathcal{P}) = pW(n, p, \mathcal{P}) \ . \tag{1.22}$$

Érdemes hangsúlyozni, hogy – különösen az aszinkron algoritmusok esetében – a ténylegesen elvégzett lépések száma lényegesen kevesebb lehet, mint amit a fenti (1.22) képlet szerint kapunk.

Egy $\mathcal{P}$ párhuzamos algoritmusnak az ugyanazon problémát megoldó soros algoritmusra vonatkozó $h(n, p, \mathcal{P}, \mathcal{A})$ ***hatékonysága*** a két algoritmus munkájának hányadosa:

$$h(n, p, \mathcal{P}, \mathcal{A}) = \frac{W(n, A)}{pW(n, p, \mathcal{P})} \ . \tag{1.23}$$

Abban a természetes esetben, amikor a párhuzamos algoritmus munkája legalább akkora, mint a soros algoritmusé, a hatékonyság nulla és egy közötti érték – és a viszonylag nagy érték a kedvező.

Központi fogalom a párhuzamos algoritmusok elemzésével kapcsolatban a munkahatékonyság és munkaoptimalitás. Ha a $\mathcal{P}$ párhuzamos és $\mathcal{A}$ soros algoritmusra

$$pW(n,p,\mathcal{P}) = \widetilde{O}(W(n,\mathcal{A})) \ , \tag{1.24}$$

akkor $\mathcal{P}$ $\mathcal{A}$-ra nézve ***munkahatékony.***

Ez a meghatározás egyenértékű a

$$\frac{pW(n,p,\mathcal{P})}{W(n,\mathcal{A})} = O(1) \tag{1.25}$$

egyenlőség előírásával. Eszerint egy párhuzamos algoritmus csak akkor munkahatékony, ha munkája nagyságrendileg nem nagyobb, mint a soros algoritmus munkája.

Ha a $\mathcal{P}$ párhuzamos és $\mathcal{A}$ soros algoritmusra

$$pW(n,p,\mathcal{P}) = O(W(n,\mathcal{A})) \ , \tag{1.26}$$

akkor $\mathcal{P}$ $\mathcal{A}$-ra nézve ***munkaoptimális.***

Ez a meghatározás egyenértékű a

$$\frac{pW(n,p,\mathcal{P})}{W(n,\mathcal{A})} = O(\lg^k n) \tag{1.27}$$

egyenlőség előírásával. Eszerint egy párhuzamos algoritmus csak akkor munkaoptimális, ha van olyan k érték, hogy a párhuzamos algoritmus munkája nagyságrendileg nem nagyobb, mint a soros algoritmus munkájának $(\lg^k n)$-szerese.

Egy párhuzamos algoritmus csak akkor munkahatékony, ha a gyorsítása legalább lineáris. Egy munkahatékony párhuzamos algoritmus hatékonysága $\Theta(1)$.

Ha egy algoritmus egy feladat megoldásához adott erőforrásból csak

$O(N(n))$ mennyiséget használ fel, akkor az algoritmust az adott erőforrásra, számítási modellre (és processzorszámra) nézve ***aszimptotikusan optimálisnak*** nevezzük.

Ha egy $\mathcal{A}$ ($\mathcal{P}$) algoritmus egy feladat megoldásához adott erőforrásból a bemenet minden lehetséges $n \geq 1$ mérete esetében csak az okvetlenül szükséges $N(n, \mathcal{A})$ – illetve ($N(n, p, \mathcal{A})$) – mennyiséget használja fel, azaz

$$W(n, \mathcal{A}) = N(n, \mathcal{A}) \ , \tag{1.28}$$

illetve

$$W(n, p, \mathcal{P}) = N(n, p, \mathcal{P}) \ , \tag{1.29}$$

akkor az algoritmust az adott erőforrásra (és az adott számítási modellre) nézve ***abszolút optimálisnak*** nevezzük és azt mondjuk, hogy a vizsgált feladat ***pontos bonyolultsága*** $N(n)$ ($N(n, p)$).

Két algoritmus összehasonlításakor a

$$W(n, \mathcal{A}) = \Theta(W(n, \mathcal{B})) \tag{1.30}$$

esetben azt mondjuk, hogy a $\mathcal{A}$ és $\mathcal{B}$ algoritmus futási idejének növekedési sebessége aszimptotikusan ***azonos nagyságrendű.***

Amikor két algoritmus futási idejét (például a legrosszabb esetben) összehasonlítjuk, akkor gyakran találunk olyan helyeket, melyeknél kisebb méretű bemenetekre az egyik, míg nagyobb méretű bemenetekre a másik algoritmus futási ideje kedvezőbb. A formális definíció a következő: ha a pozitív egész helyeken értelmezett $f(n)$ és $g(n)$ függvényekre, valamint a $v \geq 0$ pozitív egész számra teljesül, hogy

1. $f(v) = g(v)$;
2. $(f(v-1) - g(v-1))(f(v+1) - g(v+1)) < 0$,

akkor a v számot az $f(n)$ és $g(n)$ függvények ***váltási helyének*** nevezzük.

Például két mátrix szorzatának a definíció alapján történő és a Strassen-algoritmussal történő kiszámítását összehasonlítva (lásd például Cormen,

Leiserson, Rivest és Stein többször idézett új könyvét) azt kapjuk, hogy kis mátrixok esetén a hagyományos módszer, míg nagy mátrixok esetén a Strassen-algoritmus az előnyösebb – egy váltási hely van, melynek értéke körülbelül 20.

Az idő mellett algoritmusok számos más erőforrást is használnak – például memóriát, üzeneteket. Utóbbira például $W_{ü}(n, p, \mathrm{P})$ módon hivatkozhatunk.

1.3. Pszeudokód

Az algoritmusok formális leírására a következő, a Pascal és a C++ nyelvek elemeiből összeállított pszeudokódot használjuk.

1. Az algoritmus blokkszerkezetét elsősorban a tagolás tükrözi. Ezt a megoldást a programozási nyelvekben is használják – bár a használat szabályai nyelvenként változnak. A pszeudokódot lényegesen egyszerűsíti és áttekinthetőbbé teszi – értelmezése tapasztalataink szerint nem okoz gondot.

2. A változók neve betűvel kezdődik. A változók típusát külön nem deklaráljuk, mert az a környezetből látszik. Egyszerű adattípusok (mint egész, lebegőpontos, karakter, logikai stb.) fognak szerepelni.

3. A változók értékadó utasításokkal kapnak értéket:
⟨*változónév*⟩ ← ⟨*kifejezés*⟩

4. Két logikai érték van, az IGAZ és a HAMIS. Ezek a logikai értékek az **és** ($\wedge$), **vagy** ($\vee$) és a **nem** ($\neg$) logikai operátorokkal, valamint a $<$, $\leq$, $=$, $\geq$ és $>$ relációs operátorokkal állíthatók elő.

5. A többdimenziós tömbök elemei szögletes zárójelek közé tett indexekkel érhetők el, például $A[i, j]$. Az indexek nullától kezdődnek. A tömb egy részére az indextartomány megadásával hivatkozhatunk: például $A[3\,..\,n]$.

6. A következő két ciklusszervező utasítást alkalmazzuk: **while** és **for**.

A **while** ciklus alakja a következő:

while ⟨*feltétel*⟩
⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨*u*. *utasítás*⟩

Amíg a ⟨*feltétel*⟩ IGAZ, az u ($u \geq 1$) utasítás végrehajtódik. Amikor a ⟨*feltétel*⟩ HAMIS lesz, kilépünk a ciklusból.

A **for** ciklus alakja a következő:

for ⟨*ciklusváltozó*⟩ ← ⟨*kezdő érték*⟩ **to** ⟨*befejező érték*⟩
do ⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨*u*. *utasítás*⟩

Ha például a ciklusváltozó i, a kezdőérték k és a befejező érték b, akkor az u utasítás egymás után végrehajtódik az $i = k,\ k+1,\ \ldots, b$ értékekre.

7. A feltételes utasítás lehetséges alakjai:

if ⟨*feltétel*⟩
then
⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨*u*. *utasítás*⟩

vagy

if ⟨*feltétel*⟩
then
⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨*u*. *utasítás*⟩
else
⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨*v*. *utasítás*⟩

8. A bevitel/kivitel a **read** és **write** utasításokkal történik. Pontos formájuk számítási modellenként változó.

9. Egyetlen eljárás van, amely fejből és törzsből áll. A fej

⟨ELJÁRÁS NEVE⟩(⟨*paraméterlista*⟩) ⟨*eljárás típusa*⟩
Számítási modell: ⟨*számítási modell*⟩
Bemenet: ⟨*bemenő adatok leírása*⟩
Kimenet: ⟨*kimenő adatok leírása*⟩

Az eljárás típusa lehet *soros eljárás*, *soros rekurzív eljárás*, *párhuzamos eljárás* és *párhuzamos rekurzív eljárás*.

Az eljárások törzse sorszámozott utasításokból áll. Az utolsó utasítást kivéve az egyes utasítások végét a következő sorszám, a törzs végét a tagolás jelzi.

Ezek a számozott utasítások gyakran az algoritmus nagy (több lépésből álló) részét tükrözik. Ilyenkor az elemzésben ezeket a részeket szakasznak vagy fázisnak nevezzük. Más esetekben több számozott lépést együtt neve-

zünk az algoritmus szakaszának vagy fázisának.

10. Az eljárások hívásának alakja:

⟨ELJÁRÁS-NEVE⟩⟨(*paraméterek listája*)⟩

A soros és a párhuzamos eljárások esetén egyaránt ezt a hívó utasítást használjuk.

11. A megjegyzések a ▷ jellel kezdődnek és az adott sor végéig tartanak.

12. A párhuzamosságot egy p processzoros PRAM vagy lánc esetében a következőképpen írjuk le:

P_i **in parallel for** $i \leftarrow 1$ **to** p
do ⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨u. *utasítás*⟩

Egy $m_1 \times m_2 \times \ldots \times m_k$ méretű, k dimenziós rács esetében a hasonló utasítás a

$P_{i_1,i_2,\ldots,i_k}$ **in parallel for** $i_1 \leftarrow 1$ **to** m_1, $\ldots$, $i_k \leftarrow 1$ **to** m_k

sorral kezdődik.

1.4. Számítási modellek

Az algoritmusokat absztrakt vagy konkrét gépeken hajthatjuk végre. Ebben az alfejezetben először röviden bemutatjuk a párhuzamos számítógépek főbb típusait, azután az absztrakt gépek közül a párhuzamos közvetlen hozzáférésű gépekkel és a hálózatokkal foglalkozunk.

1.4.1. Számítógépek

Az elmúlt évtizedekben sok különböző párhuzamos számítógépet építettek, és számos próbálkozás történt rendszerezésükre.

Flynn 1972-ben az utasításáram és az adatáram alapján 4 csoportot különböztetett meg:

- SISD (**S**imple **I**nstruction Stream – **S**imple **D**ata Stream);
- SIMD (**S**imple **I**nstruction Stream – **M**ultiple **D**ata Stream);
- MISD (**M**ultiple **I**nstruction Strem – **S**ingle **D**ata Stream);
- MIMD (**M**ultiple **I**nstruction Stream – **M**ultiple **D**ata Stream).

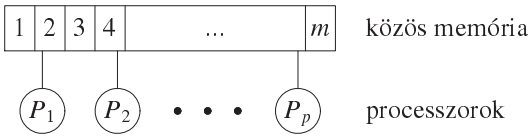
Eszerint a SISD a klasszikus soros, Neumann-elvű számítógép. A SIMD típusú számítógépben lépésenként egyféle művelet hajtódik végre, de az egyszerre több adaton. Az ILLIAC-IV számítógép példa erre a típusra.

A MISD típusú gépben egy adaton egyszerre többféle művelet hajtódik végre. A csővezeték-elvű számítógépek példák erre a típusra.

A legtöbb párhuzamos számítógép a MIMD típushoz tartozik: ebben az esetben több processzor dolgozik párhuzamosan, és rendszerint különböző adatokkal.

1.4.2. Párhuzamos gépek

A párhuzamos számítási modellek alapja a soros számításokhoz széles körben használt RAM (**R**andom **A**ccess **M**achine = *közvetlen hozzáférésű gép*) általánosítása, a PRAM (**P**arallel **R**andom **A**ccess **M**achine = ***párhuzamos közvetlen hozzáférésű gép***.) A PRAM modell tartalmaz p szinkronizáltan dolgozó processzort (P_1, P_2, $\ldots$, P_p) és az $M[1]$, $M[2]$, $\ldots$, $M[m]$ rekeszekből álló közös memóriát, ahogy azt az 1.1. ábra mutatja (az ábrán a memóriarekeszeknek csak az indexét tüntettük fel). A modell szerint minden processzor rendelkezik saját memóriával: a P_i processzor esetében ez az $M[i,1]$, $M[i,2]$, $\ldots$, $M[i,m]$ rekeszekből áll. Nem jelent megszorítást,



1.1. ábra. Párhuzamos közvetlen hozzáférésű gép (PRAM).

hogy a közös memória és a saját memóriák méretét ugyanúgy jelöltük (szokás ezeket a memóriákat végtelen méretűnek is tekinteni). Feltesszük, hogy a rekeszek tetszőleges egész szám tárolására alkalmasak.

A párhuzamos közvetlen hozzáférésű gép helyett rendszerint a rövidebb *párhuzamos gép* kifejezést fogjuk használni.

Típusok írás/olvasás alapján:

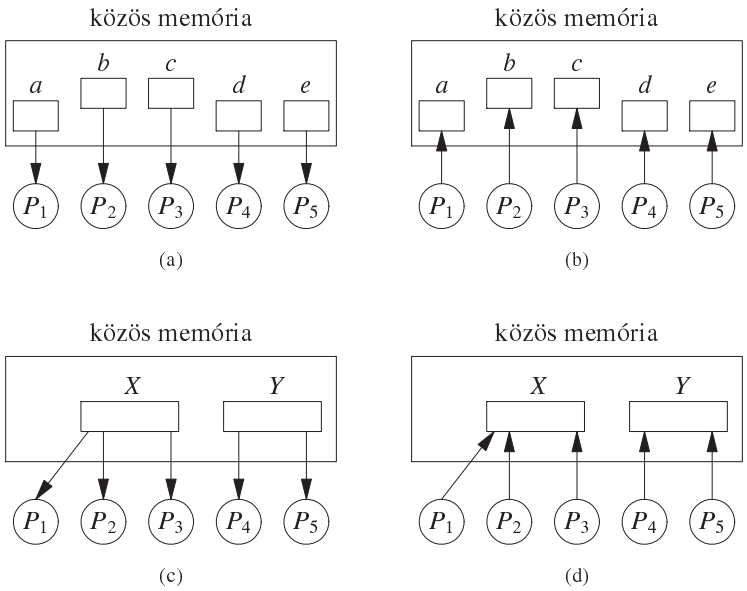
- EREW (**E**xclusive **R**ead – **E**xclusive **W**rite: kizárólagos olvasás – kizárólagos írás)
- ERCW (**E**xclusive **R**ead – **C**oncurrent **W**rite)
- CREW (**C**oncurrent **R**ead – **E**xclusive **W**rite)
- CRCW (**C**oncurrent **R**ead – **C**oncurrent **W**rite)

Ugyanabba a memóriarekeszbe egyidejűleg csak írás vagy olvasás van megengedve.

Az 1.2. ábra (a) része arra példa, hogy minden rekeszből legfeljebb egy processzor olvas (ER), a (b) részében minden rekeszbe legfeljebb egy processzor ír (EW), a (c) részében több processzor olvas párhuzamosan (CR), végül a (d) részében több processzor ír egyidejűleg ugyanabba a rekeszbe (CW).

Ha több processzor írhat egyidejűleg (ERCW vagy CRCW), akkor több eset van: az írás

- ***közös:*** a processzorok csak a közös, azonos értéket írhatják;
- ***tetszőleges:*** nem tudjuk, melyik processzor beírása marad meg a rekesz-



1.2. ábra. Számítási modellek típusa az írás és olvasás tulajdonságai alapján.

ben, vagy esetleg a beírásoktól független érték;

- ***prioritásos:*** a legnagyobb prioritású processzor ír;
- ***kombinált:*** az egyes processzorok által beírt értékek valamilyen függvénye kerül a memóriarekeszbe.

A következő példákban a párhuzamos olvasást, párhuzamos írást, illetve egy logikai érték párhuza

1.1. példa. *4 processzor párhuzamosan olvas.*

PÁRHUZAMOSAN-OLVAS($M[1]$) *párhuzamos eljárás*

Számítási modell: CREW PRAM

Bemenet: $A[1]$ (egy elemű tömb)

Kimenet: $M[1\,..\,4, 1]$ (processzorok saját memóriái első rekeszeinek közös tartalma)

```
01  P_i in parallel for i ← 1 to 4
02                    do M[i, 1] ← A[1]
```

Ebben az esetben mind a 4 processzor saját memóriájának első rekeszébe bekerül az $M[1]$ tömbelem.

A következő példában egy 16 processzoros gépben minden processzor a közös $M[1]$ rekeszbe ír.

1.2. példa. *16 processzor párhuzamosan ír.*

PÁRHUZAMOSAN-ÍR($M[1]$) *párhuzamos eljárás*
Számítási modell: közös ERCW PRAM
Bemenet: $M[1:16]$ (16 elemű tömb)
Kimenet: $M[1]$ (tömb egy eleme)

```
01  P_i in parallel for i ← 1 to 16
02                    do A[1] ← M[i, 1]
```

Ebben az esetben a 16 processzor párhuzamosan beírja az $A[1]$ tömbelembe saját memóriája első rekeszének tartalmát.

Legyen az $A[1:16]$ tömb $A[1],\ldots,A[n]$ elemeiben tárolt n bit logikai összege (diszjunkciója) $A[0] = A[1] \vee A[2] \vee \cdots \vee A[n]$.

Ekkor a következő ERCW algoritmus $O(1)$ időben dolgozik.

1.3. példa. *Logikai összeg kiszámítása n processzoron.*

LOGIKAI-ÖSSZEAD($p, A, A[0]$) *párhuzamos eljárás*
Számítási modell: ERCW PRAM
Bemenet: p (változók száma) és
$A[1..p]$ (a változókat tartalmazó tömb)
Kimenet: $A[0]$ (a bemenő változók logikai összege)

```
01  A[0] ← 0
02  P_i in parallel for i ← 1 to p
03                    do if A[i] = 1
04                          then A[0] ← A[i]
```

1.1. tétel (logikai összeadás művelet elvégzése). *A* Logikai-összead *algoritmus az n bites* $\vee$ *műveletet egy* ERCW PRAM*-en* $O(1)$ *lépés alatt elvégzi.*

Most a párhuzamos gépek néhány mennyiségi tulajdonságát mutatjuk be.

Feltesszük, hogy egy p processzoros gép bármely lépése szimulálható egy soros processzor p lépésével (vannak olyan valódi gépek, amelyekre ez a feltétel nem teljesül). Ebből adódik a következő állítás.

1.2. tétel (Brent tétele). *Ha egy feladatot a* $\mathcal{P}$ *párhuzamos algoritmus p processzoron t lépésben old meg, eközben az i-edik* $(i = 1, 2, \ldots, t)$ *lépésben* x_i *műveletet végez, akkor ez a feladat* $q < p$ *processzoron megoldható*

$$t + \left\lceil \frac{x}{q} \right\rceil \tag{1.31}$$

lépésben, ahol

$$x = \sum_{i=1}^{t} x_i. \tag{1.32}$$

Ebből a tételből adódik a következő egyszerű állítás.

1.3. következmény (lassulás). *Ha a* $\mathcal{P}$ *párhuzamos algoritmus egy p processzoros gépen t lépést tesz, akkor* $\mathcal{P}$ *minden* $q < p$ *processzort tartalmazó gépen végrehajtható* $O(\frac{pt}{q})$ *lépésben.*

Bizonyítás. A p processzoros G gépen futó $\mathcal{P}$ párhuzamos algoritmus minden lépése szimulálható egy q processzoros H gépen, legfeljebb $\lceil p/q \rceil$ lépést végezve. Ezért a H szimulációs futási ideje legfeljebb $t\lceil p/q \rceil$, és így H munkája legfeljebb

$$qt\left\lceil \frac{p}{q} \right\rceil \leq pt + qt \tag{1.33}$$

$$= O(\frac{pt}{q}) \tag{1.34}$$

lesz. ■

A következő 3 állítás az elérhető gyorsítás mértékével kapcsolatos.

1.4. tétel (Amdahl törvénye a maximális gyorsításról). *Ha egy π feladat megoldásának nem párhuzamosítható hányada $s(\pi)$, akkor egy p processzoros* PRAM*-on elérhető $g_{max}(\pi, s, p)$ legnagyobb gyorsítás*

$$g_{max}(\pi, s, p) = \frac{1}{s + \frac{1-s}{p}} . \tag{1.35}$$

1.5. tétel (Gustafson törvénye a maximális gyorsításról). *Ha egy π feladat megoldásának nem párhuzamosítható hányada s, akkor egy p processzoros* PRAM*-on elérhető $g_{max}(\pi, s, p)$ legnagyobb gyorsítás*

$$g_{max}(\pi, s, p) = \frac{s + p(1-s)}{s + (1-s)} \tag{1.36}$$

$$= s + p(1-s) . \tag{1.37}$$

Amdahl és Gustafson tételének bizonyítását meghagyjuk feladatnak (lásd az 1-2. feladatot).

Amdahl szerint s reciproka korlátot szab az elérhető gyorsításnak, míg Gustafson szerint a gyorsítás a processzorszámmal arányosan növelhető.

1.6. tétel (van-e p-nél nagyobb gyorsítás). *A p processzorszámnál nagyobb gyorsítás nem érhető el.*

Bizonyítás. Tegyük fel, hogy egy π problémára $W(n, \mathcal{A})$ a legjobb ismert soros végrehajtási idő. Ha van olyan $\mathcal{P}$ párhuzamos algoritmus, melynek gyorsítása p-nél nagyobb, akkor $pW(n, \mathcal{P}, p) < W(n, \mathcal{A})$. Mivel egy p processzoros gép egy lépésének szimulálása az 1 processzoros gépen legfeljebb p lépést igényel, ezért $\mathcal{P}$ munkája sorosan szimulálható legfeljebb $pW(n, \mathcal{P}, p)$ idő alatt, ami feltételünk szerint kisebb, mint $W(n, \mathcal{A})$. Ez ellentmond annak, hogy $\mathcal{A}$ a π megoldására ismert legjobb soros algoritmus. ■

1.4.3. Hálózatok

A számítási modellek másik típusát a hálózatok adják. Ezekben a processzorok nem a közös memórián keresztül érintkeznek, hanem adatátviteli vonalakon keresztül, amelyek jól szemléltethetők gráfok segítségével. A processzor és a csúcs szavakat szinonimaként használjuk – általában a szövegkörnyezethez jobban illeszkedőt választva.

Ez a tény egyúttal ad egy jó szempontot a hálózatok osztályozására: síkba rajzolható és síkba nem rajzolható hálózatokat különböztetünk meg.

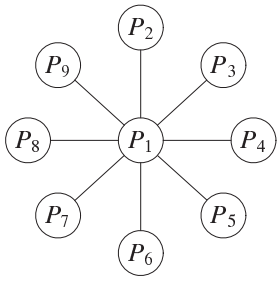
A gráfelmélet ismert eredménye, hogy minden véges gráf ábrázolható a 3-dimenziós euklideszi térben. Ezt beláthatjuk például úgy, hogy a $G = (V, E)$ véges gráf V_i ($i = 1, 2, \ldots, |V|$) csúcsát az x-y sík $(i, 0)$ pontjába rajzoljuk, majd a síkot az x-tengely körül rendre elforgatjuk $360j/|V|$ ($j = 1, 2, \ldots |V| - 1$) fokkal. Az így kapott S_j ($j = 1, 2, \ldots, |V| - 1$) síkokban rendre a V_j csúcsot a nála nagyobb indexű csúcsokkal összekötő éleket rajzoljuk meg.

A rövidség kedvéért a *síkba rajzolható* gráfokat ***síkgráfoknak,*** a síkba nem rajzolható gráfokat pedig ***térgráfoknak*** fogjuk nevezni. Ennek megfelelően beszélhetünk ***síkhálózatról*** és ***térhálózatról.***

A legismertebb hálózatok közül a síkhálózatokhoz tartozik például a csillag, fa, lánc, gyűrű, négyzet, kocka és a henger.

A p processzoros ***csillagban*** kitüntetett szerepe van a P_1 processzornak: ez van minden további processzorral összekötve. Az 1.3. ábra egy 9 processzoros csillagot mutat.

Egy d szintes (teljes) ***bináris fában*** $p = 2^d - 1$ processzor van: P_1, $P_2, \ldots, P_p$. Az adatszerkezetekkel kapcsolatos terminológia szerint a P_1 processzort ***gyökérnek,*** a $P_{(p-1/2}$, $P_{(p-1)/2+1}$, $\ldots$, P_p processzorokat ***levélnek,*** a többi processzort ***belső processzornak*** nevezzük. Ha P_i nem levél, akkor össze van kötve a ***gyerekeinek***ó nevezett P_{2i} és P_{2i+1} processzorokkal. Ha P_j nem a gyökér, akkor össze van kötve a ***szülőjének*** nevezett $P_{\lfloor j/2 \rfloor}$ pro-



1.3. ábra. 9 processzoros csillag.

cesszorral. A negyedik fejezetben lévő **??**. ábra egy háromszintes bináris fa hálózatot ábrázol. Hasonlóképpen lehetne m-áris fákat és nem teljes fákat is definiálni.

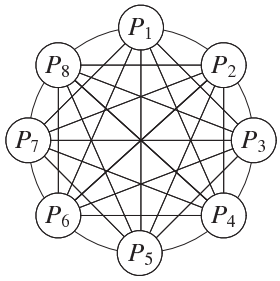
A p processzoros gyűrűs hálózatot a negyedik fejezetben ismertetjük. A **??**. ábra egy 6 processzoros ***gyűrűt*** mutat.

A térhálózatok közül megemlítjük a k dimenziós rácsot, tóruszt, piramist, pillangót, permutációs hálózatot, de Bruijn-hálózatot és a hiperkockát.

A k dimenziós ($k \geq 1$) rács egy olyan $m_1 \times m_2 \times \cdots \times m_k$ ($m_1, m_2, \ldots, m_k \geq 2$) méretű háló, amelynek minden egyes metszéspontjában van egy processzor. Az élek a kommunikációs vonalak, melyek kétirányúak. A rács minden processzorát megcímkézzük egy $(i_1, i_2, \ldots i_k)$ k-assal – erre a processzorra a $P_{i_1,\ldots,i_k}$ jelöléssel hivatkozunk.

Minden processzor egy RAM, amely rendelkezik saját (helyi) memóriával. A $P_{i_1,\ldots,i_k}$ processzor saját memóriája az $M[i_1, \ldots, i_k, 1]$, $M[i_1, \ldots, i_k, 2]$, $\ldots$, $M[i_1, \ldots, i_k, m]$ rekeszekből áll.

Minden processzor végre tud hajtani egyetlen lépésben olyan alapvető műveleteket, mint az összeadás, kivonás, szorzás, összehasonlítás, saját memória elérése és így tovább. A processzorok működése szinkron módon történik, azaz minden processzor egy globális óra ütemére egyszerre hajtja végre az aktuális feladatát.



1.4. ábra. 4×4 méretű henger.

A legegyszerűbb rács a $k = 1$ értékhez tartozó lánc alakú rács (röviden ***lánc.***)

Egy 6 processzoros lánc látható a harmadik fejezetben lévő **??**. ábrán. Ha egy lánc P_1 és P_p processzorát összekötjük, akkor gyűrűt kapunk.

Ha $k = 2$, akkor ***téglalapot*** (téglalap alakú rácsot) kapunk. Ha most $m_1 = m_2 = a$, akkor $a \times a$ méretű ***négyzetet*** kapunk. Egy 4×4 méretű négyzet látható a harmadik fejezetben lévő **??**. ábrán.

A lánc és a négyzet a síkhálózatokhoz tartoznak.

Ha $k = 3$ és $m_1 = m_2 = m_3$, akkor *téglát* kapunk. Ha ennek a hálózatnak a 3 mérete azonos, akkor *kockának* nevezzük. A 3.3. ábra egy $2 \times 2 \times 2$ méretű ***kockát*** ábrázol. Ez is ábrázolható síkban.

Ha egy rácsot további élekkel kiegészítünk, akkor ***összetett*** rácsot kapunk.

Ha egy láncban a P_1 és P_p processzorokat összekötjük, akkor gyűrűt kapunk, amely már csak két dimenzióban ábrázolható A negyedik fejezetben lévő 4.4. ábra egy 6 processzoros gyűrűt ábrázol.

Ha egy téglalapban a sorok első ($j = 1$) és utolsó ($j = m_2$) elemeit összekötjük, akkor az ugyancsak 2-dimenziós ***hengert*** kapjuk. Az 1.4. ábra egy 4×4 méretű hengert ábrázol.

Az ILLIAC-IV megépült változata olyan 2-dimenziós rácsból származ-

tatható, amelyben $m_1 = m_2 = 8$ és a $P_{i,j}$ processzor szomszédai rendre a $P_{i,j-1}$, $P_{i,j+1}$ $P_{i+1,j}$, $P_{i-1,j}$, ahol az indexek (mod 8) értendők. Az ILLIAC-IV architektúrájának ábrázolásához már három dimenzióra van szükség.

A ***tórusz*** például úgy származtatható egy kétdimenziós rácsból, hogy a belső élek mellett az egyes sorok első és utolsó processzorait, valamint az egyes oszlopok első és utolsó oszlopait is összekötjük. A negyedik fejezetben lévő 4.6. ábra egy 5×5 méretű tóruszt mutat.

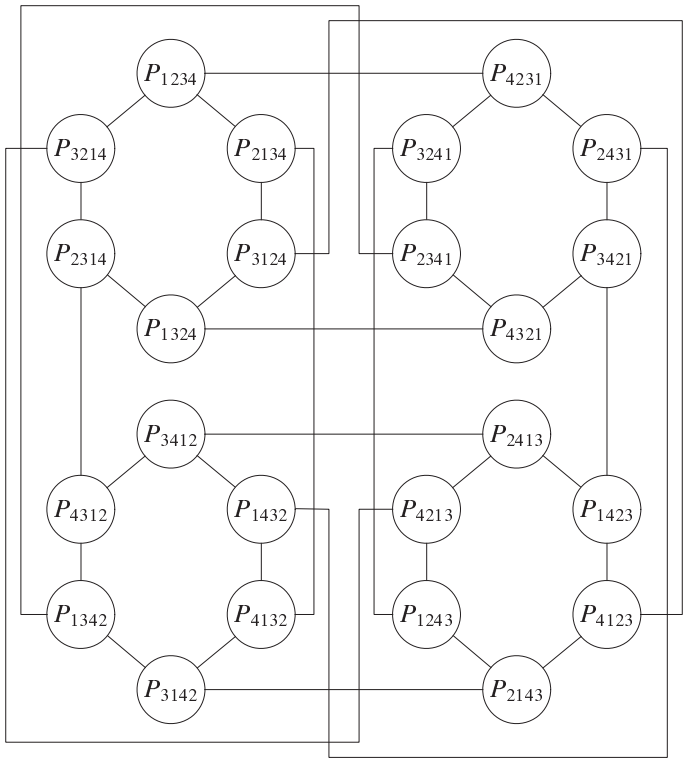
Egy ***d szintes piramis*** i-edik ($1 \leq i \leq d$) szintjén 4^{d-i-1} processzor van, amelyek az 1.5. ábra szerint vannak összekötve. Eszerint a piramis i-edik szintje egy $2^{d-i} \times 2^{d-i}$ méretű rács. A piramis az egyes szinteken lévő processzorok számát tekintve hasonló a ternáris fához, azonban a fa azonos szinten lévő processzorai között nincs kapcsolat.

A ***d dimenziós pillangó*** hálózat $(d+1)2^d$ processzorból áll, amelyek $d+1$ – egyenként 2^d processzort tartalmazó – sorban vannak elrendezve, ahogy azt a negyedik fejezetben lévő 4.2. ábra mutatja. A $(d+1)\times 2^d$ méretű pillangó kifejezést is használni fogjuk.

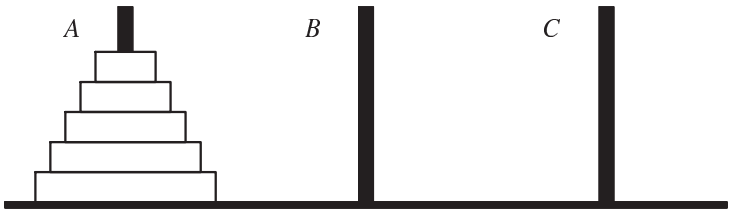
Természetes architektúra a ***teljes hálózat,*** amelyben minden processzor pár össze van kötve. Egy teljes hálózatot ábrázol az 1.6. ábra.

A (d, k)-paraméterű ***de Bruijn-hálózat*** d^k processzort tartalmaz, amelyek a d betűs $\{0, 1, \ldots, d-1\}$ ábécé feletti k hosszúságú szavakkal címezhetők. Az $a_1 a_2 \ldots a_k$ nevű processzorból az $a_2 a_3 \ldots a_k q$ című processzorokba vezet irányított él – ahol q az ábécé tetszőleges eleme. Eszerint minden processzorból d él indul, és minden processzorban d él végződik. Az 1.7. ábra egy (2,3)-paraméterű de Bruijn-hálózatot mutat. Az ábra alapján ez 2-dimenziós.

A d dimenziós ***permutációs hálózatban*** a processzorok a d-betűs $\{0, 1, \ldots, d-1\}$ ábécé elemeinek különböző permutációival címezhetők. Ezért a hálózatban $p = d!$ processzor van. A P_i processzor pontosan azokkal a processzorokkal van összekötve, amelyek címkéje előállítható úgy, hogy P_i címkéjében az első betűt a j-edik ($2 \leq j \leq d$) betűvel cseréljük ki. A 1.8. ábra



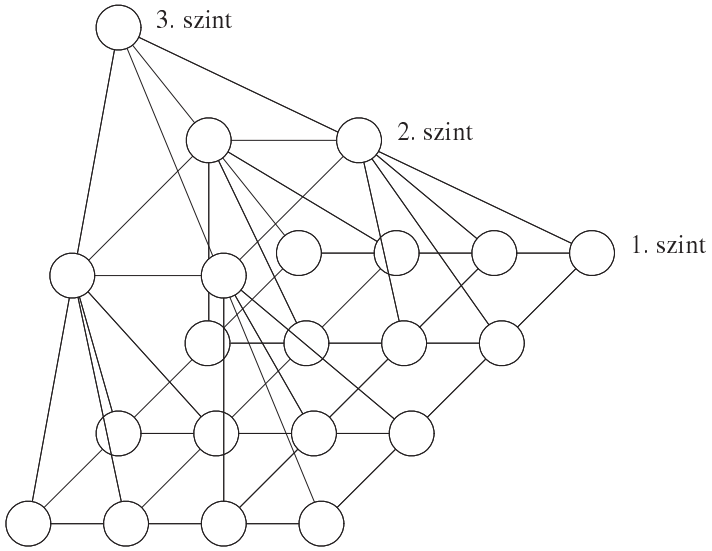
1.5. ábra. Háromszintes piramis.



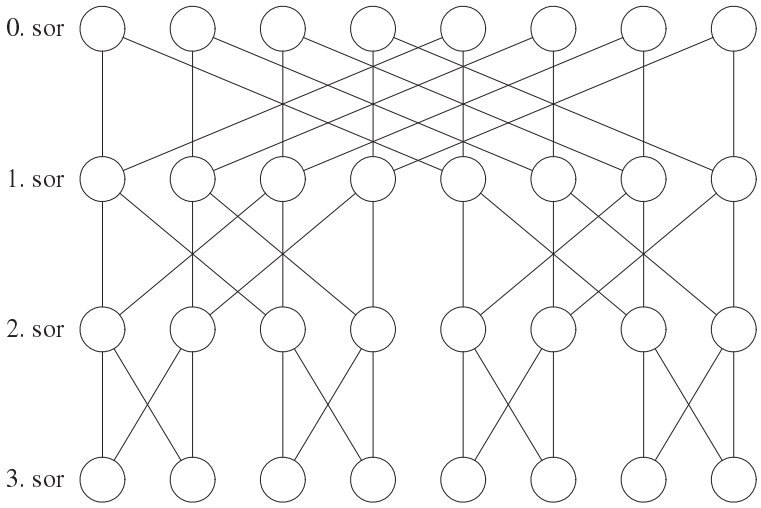
1.6. ábra. 6 processzoros teljes hálózat.

egy 4 paraméterű permutációs hálózatot ábrázol. Ebben minden processzor fokszáma $d - 1 = 3$ és a processzorok száma $4! = 24$.

A d dimenziós ***hiperkockában*** 2^d processzor van, amelyek d hosszúságú



1.7. ábra. 2×3 paraméterű de Bruijn-hálózat.



1.8. ábra. 24 processzoros permutációs hálózat.

bináris sorozatokkal címezhetők. Minden P_i processzor pontosan azokkal a processzorokkal van összekötve, amelyek címe pontosan egy bitben tér el P_i

címétől. A 3-dimenziós hiperkockában a 0,1,0 csúcs szomszédjai az 1,1,0, 0,0,0 és a 0,1,1 című csúcsok. Mivel a $2 \times 2 \times 2$ méretű kocka egyúttal 3-dimenziós hiperkocka is, a negyedik fejezetben lévő **??**. ábra egyúttal egy 3-dimenziós hiperkockát is bemutat. Az ugyancsak a negyedik fejezetben lévő 4.1. ábra pedig egy 4-dimenziós hiperkockát mutat.

A d dimenziós rácsok, permutációs hálózatok és hiperkockák természetes módon elképzelhetők és ábrázolhatók d dimenzióban. A korábban említett tétel szerint ugyanakkor tetszőlegesen nagy d esetében is ábrázolhatók a 3-dimenziós euklideszi térben úgy, hogy az élek ne metsszék egymást.

A hálózatok jellemzésére sokféle adatot használnak. Ha a H hálózatot $H = (V, E)$ formában, a csúcsok és élek halmazával adjuk meg, akkor természetes adat a ***processzorok száma*** ($|V|$) és az *adatátviteli vonalak száma* ($|E|$).

Az adatátviteli vonalak lehetnek egyirányúak és kétirányúak. Az eddigi példák közül csak a de Bruijn-hálózat tartalmazott egyirányú éleket (ezért a hálózatot irányított gráffal ábrázoljuk), míg a többi esetben kétirányú éleket tételeztünk fel.

További jellemző a csúcsok ***maximális, minimális*** és ***átlagos fokszáma,*** a csúcsok ***maximális, minimális*** és ***átlagos távolsága.***

A csúcsok maximális távolságát a hálózat ***átmérőjének*** nevezik.

Ha egy $H = (V, E)$ összefüggő hálózat csúcsait X és Y halmazra osztjuk, akkor a hálózat adott felbontáshoz tartozó ***vágási száma*** a legkisebb olyan élhalmaz elemszáma, amelynek eltávolításával a hálózat elveszti összefüggőségét.

Hálózatok ***felezési száma*** azon élek minimális száma, amelyek eltávolításával a hálózat két azonos részre bontható. Ha egy hálózat processzorainak száma páros ($p = 2k$), akkor a hálózat biztosan felbontható két azonos részre (például két, egyenként k izolált csúcsot tartalmazó hálózatra). Ha $p = 2k+1$, akkor a hálózat nem bontható két azonos részre. Ha $\sqrt{p}$ páros, akkor egy

Hálózat	Paraméter	Csúcsszám	Élszám	Átmérő
Lánc	p	p	$p-1$	$p-1$
Gyűrű	p	p	p	$\lfloor \frac{p}{2} \rfloor$
Csillag	p	p	$p-1$	2
Fa	d	2^d-1	2^d-2	2^d-2
Négyzet	$a=\sqrt{p}$	a^2	$2a(a-1)$	$2a-2$
Kocka	$a=\sqrt[3]{p}$	a^3	$3(a-1)a^2$	$3a-3$
Piramis	d	$\frac{4^d-1}{3}$	$2d$	2^d-2
Permutációs	d	$d!$	$\frac{d!(d-1)}{2}$	$2d-2$
De Bruijn	d	2^d	2^{d+1}	d
Hiperkocka	d	2^d	$d2^{d-1}$	d
Teljes	p	p	$p(p-1)/2$	$p-1$

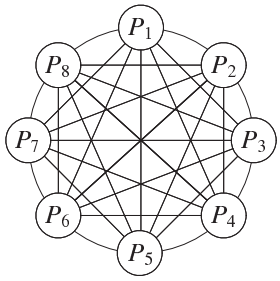
1.2. táblázat Hálózatok mennyiségi jellemzői.

$\sqrt{p} \times \sqrt{p}$ méretű rács vágási száma $\sqrt{p}$.

Az 1.2. táblázat az ismertetett hálózatok néhány alapvető adatát tartalmazza. A táblázat *Paraméter* oszlopában p a processzorszámot jelöli. d a fa és a piramis esetében a szintek számát, permutációs hálózat, de Bruijn hálózat és hiperkocka esetében a dimenziót jelöli. Négyzet és kocka esetében az oldalhosszúság a paraméter.

1.5. Rekurzió

Az algoritmusok megadásának gyakran alkalmazott módja az, hogy a végrehajtás során – változó paraméterekkel – újra és újra alkalmazzuk magát az algoritmust. Egy algoritmusnak önmaga segítségével való megadását ***rekurziónak*** az így megadott algoritmust pedig ***rekurzív algoritmusnak*** nevezzük.



1.9. ábra. Hanoi tornyai 3 rúddal és 5 koronggal.

Példaként oldjunk meg egy népszerű, 1883-ból származó, Hanoi híres tornyairól szóló feladatot, melyben korongok és 3 rúd szerepel. Brahma tornya 64 – közepén lyukas – arany korongból áll, melyet Brahma egy gyémánt rúdra tett. A korongok mérete alulról felfelé csökken. Az 1.9. ábra a kiinduló helyzetet mutatja 5 koronggal, melyek az A rúdon vannak. A B és C rudakon kezdetben nincs korong.

A Világ teremtésével egyidejűleg Brahma megbízott szerzeteseket azzal, hogy az egyik rúdon lévő korongokat helyezzék át egy másik rúdra – egy harmadik rúd felhasználásával. A szerzetesek munkájának egy lépése egy korong áthelyezése egyik rúdról egy másik rúdra – azzal a megszorítással, hogy korongot csak üres rúdra vagy nála nagyobb korongra szabad helyezni. A történet szerint amikor a szerzetesek végeznek a munkával, a torony összeomlik és vége a világnak.

Mennyi időnk van hátra, ha a szerzetesek a lehető legkevesebb lépésben elvégzik a munkát – és egy lépés egy másodpercig tart.

Jelöljük $f_K(n)$-nel az n korong áthelyezéséhez szükséges lépések számát egy K korong-áthelyező algoritmus esetében. Minden K algoritmusra igaz, hogy

$$f_K(1) \geq 1 \ . \tag{1.38}$$

n korongot csak úgy tudunk áthelyezni, hogy előbb a felső $n-1$ korongot

áthelyezzük egy másik rúdra, ezután a legalsó korongot áthelyezzük, és végül az $n-1$ kisebb korongot ráhelyezzük a legnagyobb korongra. Eszerint minden K algoritmusra

$$f_K(n) \geq 2f_K(n-1)+1\ . \tag{1.39}$$

Az ehhez hasonló egyenleteket – amelyekben egy függvény adott helyen felvett értékét más helyen vagy helyeken felvett értékei segítségével adjuk meg – ***rekurzív egyenletnek*** nevezzük.

Tegyük fel, hogy a szerzetesek a következő rekurzív program szerint dolgoznak, melyben az ÁTHELYEZ(k, *végez*) eljárás feladata az, hogy a k nevű rúdon lévő legfelső korongot áthelyezze a *végez* nevű rúdra.

HANOI-TORNYAI(n,k,*segít*,*végez*,*lépés*) *soros rekurzív eljárás*
Számítási modell: speciális
Bemenet: n (a korongok száma)
Kimenet: *lépés* (futási idő n korong áthelyezése során)

```
01  lépés ← 0
02  if n = 1
03        then ÁTHELYEZ(k, végez)
04             lépés ← lépés + 1
05        else HANOI-TORNYAI(n − 1, k, végez, segít, lépés)
06             ÁTHELYEZ(k, végez)
07             lépés ← lépés + 1
08             HANOI-TORNYAI(n − 1, segít, k, végez, lépés)
```

Eszerint a szerzetesek először – $f_{Sz}(1)$ lépés alatt – átrakják a legkisebb korongot a B rúdra, majd – $f_{Sz}(2) = 2f_{Sz}(1)+1 = 3$ lépés alatt – átrakják a két legkisebb korongot a C rúdra, majd – $f_{Sz}(3) = 2f_{Sz}(2)+1$ lépés alatt – átrakják a három legkisebb korongot a B rúdra és így tovább. Ebből következik, hogy

$$f_{Sz}(n) = 2f_{Sz}(n-1). \tag{1.40}$$

Mivel a szerzetesek minden részfeladatot a lehető legkevesebb lépésben megoldanak, algoritmusuk optimális. Módszerük futási idejét egyszerűen $f(n)$-nel jelölve a következő ***kezdeti feltételt*** és *rekurzív egyenletet* kapjuk:

$$f(1) = 1 \tag{1.41}$$

és

$$\text{ha } n \geq 2, \text{ akkor } f(n) = 2f(n-1) + 1. \tag{1.42}$$

Függvényeknek kezdeti feltétel (vagy feltételek) és rekurzív egyenlet segítségével történő megadását ***rekurziónak*** nevezzük. Eszerint a rekurzió szót kétféle – egymással összefüggő – jelentéssel használjuk. Megjegyezzük, hogy bizonyos esetekben a kezdeti feltételt nem akarjuk vagy tudjuk megadni – ilyenkor a rekurzív egyenlet rendszereint csak a megfelelő függvény nagyságrendjét határozza meg.

Teljes indukcióval és a ***rekurziótétellel*** (lásd például Halmos könyvét) bebizonyíthatjuk, hogy ennek a rekurzív egyenletnek a megoldása az

$$f(n) = 2^n - 1 \tag{1.43}$$

függvény.

Eszerint a világ élettartama

$$\begin{aligned} 2^{64} - 1 \text{ másodperc} &\approx 1.8447 \times 10^{19} \text{ másodperc} && (1.44)\\ &\approx 3.0745 \times 10^{17} \text{ perc} && (1.45)\\ &\approx 5.1242 \times 10^{15} \text{ óra} && (1.46)\\ &\approx 2.1351 \times 10^{14} \text{ nap} && (1.47)\\ &\approx 5.8495 \times 10^{11} \text{ év.} && (1.48) \end{aligned}$$

Tehát a korongok átrakása a szerzeteseknek körülbelül ötszáz milliárd évig tart. Eszerint még akkor is sok időnk lenne hátra a világ végéig, ha a szerzetesek 4,6 milliárd évvel ezelőtt (amikor a geológusok szerint a Föld

és a világegyetem az ősrobbanással létrejött) kezdték volna el a korongok átrakását.

Ez a példa több szempontból nagyon érdekes.

Mutatja, hogy – legalábbis bizonyos esetekben – elemi eszközökkel meg tudjuk határozni a rekurzív algoritmusok futási idejét.

Tegyük fel, hogy sok szerzetes dolgozik a korongok átrakásán, és s szerzetes már $1/s$ másodperc alatt átrak egy korongot, vagy pedig a lépéseket egyenletesen fel tudják egymás között osztani. Ekkor az algoritmus futási ideje az

$$f(n, s) = \frac{2^n - 1}{s} \tag{1.49}$$

értékre csökken. Ezzel elvben tetszőleges kicsire csökkenthető az átrakási idő. De hogyan tud a sok szerzetes hozzáférni a korongokhoz?

A kérdés rávilágít arra, hogy egy feladat részfeladatokra bontásának lehetőségeit gyakran korlátozzák a feladat sajátosságai.

Ha például megengedjük, hogy a szerzetesek felemeljék a felső $n - 1$ korongot és a legalsót körcikkekre vágva áthelyezzék, akkor megfelelő számú szerzetes esetében lineáris (vagy akár konstans) futási idejű algoritmust kaphatunk. Ezzel a megközelítéssel a könyvben nem foglalkozunk: a párhuzamos processzorok csak olyan műveleteket végezhetnek, amelyet a soros is végre tud hajtani.

Másik kérdés, mit érünk azzal, ha a gyakorlatilag végtelen futási időt például század vagy ezred részére csökkentjük? Ez a kérdés pedig arra világít rá, hogy a sokprocesszoros rendszerek és párhuzamos algoritmusok lehetőségei is korlátozottak.

A feladat megoldása azt is mutatja, hogy ha csak a futási időre van szükségünk, az (1.43) képletet levezetve és alkalmazva nagyon gyorsan meg tudjuk mondani.

1.6. Véletlenített algoritmusok (⋆)

A véletlenített algoritmusok bizonyos helyzetekben több döntés közül választanak – ezek valószínűsége pontosan adott.

Két típusuk van. A ***Las Vegas algoritmusok*** mindig helyes eredményt adnak – futási idejük azonban nem állandó, hanem egy valószínűségi változóval jellemezhető.

A ***Monte Carlo algoritmusok*** adhatnak hibás eredményt is.

A Las Vegas algoritmusok futási ideje tág határok között változhat, míg a Monte Carlo algoritmusoké viszonylag állandó.

A véletlenített algoritmusokat tekinthetjük algoritmushalmaznak. Adott bemenet esetében bizonyos algoritmusok futhatnak sokáig vagy adhatnak hibás választ. A tervezés célja, hogy az ilyen *rossz algoritmusok* algoritmusok hányada kicsi legyen. Ha meg tudjuk mutatni, hogy *bármilyen* bemenetre az algoritmusoknak legalább $1-\epsilon$ hányada (ahol a pozitív ϵ elég kicsi) gyors (illetve helyes eredményt ad), akkor a halmazból véletlenül választott algoritmusra igaz, hogy legalább $1-\epsilon$ valószínűséggel gyors (helyes eredményt ad). Ekkor ϵ-t *hibavalószínűségnek* nevezzük.

Az alábbi definíciókban a d, v és g függvények a pozitív egészek halmazán vannak értelmezve, a $d(n)$, $v(n)$ és $g(n)$ függvényértékek pedig nemnegatív valós számok.

Egy D determinisztikus algoritmusnak adott erőforrásból $d(n)$ egységre van szüksége, míg a V véletlenített algoritmusnak $v(n)$ egységre – ahol $v(n)$ valószínűségi változó.

Egy n-től függő ϵ esemény ***nagy valószínűséggel*** bekövetkezik, ha tetszőleges α értékre a bekövetkezés valószínűsége legalább $1-n^{-\alpha}$. Az α számot ***valószínűségi paraméternek*** nevezzük.

$\overline{O}(g(n))$ (ejtsd: ***nagy valószínűséggel nagy ordó***) azon $v(n)$ függvények halmazát jelenti, amelyekhez létezik olyan c pozitív állandó, hogy tetszőle-

ges α esetében nagy valószínűséggel teljesül

$$P[v(n) \leq c\alpha g(n)] \geq 1 - \frac{1}{n^{\alpha}} \ . \tag{1.50}$$

1.6.1. Egyenlőtlenségek

Ebben a részben olyan egyenlőtlenségeket adunk meg, amelyek azt jellemzik, hogy a valószínűségi változók ritkán térnek el lényegesen a várható értéküktől.

Ezek az egyenlőtlenségek majd hasznosak lesznek a véletlenített algoritmusok várható viselkedésének elemzése során.

1.7. lemma (Markov-egyenlőtlenség). *Ha X nemnegatív valószínűségi változó, melynek várható értéke μ, akkor*

$$P[X \geq x] \leq \frac{\mu}{x} \ . \tag{1.51}$$

A ***Bernoulli-kísérletnek*** 2 lehetséges eredménye van: p valószínűséggel sikeres, $1 - p$ valószínűséggel sikertelen a kísérlet. Ha X jelöli azt, hogy n kísérletből hány sikeres, akkor X eloszlása binomiális:

$$P[X = i] = \binom{n}{i} p^i (1-p)^{n-i} \ . \tag{1.52}$$

1.8. lemma (Csernov-egyenlőtlenségek). *Ha X (n, p) paraméterű binomiális eloszlású valószínűségi változó és $m > np$ egész szám, akkor*

$$P[X \geq m] \leq \left(\frac{np}{m}\right)^m e^{m-np} \ . \tag{1.53}$$

Továbbá minden $0 < \epsilon < 1$ számra

$$P[X \leq \lfloor (1-\epsilon)pn \rfloor] \leq e^{-\epsilon^2 np/2} \tag{1.54}$$

és

$$P[X \geq \lceil (1+\epsilon)np \rceil] \leq e^{-\epsilon^2 np/3} \ . \tag{1.55}$$

1.6.2. Példák

Ebben a pontban két példát mutatunk véletlenített algoritmusra.

1.4. példa. *Ismétlődő tömbelem meghatározása.* Tegyük fel, hogy n páros, az $A[1:n]$ tömb elemei között az $1, 2, \ldots, n/2+1$ számok egyike $(n/2)$-ször fordul elő, a többi szám pedig egyszer.

Határozzuk meg a többször előforduló elemet.

Bármely determinisztikus algoritmusnak legrosszabb esetben legalább $n/2+2$ lépésre van szüksége. Ugyanis bármely determinisztikus algoritmushoz megadható úgy a bemenet, hogy az első $n/2+1$ vizsgált elem különböző legyen.

Most megadunk egy Las Vegas algoritmust, amely legrosszabb esetben $\overline{O}(\lg n)$ ideig fut. Az algoritmus bemenete egy n szám és egy n-elemű A tömb, kimenete az egyik ismételt elem indexe.

ISMÉTELT-ELEM(A, n) *soros eljárás*

Számítási modell: RAM

Bemenet: n pozitív egész (az elemek száma) és
$A[1:n]$ (a vizsgálandó elemek)

Kimenet: *ismételt* (az egyik ismételt elem indexe)

```
01  L ← igaz
02  while L
03        i ← VÉLETLEN(n)
04        j ← VÉLETLEN(n)
05        ▷ i, j ∈ {1, 2, ..., n}, véletlen egész számok
06        if i ≠ j ∧ (a[i] = a[j])
07          then ismételt ← i
08               L ← HAMIS   09  return ismételt
```

Ebben az algoritmusban felhasználjuk a VÉLETLEN(n) eljárást, amely minden hívásakor a $[1:n]$ intervallumból vett egyenletes eloszlású, véletlen egész számot ad kimenetként.

Most megmutatjuk, hogy az algoritmus futási ideje $\overline{O}(\lg n)$. A **while** ciklus bármely végrehajtása sikeres, ha i az $n/2$ azonos elem valamelyikének indexe és j egy tőle különböző helyen lévő azonos elem indexe. Ennek az eseménynek a P valószínűsége

$$P = \frac{\frac{n}{2}(\frac{n}{2}-1)}{n^2}\ , \tag{1.56}$$

ami $n > 10$ esetében kisebb, mint $1/5$. Tehát annak valószínűsége, hogy az algoritmus egy adott lépésben nem fejeződik be, $(1/5)$-nél kisebb.

Ezért annak valószínűsége, hogy az algoritmus 10 lépés alatt nem fejeződik be, $(4/5)^{10}$-nél kisebb, ami kisebb 0.1074-nél. Ezért az algoritmus 10 lépés alatt legalább 0.8926 valószínűséggel befejeződik. Annak valószínűsége, hogy 100 lépés alatt sem fejeződik be, kisebb $(4/5)^{100}$-nál, ami pedig kisebb $2.04 * 10^{-10}$-nél.

Általában annak az ϵ eseménynek a valószínűsége, hogy az algoritmus az első $c\alpha \lg n$ (ahol a c konstans rögzített érték) lépésben nem fejeződik be,

$$P[\epsilon] < \left(\frac{4}{5}\right)^{c\alpha \lg n}\ , \tag{1.57}$$

és

$$\text{ha } c \geq \frac{1}{\lg \frac{5}{4}},\ \text{akkor } \left(\frac{4}{5}\right)^{c\alpha \lg n} < n^{-\alpha}\ . \tag{1.58}$$

Tehát az algoritmus legalább $1 - n^{-\alpha}$ valószínűséggel befejeződik legfeljebb

$$\frac{\alpha \lg n}{\lg \frac{5}{4}} \tag{1.59}$$

lépésben. Mivel minden iteráció $O(1)$ ideig tart, ezért az algoritmus futási ideje valóban $\overline{O}(\lg n)$.

Mivel algoritmusunk mindig helyes eredményt ad, ezért Las Vegas típusú. Megmutattuk, hogy nagy valószínűséggel gyorsan befejeződik.

Ha például kétmillió elemünk van, akkor bármely $\mathcal{D}$ determinisztikus algoritmushoz megadhatunk olyan bemenetet, amelyre $\mathcal{D}$ legalább egymillió

lépést végez, Az Ismételt-elem algoritmusnak viszont bármely bemenetre – nagy valószínűséggel – legfeljebb száz lépésre van szüksége.

Ugyanakkor az Ismételt-elem algoritmus – igaz, csak kis valószínűséggel – egymilliónál lényegesen több lépést is végezhet.

Az adott problémát másképp is megoldhatjuk: először rendezünk, azután lineárisan keresünk – ekkor $\Omega(n)$ időre van szükség. Vagy hármas csoportokra osztjuk a tömböt – ekkor $\Theta(n)$ a futási idő.

1.5. példa. *Pénzérme ismételt feldobása.* Dobjunk fel egy szabályos pénzérmét ezerszer. Mekkora annak valószínűsége, hogy legalább hatszázszor fejet dobunk?

A Markov-egyenlőtlenség szerint legfeljebb 5/6.

Az (1.55) képletet, azaz a harmadik Csernov-egyenlőtlenséget is alkalmazhatjuk az $n = 1000$, $p = 1/2$, $\epsilon = 0.2$ értékekkel. Eszerint

$$\begin{aligned} P[X \geq 600] &\leq e^{-(0,2)^2(500/3)} && (1.60)\\ &= e^{-20/3} && (1.61)\\ &\leq 0.001273\ . && (1.62) \end{aligned}$$

Meghagyjuk gyakorlatnak az ennél pontosabb becslést.

1.7. Alsó korlátok

Az algoritmusok elemzése során számos korábbi tankönyv szerzői (a múlt század hetvenes és a nyolcvanas éveiben) megelégedtek azzal, hogy többékevésbé pontos felső korlátokat adtak az adott algoritmus által igényelt erőforrások legnagyobb és átlagos mennyiségére.

Ezeknek a becsléseknek a pontossága azonban gyakran homályban maradt. Pedig e nélkül megválaszolatlan marad az a fontos kérdés, hogy érdemes-e jobb algoritmust keresni.

Különösen fontos az erőforrásigény pontos ismerete a párhuzamos algoritmusok vizsgálata során, hiszen e nélkül nem tudjuk eldönteni, hogy adott

párhuzamos algoritmus munkahatékony-e, munkaoptimális-e.

A munkahatékonyság és munkaoptimalitás bizonyításához a párhuzamos algoritmus igényét felülről, a feladat jellegéből fakadó igényt pedig alulról kell becsülnünk.

A munkahatékonyság cáfolásához pedig elég egy jó soros algoritmus igényének felülről, a párhuzamos algoritmus igényének pedig alulról való becslése.

Ebben az alfejezetben olyan módszereket mutatunk be, melyek segítségével alsó korlátokat bizonyíthatunk. Ez az alfejezet bevezető jellegű. A későbbiekben a számítási modellekhez és problémákhoz kapcsolódva további alsó korlátokat adunk meg.

1.7.1. Egyszerű számolás

Tegyük fel, hogy adott egy n méretű $L[1\,..\,n]$ lista és feladatunk a lista legnagyobb elemének megkeresése úgy, hogy elempárok összehasonlítása a megengedett művelet.

Ha $A(n)$-nel jelöljük az A kereső algoritmus által elvégzett összehasonlítások számát, akkor tetszőleges A összehasonlítás alapú algoritmusra teljesül, hogy

$$A(n) \geq N(n) \tag{1.63}$$

$$\geq \left\lceil \frac{n}{2} \right\rceil . \tag{1.64}$$

Megmutatjuk, hogy ennél több is igaz.

Az egyszerűség kedvéért tegyük fel, hogy a lista különböző elemeket tartalmaz. Amikor egy A összehasonlítás alapú algoritmus összehasonlítja az X és Y elemet, akkor az $X > Y$ esetben azt mondjuk, hogy X megnyerte az összehasonlítást – ellenkező esetben pedig azt mondjuk, hogy X elvesztette az összehasonlítást. Tehát minden összehasonlítás egy vereséget eredményez. Mivel mindazon elemeknek, amelyek nem a legnagyobbak, lega-

lább egy összehasonlítást el kell veszíteniük, ezért n elem legnagyobbikának meghatározásához

$$B_A(n) \geq n - 1 \tag{1.65}$$

összehasonlításra van szükség.

Ennek szigorú bizonyításához tegyük fel, A úgy fejezi be a keresést, hogy az $L[1\,..\,n]$ lista két különböző eleme, például $L[i]$ és $L[j]$, egyetlen összehasonlítást sem veszítettek el. Feltehetjük, hogy A az $L[i]$ elemet adta meg legnagyobbként. Most tekintsük azt az $L'[1\,..\,n]$ bemenő listát, amely csak annyiban különbözik az előzőtől, hogy abban $L'[j]$ nagyobb, mint $L[j]$. Mivel A összehasonlítás alapú, ezért ugyanolyan összehasonlításokat végez L és L' esetében. Ez következik abból, hogy különbség legfeljebb akkor fordulhatna elő, amikor $L'[j]$ is részt vesz az összehasonlításban. $L[j]$ azonban minden összehasonlítást megnyert és $L'[j] > L[j]$. Tehát az A algoritmusnak ismét az $L'[i] = L[i]$ elemet kell a legnagyobbnak nyilvánítania, ami ellentmondás.

Például az alábbi közismert program nemcsak legjobb, hanem legrosszabb esetben is $n - 1$ összehasonlítással meghatározza az $L[1 : n]$ lista maximális elemét.

MAX($n, L[1 : n], legnagyobb$) *soros eljárás*
Számítási modell: RAM
Bemenet: n (az elemek száma) és $L[1 : n]$ (n hosszúságú,különböző elemeket tartalmazó lista)
Kimenet: *legnagyobb* (a bemeneti lista egyik maximális eleme)

```
01  legnagyobb ← L[1]              ▷ kezdeti érték beállítása
02  for i ← 2 to n
03      if L[i] > legnagyobb
04          then legnagyobb ← L[i]
05                    ▷ legnagyobb frissítése
```

Tehát beláttuk, hogy MAX az összehasonlítás alapú maximumkeresést a feladat megoldásához okvetlenül szükséges számú összehasonlítással megoldja.

Azokat az algoritmusokat, amelyekre a legjobb és a legrosszabb futási idő megegyezik, azaz amelyekre

$$W(n) = B(n) \,, \tag{1.66}$$

stabilnak nevezzük. Megállapításainkat a következőképpen összegezhetjük.

1.9. tétel. (MAX *abszolút optimális.*) *Ha különböző elemeket tartalmazó, n hosszúságú lista maximális elemét összehasonlítás alapú algoritmussal határozzuk meg, akkor ennek a problémának az időbonyolultsága legjobb, legrosszabb és átlagos esetben is $n-1$. A feladat megoldására* MAX *abszolút optimális algoritmus.*

Érdemes megemlíteni, hogy a pontos bonyolultságot csak nagyon ritkán tudjuk meghatározni.

1.7.2. Leszámlálás

A leszámlálási módszereket leggyakrabban az átlagos jellemzők meghatározására használjuk. Minden I bemenethez hozzárendelünk egy $\alpha(I)$ jellemző adatot, majd leszámláljuk, hogy mennyi lesz ezen jellemző adatok összege az összes lehetséges bemenetre nézve.

Tekintsük például azt a feladatot, hogy az $1, 2, \ldots, n$ számok különböző permutációit kell rendeznünk úgy, hogy a permutációban szomszédos elemek összehasonlítása (és egyúttal cseréje) a felhasználható művelet. Minden I permutációhoz $\alpha(I)$-ként hozzárendeljük a benne lévő *inverziók* számát és leszámláljuk, hogy összesen hány inverzió van a permutációkban.

1.10. tétel. *Ha $\mathcal{A}$ egy különböző elemekből álló sorozatot a szomszédos ele-*

mek összehasonlításával rendező algoritmus, akkor

$$N(n, \mathcal{A}) \geq \frac{n(n-1)}{4} \,. \tag{1.67}$$

Bizonyítás. Ha $n = 1$, akkor igaz az állítás.

Ha $n \geq 2$, akkor rendeljük minden permutációhoz az ***inverzét,*** azaz azt a sorozatot, amelyben az elemek fordított sorrendben következnek. Mivel tetszőleges i és j indexre $(1 \leq i,\ j \leq n)$ igaz, hogy az i-edik és a j-edik elem az egymáshoz rendelt permutációk közül *pontosan* az egyikben van inverzióban, ezért minden párra igaz, hogy bennük együttesen annyi inverzió van, ahány pár (külön-külön), azaz $\binom{n}{2}$. Ezért n elem permutációiban összesen

$$\frac{n!}{2}\binom{n}{2} \tag{1.68}$$

inverzió van. Ha ezt a számot elosztjuk n elem permutációinak számával, megkapjuk a kívánt alsó korlátot. ■

1.7.3. Döntési fák

Egy ***döntési fa*** segítségével modellezhetjük az összes döntést, amelyet egy determinisztikus algoritmus végezhet. Egy adott bemenethez tartozó döntések megadnak egy irányított utat a döntési fa gyökerétől valamelyik leveléig, amely megadja az algoritmusnak az adott bemenethez tartozó kimenetét. A döntési fa csak azokat a csúcsokat tartalmazza, amelyekbe legalább egy bemenet esetében eljutunk. A belső csúcsok megfelelnek az algoritmus döntéseinek és az ahhoz tartozó alapműveleteknek (mint például két elem összehasonlítása vagy egy mátrix két elemének összeszorzása). Mivel minden belső csúcshoz legalább egy elvégzendő művelet tartozik, ezért a fa magassága alsó korlát $N(n, A)$-ra.

Hasonlóképpen a levelek ***átlagos szintje*** alsó korlát $A(n, \mathcal{A})$-ra.

Bizonyítás nélkül idézünk néhány közismert eredményt, amelyek például

döntési fák segítségével igazolhatók.

1.11. tétel (alsó korlát a bináris keresés futási idejére). *Ha* BINÁRISAN-KERES *a rendezett sorozatban binárisan kereső algoritmus, akkor minden pozitív n számra*

$$N(n, \text{BINÁRISAN-KERES}) \geq \lceil \lg n \rceil + 1 \ . \tag{1.69}$$

1.12. tétel. (*Alsó korlát az összehasonlítás alapú rendező algoritmusok futási idejére.*) *Ha* ÖSSZEHASONLÍT *összehasonlítás alapú rendező algoritmus, akkor*

$$\begin{aligned} N(n, \text{ÖSSZEHASONLÍT}) &\geq \lceil \lg n! \rceil & (1.70) \\ &\geq n \lg n + \frac{\lg n}{2} + O(1) \ . & (1.71) \end{aligned}$$

1.7.4. Tanácsadói érvelés

Ennél a módszernél úgy jutunk alsó korláthoz, hogy dinamikusan olyan bemenetet állítunk elő, amely az algoritmust lehetőleg nagyszámú művelet elvégzésére kényszeríti.

Ez a módszer olyan játékhoz hasonlít, amelyben egy tanácsadónak feltett kérdésekkel juthatunk információhoz, és a tanácsadó arra törekszik, hogy a válaszaival minél kevesebb információt adjon.

A $W(n, \mathcal{A})$ futási idő alsó becsléséhez úgy jutunk, hogy összeszámoljuk, hány műveletet kellett az adott bemenetre a vizsgált algoritmusnak elvégeznie.

1.7.5. Információelméleti érvelés

Ez a módszer azon alapul, hogy az egy művelettel nyerhető információra felső, a feladat megoldásához szükséges információ mennyiségére pedig alsó korlátot adunk. Ha például egy összehasonlítás lehetséges eredményei **true** és **false**, akkor n összehasonlítással legfeljebb 2^n lehetőséget tudunk megkü-

lönböztetni. Mivel egy n elemet rendező, összehasonlítás alapú algoritmusnak $n!$ lehetőséget kell megkülönböztetnie, ezzel a módszerrel is bizonyíthatjuk a 1.12. tételt.

1.7.6. Gráfelméleti érvelés

Mivel a hálózatokat rendszerint gráfokkal modellezzük, ezért természetes, hogy az alsó korlátok bizonyításában is gyakran szerepelnek gráfok.

Erre a későbbiek során több példát is mutatunk.

1.8. Anomália

A számítógépes rendszerekben *anomáliának* nevezzük azt a jelenséget, amikor egy feladat megoldásához több erőforrást felhasználva rosszabb eredményt kapunk.

Négy konkrét példát említünk. Az egyik a virtuális memória lapjait párhuzamosan használó FIFO (**F**irst **I**n – **F**irst **O**ut) lapcserélési algoritmussal, a másik a processzorok ütemezésére használt LISTÁSAN-ÜTEMEZ algoritmussal, a harmadik az átfedéses memóriájú számítógépekben folyó párhuzamos programvégrehajtással, végül a negyedik a PÁRH-KORLÁTOZ-SZÉTVÁLASZT optimalizációs algoritmussal kapcsolatos.

1.8.1. Lapcsere

Legyenek m, M, n és p pozitív egészek ($1 \leq m \leq M \leq n < \infty$), k nemnegatív egész, $A = \{a_1, a_2, \ldots, a_n\}$ egy véges ábécé. A^k az A feletti, k hosszúságú, A^* pedig az A feletti véges szavak halmaza.

Legyen m egy *kis*, M pedig egy *nagy* számítógép fizikai memóriájában lévő lapkeretek száma, n a háttérmemóriában lévő lapok száma (mindkét számítógépben), A a lapok halmaza.

A lapcserélési algoritmusokat automatákként kezeljük, melyekre m és M

a memória mérete, A a bemenő jelek halmaza, $Y = A \cup \{\epsilon\}$ a kimenő jelek halmaza. Ezek az automaták a bemenő jelek $R = (r_1, r_2, \ldots, r_p)$ vagy $R = (r_1, r_2, \ldots)$ sorozatát dolgozzák fel. Az S_t $(t = 1, 2, \ldots,)$ memóriaállapot a t időpontban (azaz az r_t bemenő jel feldolgozása után) a memóriában tárolt bemenő jelek halmaza. A lapcserélési algoritmusok $S_0 = \{\}$ üres memóriával kezdik a feldolgozást. Egy konkrét P lapcserélési algoritmust a $P = (Q_P, q_0, g_P)$ hármassal definiálunk, ahol Q_P a vezérlő jelek halmaza, $q_0 \in Q_P$ a kezdeti vezérlő jel, g_P az állapot-átmenet függvény, S' az új memóriaállapot, q' az új állapot és y a kimenő jel, S a régi memóriaállapot, q a régi vezérlőállapot és x a bemenő jel.

Az F = FIFO (First In First Out) algoritmus definíciója: $q_0 = ()$ és

$$g_{\mathrm{F}}(S, q, x) = \begin{cases} (S, q, \epsilon), & \text{ha } x \in S, \\ (S \cup \{x\}, q', \epsilon), & \text{ha } x \notin S,\ |S| < m, \\ (S \setminus \{y_1\} \cup \{x\}, q", y_1), & \text{ha } x \notin S \text{ és } |S| = m, \end{cases} \tag{1.72}$$

ahol $q = (y_1, y_2, \ldots, y_k)$, $q' = (y_1, y_2, \ldots, y_k, x)$ és $q'' = (y_2, y_3, \ldots, y_m, x)$.

A laphibáknak (a memóriaállapot változásainak) a számát $f_P(R, m)$-vel jelöljük. *Anomáliának* nevezzük azt a jelenséget, amikor $M > m$ és $f_P(R, M) > f_P(R, m)$. Ekkor az $f_P(R, M)/f_P(R, m)$ hányados az ***anomália mértéke.***

A P algoritmus hatékonyságát az $E_P(R, m)$ ***lapozási sebességgel*** jellemezzük, amit $R = (r_1, r_2, \ldots, r_p)$ véges hivatkozási sorozatra az

$$E_P(R, m) = \frac{f_P(R, m)}{p}, \tag{1.73}$$

$R = (r_1, r_2, \ldots)$ végtelen hivatkozási sorozatra pedig a

$$E_P(R, m) = \liminf_{k \to \infty} \frac{f_P(R_k, m)}{k} \tag{1.74}$$

módon definiálunk, ahol $R_k = (r_1, r_2, \ldots, r_k)$.

Legyen $1 \le m < n$ és $C = (1, 2, \ldots, n)^*$ egy végtelen ciklikus hivatkozási

sorozat. Ekkor $E_{\mathrm{FIFO}}(C,m) = 1$.

Ha végrehajtjuk a $R = (1,2,3,4,1,2,5,1,2,3,4,5)$ hivatkozási sorozatot, akkor $m = 3$ esetében 9, $m = 4$ esetében pedig 10 laphibát kapunk, így $f_{\mathrm{FIFO}}(R,M)/f_{\mathrm{FIFO}}(R,m) = 10/9$.

Bélády, Nelson és Shedler a következő szükséges és elégséges feltételt adták az anomália létezésére.

1.13. tétel. *Akkor és csak akkor létezik olyan hivatkozási sorozat, amelyre a* FIFO *lapcserélési algoritmus anomáliát okoz, ha* $m < M < 2m - 1$.

Az anomália mértékével kapcsolatban pedig a következőt bizonyították.

1.14. tétel. *Ha* $m < M < 2m - 1$, *akkor tetszőleges* $\epsilon > 0$ *számhoz létezik olyan* $R = (r_1, r_2, \ldots, r_p)$ *hivatkozási sorozat, amelyre*

$$\frac{f_{FIFO}(R,M)}{f_{FIFO}(R,m)} > 2 - \epsilon\,. \tag{1.75}$$

Bélády, Nelson és Shedler a következőt sejtették.

1.15. sejtés. *Tetszőleges* R *hivatkozási sorozatra és* $M > m \geq 1$ *memóriaméretekre*

$$\frac{f_{FIFO}(R,M)}{f_{FIFO}(R,m)} \leq 2. \tag{1.76}$$

Ezt a sejtést cáfolja Fornai Péter és Iványi Antal következő tétele, amely szerint az anomália mértéke tetszőlegesen nagy lehet.

1.16. tétel. *Tetszőlegesen nagy* L *számhoz megadhatók olyan* m, M *és* R *paraméterek, melyekre*

$$\frac{f_{FIFO}(R,M)}{f_{FIFO}(R,m)} > L\,. \tag{1.77}$$

1.8.2. Ütemezés

Tegyük fel, hogy n programot akarunk végrehajtani egy p processzoros láncon. A végrehajtásnak figyelembe kell vennie a programok közötti megelőzési relációt. A processzorok mohók, és a végrehajtás egy adott L lista szerint történik.

E. G. Coffman jr. 1976-ban leírta, hogy a p processzorszám csökkenése, az egyes programok végrehajtásához szükséges lépések t_i számának csökkenése, a megelőzési korlátozások enyhítése és a lista változtatása külön is anomáliát okozhat.

Legyen a programok végrehajtásának futási ideje τ, a megelőzési reláció $<$, a lista L és a programok közös listás végrehajtásához szükséges lépések száma p azonos processzorból álló láncon $\omega(p, L, <, \tau)$. Az L' lista, $<' \subseteq <$ megelőzési reláció, $\tau' \leq \tau$ futási idő vektor és $p' \geq p$ processzorszám esetén a futási idő legyen $\omega'(p', L', <', \tau')$.

Az anomália mértékét ezúttal a gyorsítással jellemezzük.

1.17. tétel (ütemezési korlát). *Az előbbi feltételek esetében*

$$0 \leq \frac{\omega'}{\omega} \leq 1 + \frac{p-1}{p'} \,. \tag{1.78}$$

A korlát pontosságát jellemzi a következő állítás.

1.18. tétel (ütemezési korlát élessége). *A relatív sebességre adott korlát az m, t, $<$ és L paraméterek mindegyikének változására nézve (külön-külön is) aszimptotikusan éles.*

1.8.3. Párhuzamos feldolgozás átfedéses memóriával

Népszerű formában fogalmazzuk meg az átfedéses memóriájú számítógépek működését modellező párhuzamos algoritmust. A T_0, T_1, ..., T_r törpék n különböző fajtájú gombócot főznek. Ezeket az egyszerűség kedvéért az 1, 2, ..., n számokkal jelöljük. Mindegyik törpe végtelen gombócsorozatot állít

elő.

Ezeket a gombócokat O_f óriások eszik meg – ahol az f paraméter azt mutatja, hogy az adott óriás az egyes gombócfajtákból legfeljebb hányat ehet meg egy falatban.

Az O_f óriás a következőképpen eszik. Első falatához a T_0 törpe sorozatának elejéről a lehető legtöbb gombócot kiválasztja (de egy-egy fajtából legfeljebb f darabot). Ezt a falatot még a T_1, ..., T_r törpék sorozatának elejéről kiegészíti – az f korlátot betartva.

A további falatokat hasonlóan állítja össze.

Legyen $h_i(f)$ $(i = 1, 2, \ldots)$ az O_f óriás i-edik harapásában lévő gombócok száma. Ekkor az O_f óriás S_f ***gombócevési sebességét*** az

$$S_f = \liminf_{t\to\infty} \frac{\sum_{i=1}^{t} h_i(f)}{t} \tag{1.79}$$

határértékkel definiáljuk.

Könnyen belátható, hogy ha $1 \le f \le g$, akkor a gombócevési sebességekre fennáll

$$f \le S_f \le fn, \quad g \le S_g \le gn\,, \tag{1.80}$$

a két óriás relatív gombócevési sebességére pedig

$$\frac{f}{gn} \le \frac{S_f}{S_g} \le \frac{fn}{g}\,. \tag{1.81}$$

Most megmutatjuk, hogy a *kis óriás* gombócevési sebessége akárhányszor nagyobb lehet, mint a *nagy óriásé*.

1.19. tétel. *Ha $r \ge 1$, $n \ge 3$, $g > f \ge 1$, akkor léteznek olyan gombócsorozatok, amelyekre egyrészt*

$$\frac{S_g}{S_f} = \frac{g}{fn}\,, \tag{1.82}$$

másrészt

$$\frac{S_g}{S_f} = \frac{gn}{f}\,. \tag{1.83}$$

Bizonyítás. A természetes korlátokat megadó **??** egyenlőtlenségben szereplő alsó korlát élességének belátásához tekintsük az

$$1^f 2^{2f+1} 1^* \tag{1.84}$$

és

$$1^{f+1}(2\ 3\ \ldots\ n)^* \tag{1.85}$$

sorozatokat.

A felső korlát élességét a

$$1\ 2^{2f+1}\ 1^* \tag{1.86}$$

és

$$1^{f-1}\ 3^{2f}\ 1^f\ (2\ 3\ \ldots\ n)^* \tag{1.87}$$

sorozatok „megevésével” láthatjuk be. ■

Az alsó korlát élessége azt fejezi ki, hogy a kis óriás ehet sokkal kevesebbet – ami természetes. Az n növelésével tetszőlegesen naggyá tehető felső korlát élessége azonban *erős anomália*.

1.8.4. Párhuzamos korlátozás és szétválasztás

A *korlátozás és szétválasztás* gyakran alkalmazott optimalizációs módszer.

A módszer alkalmazása során olyan $x = (x_1, x_2, \ldots, x_n)$ vektort keresünk, amely minimalizál egy $f(x)$ függvényt – figyelembe véve korlátozások K halmazát. A korlátozások lehetnek explicitek vagy implicitek. Az implicit korlátozások az x_i értékek kapcsolatát jellemzik, például

$$\sum_{i=1}^{n} a_i x_i \le b, \tag{1.88}$$

$$a_1 x_1^2 - a_2 x_1 x_2 + a_3 x_3^4 = 6\ . \tag{1.89}$$

Az explicit korlátozások az x_i értékekre adnak korlátokat, például

$$x_i \in \{0, 1\} \, , x_i \geq 0 \, . \tag{1.90}$$

Az explicit korlátozásokat kielégítő vektorok alkotják a *megoldási teret*. Ezek a vektorok rendszerint egy fát alkotnak, melyeket ***megoldási fának*** nevezünk. A fa gyökerétől bizonyos levelekhez vezető utak a megoldási tér egy elemét határozzák meg. Az ilyen csúcsokat ***megengedett megoldás csúcsnak*** nevezzük. Egy ilyen csúcs ***költsége*** az f függvény értéke az adott csúcsban. Az optimalizálás célja a ***minimális költségű*** csúcs meghatározása.

Az állapotfa minden N csúcsához hozzárendeljük az

$f_{min}(N) = \min\{f(Q)|Q$ megengedett megoldás az N csúcshoz tartozó részfában$\}$ (ha nincs ilyen Q, akkor $f_{min}(N) = \infty$). értéket.

A továbbiakban az LCBB (**L**east-**C**ost **B**ranch-and-**B**ound) korlátozó-szétválasztó módszerrel foglalkozunk, melyben a következő tulajdonságokkal rendelkező $g()$ heurisztikus függvényt alkalmazzuk:

1. $g(N) \leq f_{min}(N)$ az állapottér minden N csúcsára.
2. $g(N) = f(N)$ a válaszcsúcsokra.
3. $g(N) = \infty$ a megengedett megoldásokra.
4. $g(N) \geq g(P)$, ha N a P csúcs gyereke.

$g(\,)$-t ***korlátozó függvénynek*** nevezzük. LCBB az állapottérhez tartozó csúcsokat állít elő – $g(\,)$ felhasználásával. Az olyan előállított csúcsot, amelynek gyerekeit még nem állítottuk elő, és megengedett megoldásokhoz vezethet, ***élő csúcsnak*** nevezzük. Az élő csúcsok listáját karbantartjuk – rendszerint kupacként. LCBB minden iterációs lépésben kiválaszt egy N élő csúcsot, amelynek $g(\,)$ értéke minimális. Ezt a csúcsot aktuális ***E-csúcsnak*** nevezzük. Ha N válasz csúcs, akkor minimális költségű válasz csúcs. Ha N nem válasz csúcs, akkor előállítjuk a gyerekeit. Azokat a gyerekeket, amelyek nem vezethetnek minimális költségű válaszhoz, eldobjuk (ezeket a csúcsokat bizonyos heurisztikával választjuk ki). A megmaradó gyerekeket hozzáadjuk az

élő csúcsokhoz.

Az LCBB módszer többféleképpen párhuzamosítható. Az egyik lehetőség, hogy minden iterációs lépésben több E-csúcs is kiterjeszthető.

Ha a processzorok száma p, $q = \min\{p, \text{élő csúcsok száma}\}$ csúcsot választunk, mint E-csúcsot (azt a q élő csúcsot, melyekre a legkisebb a $g(\ )$ függvény értéke). Legyen g_{min} ezen csúcsok $g(\ )$ értékeinek minimuma. Ha ezen E-csúcsok bármelyike válaszcsúcs, és a $g(\ )$-értéke g_{min}, akkor az a csúcs minimális költségű válaszcsúcs. Egyébként mind a q csúcsot kiterjesztjük (minden processzor egy csúcsot terjeszt ki), és gyerekeit hozzáadjuk az élő csúcsok listájához. q darab E-csúcs minden ilyen kiterjesztése a párhuzamos LCBB egy iterációja. Adott I-re és g-re jelöljük $I(p)$-vel a p processzor esetében szükséges iterációk számát. Ekkor természetesnek látszanak $I(p)$ következő tulajdonságai:

1. ha $p_1 < p_2$, akkor $I(p_1) \geq I(p_2)$;
2. $\frac{I(p_1)}{I(p_2)} \leq \frac{p_2}{p_1}$.

Az első tulajdonság szerint a processzorok számának növelésekor nem nőhet az iterációk száma.

A második tulajdonság szerint a teljesítmény nem lehet nagyobb, mint a felhasznált erőforrások mennyisége.

Lai és Sahni 1984-ben megmutatták, hogy egyik tulajdonság sincs biztosítva – még akkor sem, ha $g(\ )$ eleget tesz a (1)–(4) korlátozásoknak.

1.8.5. Az anomália elkerülése

Az anomáliát általában igyekeznek elkerülni.

A lapcserélésnél például az elkerülés elégséges feltétele az, ha a cserélési algoritmus rendelkezik a ***verem tulajdonsággal:*** ha ugyanazt a hivatkozási sorozatot m és $m + 1$ méretű memóriájú gépen futtatjuk, akkor minden hivatkozás után igaz az, hogy a nagyobb memória mindazokat a lapokat tartalmazza, amelyeket a kisebb tartalmaz.

A vizsgált ütemezési feladatnál elegendő az, ha nem követeljük meg az ütemező algoritmustól a lista alkalmazását.

Gyakorlatok

1.8-1. Az A és B párhuzamos algoritmusok megoldják a kiválasztási feladatot. Az A algoritmus $n^{0.5}$ processzort használ és a futási ideje $\Theta(n^{0.5})$. A B algoritmus n processzort használ és a futási ideje $\Theta(\lg n)$. Határozzuk meg az elvégzett munkát, a gyorsítást és a hatékonyságot mindkét algoritmusra. Munkahatékonyak-e ezek az algoritmusok?

1.8-2. Elemezzük a következő két állítást.

a. Az $\mathcal{A}$ algoritmus futási ideje legalább $O(n^2)$.

b. Mivel az $\mathcal{A}$ algoritmus futási ideje $O(n^2)$, a $\mathcal{B}$ algoritmus futási ideje pedig $O(n \lg n)$, ezért a $\mathcal{B}$ algoritmus a hatékonyabb.

1.8-3. Terjesszük ki a váltási hely definícióját nem egész v értékekre és párhuzamos algoritmusokra.

1.8-4. Becsüljük meg annak valószínűségét, hogy egy szabályos pénzérmét ezerszer feldobva legalább hatszázszor dobunk fejet. *Útmutatás.* A $Pr[X = 600]$ valószínűséget a Stirling-képlettel, a $Pr[X = 601], \ldots, Pr[X = 1000]$ valószínűségeket pedig geometriai sorral becsüljük.

1.8-5. Egészítsük ki a hálózatok adatait tartalmazó 1.2. táblázatot további adatokkal és hálózatokkal.

Feladatok

1-1. Variációk O-ra és Ω-ra

Az Ω és O különböző definíciói ismertek. Az egyikre a $\overset{\infty}{\Omega}$ (olvasd: omega végtelen) jelölést használjuk. Azt mondjuk, hogy $f(n) = \overset{\infty}{\Omega}(g(n))$, ha létezik c pozitív valós állandó úgy, hogy $f(n) \geq cg(n) \geq 0$ teljesül végtelen sok

egész n-re.

a. Egy függvényt aszimptotikusan nemnegatívnak nevezünk, ha minden elég nagy n-re nemnegatív. Mutassuk meg, hogy bármely két aszimptotikusan nemnegatív $f(n)$ és $g(n)$ függvény esetében vagy $f(n) = O(g(n))$, vagy $f(n) = \overset{\infty}{\Omega}(g(n))$, vagy mindkettő teljesül, ha azonban $\overset{\infty}{\Omega}$ helyett Ω-t használunk, akkor nem igaz az állítás.

b. Mik a lehetséges előnyei és hátrányai annak, ha a programok futási idejének jellemzésére $\overset{\infty}{\Omega}$-t használunk Ω helyett?

Néhány szerző a O-t is kissé másképp definiálja; használjuk az O' jelölést erre az alternatív definícióra. Azt mondjuk, hogy $f(n) = O'(g(n))$ akkor és csak akkor, ha $|f(n)| = O(g(n))$.

c. Ismert, hogy $f(n) = \Theta(g(n))$ akkor és csak akkor teljesül, ha $f(n) = \Omega(g(n))$ és $f(n) = O(g(n))$. Mit mondhatunk, ha Ω helyett $\overset{\infty}{\Omega}$ szerepel?

Szokásos a $\tilde{O}$-n (olvasd: gyenge ordó) szimbólumot a logaritmikus tényezők elhanyagolásával kapott O-jelölésre használni. Azt mondjuk, hogy $\tilde{O}(g(n)) = f(n)$, ha léteznek olyan c pozitív valós, k és n_0 pozitív egész állandók úgy, hogy, ha $n \geq n_0$, akkor $0 \leq f(n) \leq cg(n)\lg^k(n)$.

d. Definiáljuk hasonló módon $\tilde{\Omega}$-t és $\tilde{\Theta}$-t. Bizonyítsuk be a ***c.*** részben kimondott állítás ennek megfelelő változatát.

1-2. A gyorsítás korlátai

a. Bizonyítsuk be Amdahl és Gustafson törvényét.

b. Magyarázzuk meg, hogy a két törvény közötti ellentmondás látszólagos.

c. A gyakorlatban milyen korlátai vannak a gyorsításnak?

1-3. Hanoi tornyai általánosan

Általánosítsuk a Hanoi tornyaira vonatkozóan kapott eredményt.

a. Mit mondhatunk a lépésszámról akkor, ha négy rudat használhatunk?

b. Mit mondhatunk a lépésszámról akkor, ha $2 + k$ ($k \geq 1$) rudat használhatunk?

c. Mit mondhatunk a lépésszámról akkor, ha korlátozzuk a rudak közötti mozgatás lehetőségeit, például az A rúdról a B rúdra és a B rúdról az A rúdra nem szabad korongot áthelyezni?

d. Mennyire csökkenthető a lépésszám konstans számú rúd esetén?

e. Mennyi rúd elegendő a lépésszám $O(n)$-re csökkentéséhez?

f. Mit mondhatunk a lépésszámról akkor, ha s szerzetes párhuzamosan dolgozhat? Feltételezzük, hogy minden lépésben minden rúdról legfeljebb egy korongot szabad elvenni és minden rúdra legfeljebb egy korongot szabad ráhelyezni, továbbá minden szerzetes legfeljebb egy korongot mozgathat?

g. Mit mondhatunk a lépésszámról akkor, ha az s szerzetes párhuzamos munkáját paraméteresen értelmezzük: az m-párhuzamos olvasás jelentse azt, hogy a szerzetesek egy lépésben rudanként legfeljebb az m legfelső koronghoz férnek hozzá; az m-párhuzamos írás pedig jelentse azt, hogy minden korongra külön legfeljebb m korongot helyezhetnek egyidejűleg.

1-4. Anomália

Tervezzünk egy algoritmust (és valósítsuk is meg), amely azt bizonyítja, hogy egy adott feladatot $q > p$ processzoron megoldani tovább tart, mint $p > 1$ processzoron.

1-5. Párhuzamossággal a hírnévért és dicsőségért

1997-ben Andrew Beale dallasi bankár 50 ezer dollár díjat tűzött ki annak, aki bizonyítja vagy cáfolja sejtését. Eszerint ha

$$a^q + b^p = c^r, \tag{1.91}$$

akkor az a, b és c számoknak van egynél nagyobb közös osztója (az egyenletben mind a hat betű egész számot jelent és a kitevők értéke legalább három).

Hogyan használhatók fel a párhuzamos algoritmusok a díj megszerzésére?

1-6. Rangsorolás

Tekintsük az összehasonlítás alapú rendezés következő általánosítását. n elemet – például n sportolót – páronként összehasonlítunk, és a *győztesnek* 1 pontot, a vesztesnek pedig 0 pontot adunk. Legyen p_i az i-edik játékos *pontszáma* (győzelmeinek száma).

a. Tervezzünk párhuzamos algoritmust, amely adott $\mathbf{q} = q_1, q_2, \ldots, q_n$ sorozatról eldönti, hogy lehet-e az előbb leírt *verseny* pontsorozata. *Útmutatás.* Használjuk fel Landau tételét, amely szerint $\mathbf{q}$ akkor és csak akkor lehet pontsorozat, ha a következő két feltétel mindegyike teljesül:

$$\sum_{i=1}^{k} q_i \geq \binom{k}{2} \quad (\text{ha } 1 \leq k \leq n) \tag{1.92}$$

és

$$\sum_{i=1}^{n} q_i = \binom{n}{2}. \tag{1.93}$$

b. Oldjuk meg a feladatot akkor, ha minden összehasonlításnál $k \geq 1$ pontot osztunk ki az összehasonlított sportolók között, és a k pont minden lehetséges módon feloszthető (azzal a megszorítással, hogy mindkét játékosra a kapott pontok száma nemnegatív egész).

c. Oldjuk meg a feladatot akkor, ha minden összehasonlításnál k ($1 \leq a \leq k \leq b$) pontot osztunk ki az összehasonlított sportolók között, – a k pont minden lehetséges módon feloszthető (azzal a megszorítással, hogy mindkét játékosra a kapott pontok száma nemnegatív egész), az a és b számok pozitív egészek.

d. Elemezzük a feladatnak azt az általánosítását, amelyben az összehasonlításoknak csak egy részét végeztük el. Lényeges-e annak ismerete, hogy mely összehasonlításokat végeztük el?

Névmutató

A névmutatóban a szerzők utóneveit csak akkor rövidítjük, ha nem ismerjük a teljes nevet.

Tárgymutató

Ez a tárgymutató a következő szempontok szerint készült.

Először a matematikai jelöléseket soroljuk fel (latin ábécé, majd a görög ábécé szerinti sorrendben), azután a tárgyszavakat.

A számokat és görög betűket tartalmazó tárgyszavakat kiejtésük szerint rendezzük: például az „1-értékű”-t „egyértékékű”-ként, a λ-t „lambda”-ként. A jelölést tartalmazó tárgyszavakat elemeik szerint rendezzük: például a „k-megegyezés”-t „k megegyezés”-ként.

A különböző típusú objektumokat lehetőség szerint tipográfiailag is megkülönböztettük. A matematikai jelöléseket és a programokban használt változók neveit dőlt betűk emelik ki, mint például $\Omega(n \lg n)$ vagy *rang*[*lépés*]. Az algoritmusok neveit kis kapitális betűkkel írtuk, mint például KIVÁLASZT. Az algoritmusok kódjában a programozási alapszavakat félkövéren szedtük, mint például **if, then, for, in parallel for**.

Az algoritmusok nevében kiskötőjelet, a változók nevében pedig alsó kötőjelet használunk, mint például PÁRHUZAMOSAN-OLVAS és *bal_szomsz*. Az egyes fogalmak meghatározásának helyére a tárgymutató dőlt oldalszámmal utal.

Elsősorban az algoritmusokat tárgyaló tankönyvek matematikai jelöléseit alkalmaztuk. Az oldalszámok felsorolásánál nem törekedtünk teljességre.

Megoldások

1.8-1. megoldása. Az A algoritmus munkája $\Theta(n)$, a B algoritmusé $\Theta(n \lg n)$. Mivel a kiválasztási feladatot megoldó leggyorsabb soros algoritmus futási ideje $O(n)$, így az A algoritmusnak a leggyorsabb soros algoritmusra vonatkozó gyorsítása $\sqrt{(n)}$, a B algoritmusé pedig $\Theta(n/\lg n)$. Az A algoritmus hatékonysága $\Theta(1)$, a B algoritmusé $\Theta(1/\lg n)$. Eszerint az A algoritmus munkahatékony, a B algoritmus viszont nem az.

1.8-2. megoldása. Az első állításban a „legalább” szó és az O-jelölés egymás mellett értelmetlen.

A második állítás hibás: a B algoritmus futási idejének alacsonyabb felső korlátjából nem következik, hogy gyorsabb: előfordulhat például, hogy az A algoritmus futási ideje lineáris, míg a B algoritmus esetén a felső korlát a pontos nagyságrendet adja meg.

Tartalomjegyzék