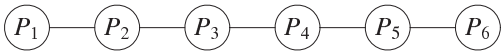


3.1. ábra. 6 processzoros lánc.

3. Rácsok

Ebben a fejezetben rácsokat (láncot, négyzetet és kockát) alkalmazunk számítási modellként.

3.1. Számítási modellek

A k dimenziós ($k \geq 1$) dimenziós 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 egy 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.

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

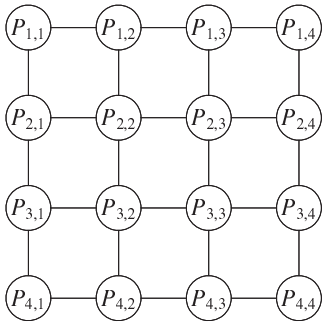
Egy 6 processzoros lánc látható a 3.1. ábrán.

Egy lánc processzorai P_1, P_2, $\ldots$, P_p. Ezek a következőképpen vannak összekötve. P_1 és P_p kivételével mindegyik processzor össze van kötve a nála eggyel nagyobb (jobb szomszéd), illetve eggyel kisebb (bal szomszéd) indexűvel, míg a két szélső processzornak csak egy szomszédja van, P_2, illetve P_{p-1}. Az összeköttetés kétirányú.

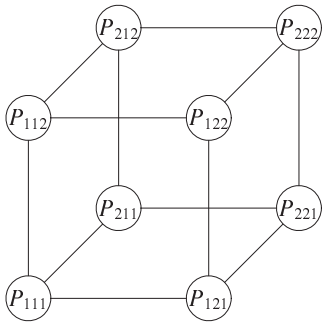
Ha $k = 2$, akkor téglalap alakú rácsot kapunk. Ha most $m_1 = m_2 = \sqrt{p} = a$, akkor $a \times a$ méretű *négyzetet* kapunk.

Egy 4×4 méretű négyzet látható a 3.2. ábrán.

Egy $a \times a$ méretű négyzet tartalmaz részgráfként számos a processzoros láncot. A rács



3.2. ábra. 4×4 méretű négyzet.



3.3. ábra. $2 \times 2 \times 2$ méretű kocka.

algoritmus egyes lépései gyakran tekinthetők láncokon végzett műveleteknek. A processzorok közötti kommunikáció bármely rögzített összekötöttségű gépben kommunikációs láncok segítségével történik. Ha két olyan processzor akar kommunikálni egymással, amik egy éllel össze vannak kötve, akkor azt egyetlen lépésben elvégezhetik. Ha nincs közöttük él, akkor az őket összekötő utak valamelyikén történhet meg a kommunikáció, tehát a szükséges lépésszám (legalábbis rövid üzenetek esetén) függ az út hosszától. Feltesszük, hogy egy processzor egyetlen lépésben képes végrehajtani egy számítást és/vagy kommunikálni akár mind a négy szomszédjával.

Egy rácsban azok a processzorok, amelyeknek első (második) koordinátája megegyezik, egy sort (oszlopot) alkotnak. Például egy $a \times a$ méretű négyzetben az i-edik sor a $P_{i,1}$, $P_{i,2}$, $\ldots$, $P_{i,a}$ processzorokból áll. Mindegyik sor vagy oszlop egy a processzoros lánc. Egy rács algoritmus gyakran áll olyan lépésekből, melyeket csak bizonyos sorokban vagy oszlopokban lévő processzorok végeznek el.

Ha $k = 3$, akkor tégla alakú rácsot kapunk. Az $m_1 = m_2 = m_3 = \sqrt[3]{p}$ speciális esetben ***kockáról*** és a kocka méretére az $n \times n \times n$ jelölést alkalmazzuk.

A 3.3. ábra egy $2 \times 2 \times 2$ méretű kockát ábrázol.

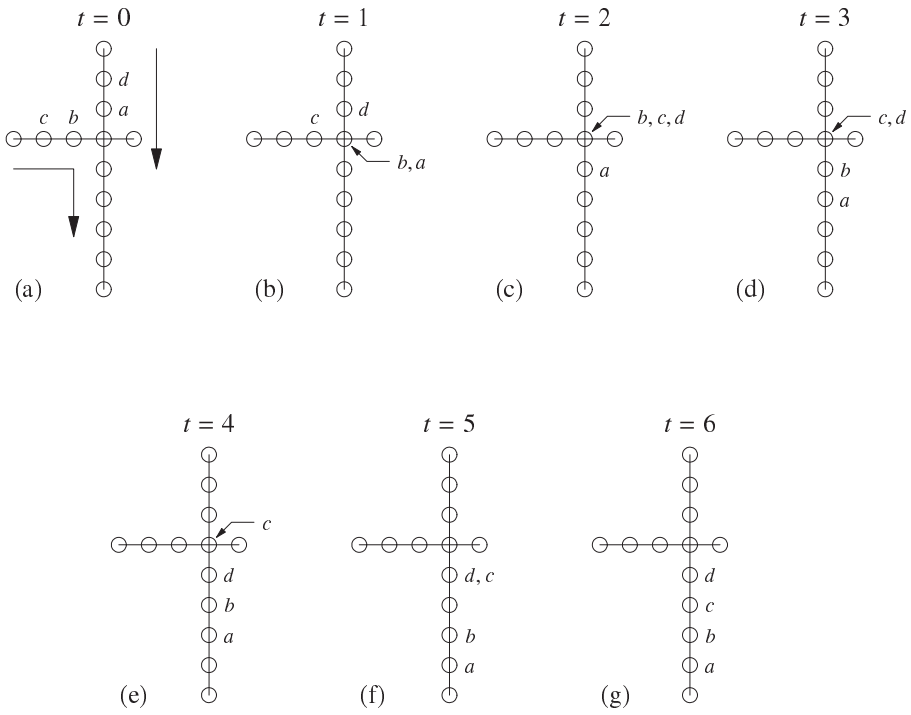
3.2. Csomagirányítás

A processzorok közötti kommunikáció egyetlen lépése egy rögzített szerkezetű hálózatban a következő – ***csomagirányítási problémának*** nevezett – feladatként fogható fel. A hálózatban minden processzornak van egy adatcsomagja, amit egy másik processzornak akar elküldeni. A feladat a csomagok eljuttatása a céljukhoz a lehető leggyorsabban úgy, hogy egy lépésben egy kommunikációs csatornán egy irányban egyszerre csak egy csomag utazhat. Az utóbbi feltételre a csatorna sávszélességének korlátozottsága miatt van szükség. Könnyen előfordulhat, hogy egy adott lépésben kettő vagy több csomag érkezik egy processzorhoz, és mindegyik ugyanazon a csatornán szeretne továbbhaladni. Ilyen esetben természetesen csak egy csomag utazhat a következő lépésben, a többiek pedig a processzornál egy várakozási sorba kerülnek a későbbi továbbküldés miatt. Egy ***elsőbbségi szabály*** alapján döntjük el, hogy melyik csomagot küldjük el ilyen esetekben. Ilyen elsőbbségi szabályok például az FDF (**F**arthest **D**estination **F**irst) FOF (**F**arthest **O**rigin) **F**irst) , FIFO, RAN (véletlenül választunk)

A csomagirányítási feladat egy speciális esete a ***parciális permutációs csomagirányítás*** (**P**artial **P**ermutation **R**outing). A PPR-ben minden processzornak legfeljebb egy elküldendő csomagja van, és egy adott processzorhoz legfeljebb egy csomagot kell küldenünk. A PPR-t egy ERCW PRAM gépen egy párhuzamos írással megoldhatjuk. De egy általános rögzített összekötöttségű hálózatban a probléma gyakran igen bonyolult. Tipikusan a bemenet egy bizonyos sorrendben kerül a processzorokhoz, és a kimenetnek úgyszintén egy előre megadott sorrendben kell megjelennie. Csupán egy ilyen sorrend átrendezés néha több PPR-t igényelhet. Ezért bármely nem triviális rögzített összekötöttségű hálózati algoritmus tervezéséhez szükség van PPR-ekre. Ez az egyik lényeges különbség a hálózati és a PRAM algoritmusok között.

Egy csomagirányítási algoritmus legfontosabb jellemzői a *futási ideje* amíg az utolsó csomag is eléri a célját, és a ***várakozási sor hossza*** ami az egy processzornál maximálisan várakozó csomagok száma. A sor lehetséges hosszát alulról korlátozza az egy processzortól induló, valamint az egy processzorhoz érkező csomagok maximális száma. Feltételezzük, hogy a csomag nemcsak a küldendő információt tartalmazza, hanem a küldő és a célprocesszor azonosítóját is. Egy csomagirányítási algoritmus bemenő adatai a csomagok indulási helye és célja, valamint a használni kívánt elsőbbségi szabály. Egy csomag utazásának lépésszáma az indulás és a cél között megtett út és a sorokban eltöltött várakozás hosszától függ.

3.1. példa. *4 csomag irányítása.* Tekintsük az a, b, c, d csomagokat a 3.4. ábra (a) részén. A rendeltetési helyüket az ábra (g) része mutatja. Használjuk a FIFO elsőbbségi szabályt úgy, hogy holtverseny esetén önkényesen döntsünk valamelyik csomag a javára. A csomagok a lehető legrövidebb úton közlekedjenek. A $t = 1$ sorszámú lépésben minden csomag eggyel közelebb kerül a céljához. Ennek eredményeként az a és a b ugyanazon a ponton vannak, tehát a $t = 2$ sorszámú lépésben valamelyiket várakoztatni kell. Mivel mindketten egyszerre érkeztek, ezért ez egy holtverseny. Önkényesen döntünk, ezért nyerjen mondjuk a. Ugyanebben a lépésben még c és d is csatlakozik b-hez. A következő lépésben b fog továbbmenni, mivel ő előbb érkezett, és ezért elsőbbsége van a többihez képest. A negyedik lépésben c és d holtversenyben küzd a továbbhaladásért. Legyen d a győztes, aki tovább



3.4. ábra. Példa csomagirányításra.

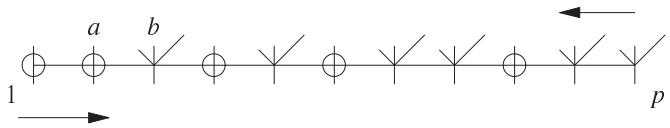
haladhat. További két lépésben c is eljut a céljába. Ekkorra minden csomag megérkezett.

Az a csomagnak öt élen kellett áthaladnia, és közben sehol sem várt – ezért öt időegység alatt ért célba. A c csomagnak négy élen kellett átutaznia, és két alkalommal várt, ezért hat időegység alatt ért célba – ez adja az algoritmus futási idejét. Vajon más elsőbbségi szabályt használva csökkenhet a lépésszám? Játsszuk végig az előző példát az FDF szabállyal. Ekkor a futási idő öt időegység lesz.

3.2.1. Csomagirányítás láncon

Egy láncon, mivel az összekötöttség kétirányú, a processzor egyszerre küldhet és fogadhat a szomszédjaitól üzeneteket. Ennek következtében két ellentétes irányú csomaglánc nem zavarja egymást. Ebben a részben megmutatjuk, hogy a PPR láncon megoldható legfeljebb $p-1$ lépésben. Vegyük észre, hogy a legrosszabb esetben legalább $p-1$ lépés kell, hiszen ez a lehető legnagyobb távolság két processzor között. Láncon PPR-en kívül vizsgálunk még néhány általánosabb csomagirányítási feladatot is.

3.2. példa. *Független csomagáramok.* A 3.5. ábrán a balról jobbra haladó csomagokat körökkel, a jobbról balra haladó csomagokat pedig pipával jelöltük. feltesszük, hogy a balra haladó csoma-



3.5. ábra. A jobbra és balra haladó csomagáramok függetlenek.

gok célja P_1, a jobbra haladó csomagoké pedig P_p. Például az a és b csomagoknak egyidejűleg van szüksége ugyanarra az élre, amikor az első lépést megteszik (ellenkező irányban). Mivel az élek kétirányúak, nincs verseny, a csomagok használhatják egyszerre az élet. Mivel a P_1 processzortól induló csomagnak $p-1$ élen kell keresztül haladnia, ezért a célba éréshez legalább $p-1$ lépésre van szüksége.

Tegyük fel, hogy egy p processzorból álló láncban minden processzor legfeljebb egy üzenetet küld, az üzenetek célja tetszőleges lehet. Továbbítsuk a csomagokat a célprocesszorokhoz. Ezt a feladatot ***egy kezdőcsomagos feladatnak*** nevezzük.

3.1. lemma (egy kezdőcsomagos feladat). *Az egy kezdőcsomagos feladat egy p processzoros láncon megoldható legfeljebb $p-1$ lépésben.*

Bizonyítás. Minden q üzenetet küldhetünk a kezdő és a végpont közötti legrövidebb úton. Tekintsük csak azokat az üzeneteket, amelyek balról jobbra utaznak, hiszen a jobbról balra utazókat teljesen hasonlóan függetlenül vizsgálhatjuk. Ha q a P_i processzortól indul és P_j felé tart, akkor $j-i$ lépésben éri azt el, hiszen sosem kell várakoznia egy másik üzenetre. A leghosszabb ilyen út 1-től p-ig vezet, ezért $p-1$ felső korlát a lépésszámra. Az egy csúcsba küldött csomagok maximális száma jellemzi az algoritmus várakozási sorának hosszát. ■

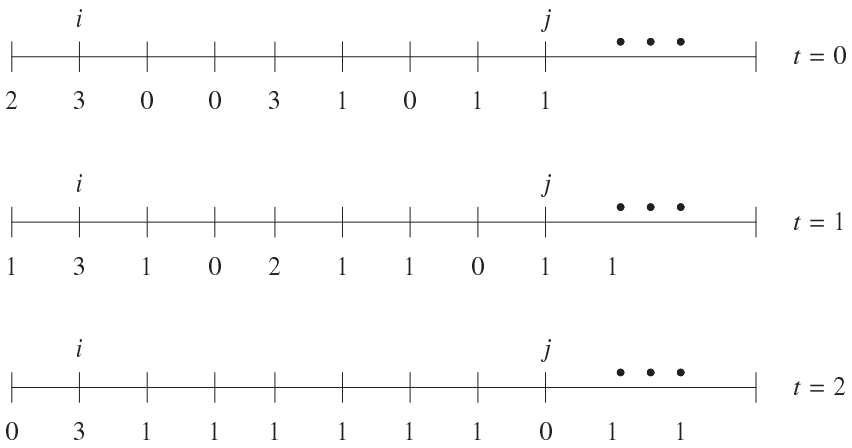
Adott egy p processzorból álló lánc. A P_i processzor k_i $(0 \le k_i \le p)$ üzenetet akar küldeni és

$$\sum_{i=1}^{p} k_i = p. \tag{3.1}$$

Nincs két olyan üzenet, amelyeket azonos processzorhoz kellene küldeni. Továbbítsuk a csomagokat a célprocesszorokhoz. Ezt a feladatot ***egy célcsomagos feladatnak*** nevezzük.

3.2. lemma (egy célcsomagos feladat). *Ha az FDF elsőbbségi szabályt használjuk, akkor az egy célcsomagos feladat egy p processzoros láncon megoldható legfeljebb $p-1$ lépés alatt.*

Bizonyítás. Tegyük fel, hogy q üzenet P_i-ből P_j-be megy. Az általánosság megszorítása nélkül feltehetjük, hogy a csomag balról jobbra halad. A jobbról balra haladó csomagoktól eltekinthetünk hasonló okok miatt, mint az előző lemmánál. Minden csomag a lehető



3.6. ábra. Szabad sorozat.

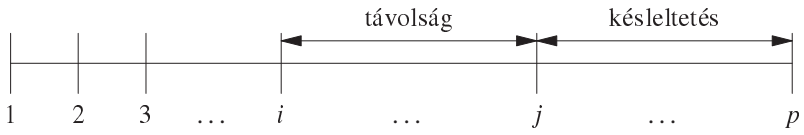
legrövidebb úton halad, ami a mi esetünkben $j-i$ lépést jelent. Ne feledkezzünk meg azonban azokról a csomagokról, amelyek j-nél nagyobb indexű processzorokhoz utaznak, mivel ezek (és csak ezek) megvárakoztathatják q-t. Ezen csomagok száma legfeljebb $p-j$. Jelölje $k_1, k_2, \ldots, k_{j-1}$ a kezdeti állapotban rendre a $P_1,\ P_2, \ldots, P_{j-1}$ processzortól induló ilyen csomagok számát. Jelölje m minden lépés után azt az indexet, melyre $k_{m-1} > 1$, és ha $m \leq s \leq j$, akkor $k_s \leq 1$. Nevezzük a $k_m, k_{m+1}, \ldots, k_{j-1}$ sorozatot ***szabad sorozatnak***.

Vegyük észre, hogy az elkövetkezőkben a szabad sorozatban egyik csomag sem fog várakozni. Ezenfelül minden egyes lépésben legalább egy csomag csatlakozik a szabad sorozathoz.

A 3.6. ábra egy szabad sorozatot mutat. Az ábrán a számok a megfelelő processzoroknál (melyeknek csak az indexe szerepel az ábrán) lévő csomagok számát mutatják. Például a nulladik lépésben ($t = 0$) lépésben P_i-nél 3 csomag van és 1, 0, 1, 1 a szabad sorozat. A következő lépésben ($t = 1$) egy, majd az azt követő lépésben ($t = 2$) 4 új csomag csatlakozik a szabad sorozatban lévő csomagokhoz.

Így legfeljebb $p - j$ lépés után minden olyan csomag csatlakozott a szabad sorozathoz, amely miatt q várakozásra kényszerülhet. Ezt a helyzetet mutatja a 3.7. ábra: a q csomagnak a legfeljebb $p - j$ lépésnyi várakozáson felül $j - i$ lépést kell tennie, ezért legfeljebb $p - i$ lépés után célba ér.

A jobbról balra haladó csomagok esetében hasonlóan belátható, hogy legfeljebb $i - 1$ lépésben megérkeznek a rendeltetési helyükre. ■



3.7. ábra. A 3.1. lemma bizonyításának szemléltetése

Most definiáljuk az ***általános csomagirányítási*** feladatot. Tételezzük fel, hogy egy p processzoros láncon több csomag származhat egy processzortól és több csomagot küldhetünk egy processzorhoz. Továbbá a P_1, $P_2, \ldots$, P_j ($j = 1, 2, \ldots, p$) processzoroktól összesen induló csomagok száma nem több, mint $j + f(p)$, p valamely rögzített $f(p)$ függvényére. Továbbítsuk a csomagokat a célprocesszorokhoz.

3.3. lemma. (*Általános csomagirányítási feladat.*) *Ha a* FOF *elsőbbségi szabályt használjuk, akkor az általános csomagirányítási probléma megoldható $p + f(p)$ lépésben.*

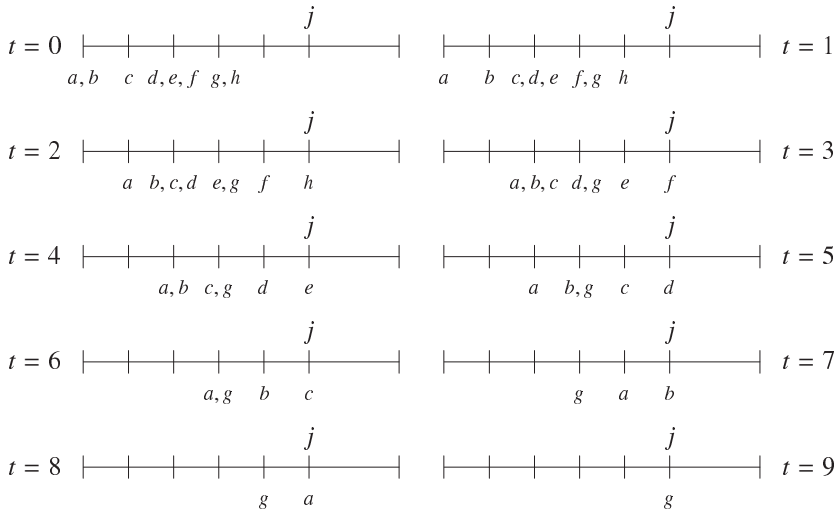
Bizonyítás. Legyen q egy P_i-ből P_j-be utazó üzenet, ahol $i < j$ (a $j > i$ eset hasonlóan kezelhető). A q csomag legfeljebb $i + f(p)$ csomag miatt várakozhat, hiszen csak ennyi csomag érkezhet távolabbról és lehet ezért nagyobb prioritása. Ha q minden egyes ilyen csomag miatt legfeljebb egyszer kényszerül várakozni, akkor az összes várakozási lépésszáma legfeljebb $i + f(p)$. Ha azonban valamely r csomag mondjuk kétszer várakoztatja meg q-t, ez azt jelenti, hogy r várakozásra kényszerült egy másik, nagyobb prioritású csomag miatt, ami sosem fogja q-t megvárakoztatni. Emiatt q várakozása legfeljebb $i + f(p)$ lehet. Mivel q $-$*nak* már csak $j - i$ lépést kell megtennie, ezért legfeljebb $j + f(p)$ lépés kell q helyre szállításához. Az összes csomagra ez a lépésszám legfeljebb $p + f(p)$. ■

3.3. példa. A 3.3 lemma bizonyításának szemléltetése

A 3.8. ábra mutatja a 3.3. lemma bizonyításának menetét (a processzoroknak itt is csak az indexe szerepel). A példában 8 csomag van: a, b, c, d, e, f, g, h. Vizsgáljuk a g csomagot. Ezt a csomagot a többi 7 csomag késleltetheti. A g csomag a $t = 9$ sorszámú lépésben éri el célját. Megtett útjának hossza 2, késleltetése 7. Az ábra nem mutatja azokat a csomagokat, amelyek a P_j csúcsban keresztezték egymást.

3.2.2. Egy mohó algoritmus a PPR megoldására rácson

Egy PPR probléma megoldásához egy $\sqrt{p} \times \sqrt{p} = a \times a$ méretű rácson egy üzenetnek az $(1, 1)$ csúcsból az (a, a) csúcsba szállításához legalább $N(n, a^2, \mathcal{P}) \geq 2(a - 1)$ lépésre van szükség. Ezért $2(a - 1)$ egy alsó korlát bármely csomagirányító algoritmus lépésszámának



3.8. ábra. A 3.3. lemma bizonyítását illusztráló példa

legrosszabb esetét vizsgálva: $W(n, a^2, \mathcal{A}) \geq 2(a-1)$.

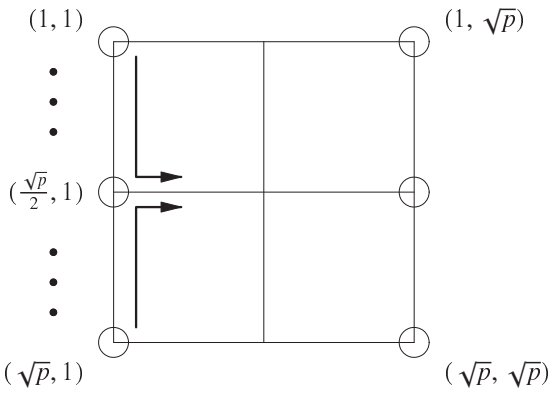
Egy egyszerű PPR algoritmus, amely felhasználja a láncoknál ismertetett csomagirányítási algoritmusokat, a következő. Legyen q egy tetszőleges csomag, amelyet a $P_{i,j}$ processzortól a $P_{k,l}$ indexűnek kell elküldeni. Az csomag az első fázisban a j-edik oszlop mentén a k-adik sorig halad a legrövidebb úton. A második fázisban a k-adik sor mentén a legrövidebb úton az l-edik oszlophoz elérve a csomag megérkezett a rendeltetési helyére. Egy csomag azonnal megkezdheti a második fázist az első befejezése után, nem kell semmi másra várnia.

Az első fázis legfeljebb $a-1$ lépésben elvégezhető, mivel a 3.1. lemma alkalmazható. A második fázis a 3.3. lemmából következően nem tart $a-1$ lépésnél tovább. Így az algoritmus lépésszáma legfeljebb $2(a-1)$, ami az elméleti alsó korlát, azaz az algoritmus abszolút optimális.

De van egy komoly hátránya ennek az algoritmusnak, mégpedig az, hogy a várakozási sor hossza $a/2$. Nézzünk erre egy példát. Legyen a csomagirányítási probléma olyan, hogy az első oszlop minden csomagját a $a/2$-edik sorba kelljen szállítani. Egy ilyen PPR esetén a $(a/2, 1)$ indexű processzor minden lépésben két csomagot kap. Mivel mindkettő ugyanazt a kommunikációs vonalat szeretné használni, ezért az egyik a várakozási sorba kerül. Ez megismétlődik egészen az $a/2$-edik lépésig, mikor is $a/2$ csomag lesz az $(a/2, 1)$ processzor várakozási sorában.

Ezt a helyzetet mutatja a 3.9. ábra.

Az ideális olyan algoritmus lenne, amelynek $O(1)$ méretű várakozási sorra van szüksége (vagy legalább olyan algoritmus, melyre az $f(p)$ függvény lassan nő, mint például a $\lg p$).



3.9. ábra. A mohó algoritmusnak hosszú sorra van szüksége

3.2.3. Egy kis várakozási sort használó véletlenített algoritmus (⋆)

A kétfázisú mohó algoritmus módosítható a véletlenítés segítségével úgy, hogy csak $\overline{O}(\lg p)$ méretű várakozási sort használjon. Az új HÁROM-FÁZISÚ algoritmus három fázisú és a lépésszáma $3a+\overline{o}(a)$. Ebben az algoritmusban a három fázis teljesen elkülönül, azaz egy fázis csak akkor kezdődhet el, ha az előző fázist már minden csomag befejezte. Ez a megszorítás egyszerűbbé teszi az elemzést

Legyen q egy tetszőleges csomag, amelyet $P_{i,j}$-től $P_{k,l}$-hez kell továbbítani.

1. fázis. A q csomag választ egy véletlen $P_{i',j}$ processzort starthelyének oszlopában és a legrövidebb úton eljut oda.

2. fázis. A q csomag az i'. sor mentén továbbítódik a $P_{i',l}$ processzorhoz.

3. fázis. Végül q – az l. oszlop mentén haladva – eléri rendeltetési helyét.

3.4. tétel. *A* HÁROM-FÁZISÚ *algoritmus befejeződik* $3\sqrt{p}+\overline{O}(p^{1/4}\lg p)$ *lépés alatt.*

Bizonyítás. A 3.1. lemma alkalmazásából adódik, hogy az első fázis legfeljebb a lépésben befejeződik, mivel egyetlen csomag sem kényszerül várakozásra.

Tegyük fel, hogy egy csomag a második fázist a $P_{i',j}$ processzornál kezdi. Az általánosság megszorítása nélkül feltehetjük, hogy a csomag jobbra mozog. A $P_{i',j}$ processzortól induló csomagok száma ennél a lépésnél egy binomiális eloszlású valószínűségi változó

$$B\left(a, \frac{1}{a}\right) \tag{3.2}$$

paraméterekkel. Azért ennyi, mert a csomag van a j-edik oszlopban és mindegyik $\frac{1}{a}$ valószínűséggel kerül $P_{i',j}$-hez az első fázis végén. Ezenfelül a második fázis kezdetén a $P_{i',1}$, $P_{i',2}$, …, $P_{i',j}$ processzoroktól induló csomagok száma szintén binomiális eloszlású

$$B\left(ja, \frac{1}{a}\right) \tag{3.3}$$

paraméterekkel. (Felhasználtuk azt a tényt, hogy $B(n_1,x)+B(n_2,x)=B(n_1+n_2,x)$.) Ennek a változónak a várható értéke j. A Csernov-egyenlőtlenséget felhasználva ez a szám $\geq 1-p^{-\alpha-1}$ valószínűséggel legfeljebb $j+3\alpha p^{1/4}\ln p$ minden $\alpha\geq 1$ számra. Így ez az érték $j+(p^{1/4}\lg p)$. A 3.3. lemmát alkalmazva most azt kapjuk, hogy a második fázis $a+\overline{O}(p^{1/4}\lg p)$ lépésben befejeződik.

A harmadik fázis elején bármely oszlopban akármelyik processzortól legfeljebb a csomag indul és legfeljebb egy csomag érkezik. Ezért a 3.3. lemmának megfelelően a harmadik fázis legfeljebb a lépést igényel. ■

3.3. Alapfeladatok

Ebben az alfejezetben négy alapfeladat megoldását mutatjuk be: üzenetszórás, prefixszámítás, adatkoncentráció és ritka rendezés. Egy $a\times a$ méretű rácson mind a négy feladat megoldható $O(a)$ lépésben.

Mivel egy üzenet a négyzetrács egyik sarkából csak $d=2(a-1)$ lépésben juthat el az átellenes sarokba, ez a szám a lépésszám alsó korlátja az előbbi feladatokra, és legrosszabb esetben szükség van az átellenes sarokban lévő processzorok közti kommunikációra. A d szám a négyzetrács ***átmérője.***

A ***k-k irányítási feladat*** a következő. A hálózat bármely processzorától bármely másikhoz legfeljebb k csomagot kell elküldeni.

3.3.1. Üzenetszórás

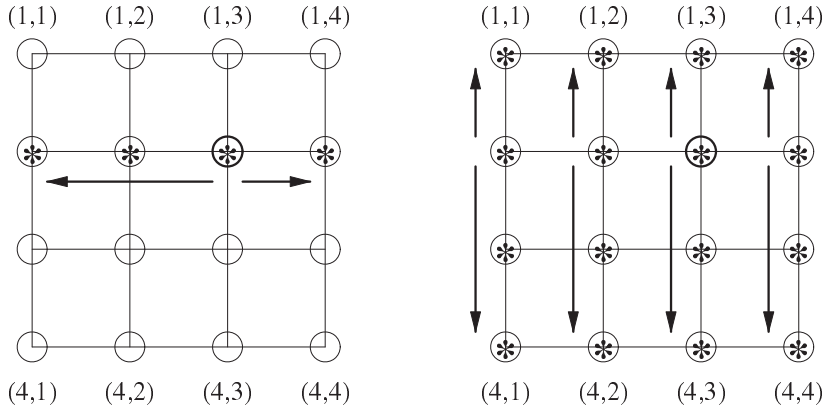
Az ***üzenetszórási feladat*** szerint a hálózat megadott processzorától üzenetet kell eljuttatni megadott célprocesszorokhoz (rendszerint az összes többi processzorhoz).

Legyen $\mathcal{L}$ egy p processzoros lánc és legyen M egy üzenet, amelyet a P_1 processzornak kell elküldenie a többi processzorhoz. A feladat megoldható olyan egyszerűen, hogy P_1 elküldi az üzenetet P_2-nek, az tovább küldi P_3-nak és így tovább. Ekkor az üzenet $p-1$ lépésben eljut a legtávolabbi processzorhoz, P_p-hez. Mivel a lánc átmérője $p-1$, ez a lehető legkisebb lépésszám.

Egy $a\times a$ méretű négyzetrácsban az üzenetküldést két fázisban valósíthatjuk meg. Az első fázisban az üzenetet küldő $P_{i,j}$ processzor eljuttatja üzenetét az i-edik sor minden processzorához. Ezután a második fázisban az i-edik sor minden processzora eljuttatja az üzenetet a vele azonos oszlopban lévő processzorokhoz. Ez összesen legfeljebb $2(a-1)$ lépést vesz igénybe.

Formalizáljuk állításunkat.

3.5. tétel. *Egy p processzoros láncon az üzenetszórás $p-1=O(p)$ lépés alatt elvégezhető. Egy $a\times a$ méretű rácson az üzenetszórás $2(a-1)=O(a)$ lépésben elvégezhető.*



3.10. ábra. Üzenetszórás négyzeten

3.4. példa. *Üzenetszórás rácson.* A 3.10. ábra egy 4×4 méretű négyzeten való üzenetszórás két fázisát mutatja. Az üzenetek a $P_{2,3}$ processzortól indulnak és az első fázisban eljutnak a $P_{2,1}$, $P_{2,2}$ és $P_{2,4}$ processzorokhoz. A második fázisban a második sor processzorai mind felfelé, mind lefelé elküldik az M üzenetet. Az üzenetszórás a legrosszabb esetre jellemző 6 lépés helyett már 4 lépés alatt befejeződik.

3.3.2. Prefixszámítás

A második fejezetben több algoritmust ismertettünk, amelyek különböző párhuzamos gépeken megoldják a prefixszámítási feladatot. Most láncon és négyzeten alkalmazható algoritmusok következnek.

Prefixszámítás láncon

Tegyük fel, hogy az $\mathcal{L} = P_1, P_2, \ldots, P_p$ lánc P_i ($i = 1, 2, \ldots, p$) processzorának saját memóriájában van az x_i elem, és a számítás végén a P_i processzor memóriájában lesz az y_i prefix.

Először nézzünk egy naiv algoritmust.

LÁNC-PREFIX$(\mathcal{L}, X, Y)$ *párhuzamos eljárás*
Számítási modell: lánc
Bemenet: $X = \langle x_1, x_2, \ldots, x_p \rangle$ (az összeadandó elemek)
Kimenet: $Y = \langle y_1, y_2, \ldots, y_p \rangle$ (a prefixek)

```
01  Pi in parallel for i ← 1 to p
02        if i = 1
            then P1 az első lépésben elküldi x1-et P2-nek
```

03 **if** $i = p$
then P_p a p-edik lépésben kap egy elemet (z_{p-1}-et)
P_{p-1}-től, kiszámítja és tárolja $z_{p-1} \oplus x_p$-t
04 **if** $i \neq 1 \wedge i \neq p$
then P_i az i-edik lépésben kap egy elemet (z_{i-1}-t)
P_{i-1}-től, kiszámítja és tárolja $z_i = z_{i-1} \oplus x_i$-t,
majd z_i-t elküldi P_{i+1}-nek

Közvetlenül adódik ennek az algoritmusnak a futási ideje.

3.6. tétel. *A* Lánc-prefix *algoritmus egy láncon* $\Theta(p)$ *idő alatt határozza meg* p *elem prefixeit.*

Mivel egy soros processzorral p elem prefixei $O(p)$ lépésben meghatározhatók, ugyanakkor $W(p, p, \text{Lánc-prefix}) = p^2$, ezért Lánc-prefix nem munkahatékony.

Hasonló algoritmus alkalmazható négyzeten is. Tekintsünk egy $a \times a$ méretű négyzetet. Szükségünk van a processzorok egy ***indexelésére.*** Ilyenek a ***sorfolytonos,*** az ***oszlopfolytonos, kígyószerű sorfolytonos*** és a ***blokkonként kígyószerű sorfolytonos indexelés.***

A blokkonként kígyószerűen sorfolytonos indexelésnél a rácsot megfelelő méretű kisebb blokkokra bontjuk, és a blokkokon belül alkalmazzuk valamelyik indexelést.

A négyzeten való prefixszámítás 3 fázisra osztható, melyekben a számításokat minden fázisban vagy csak soronként, vagy oszloponként végezzük. Az alábbi Négyzeten-prefix algoritmus sorfolytonos indexelést alkalmaz.

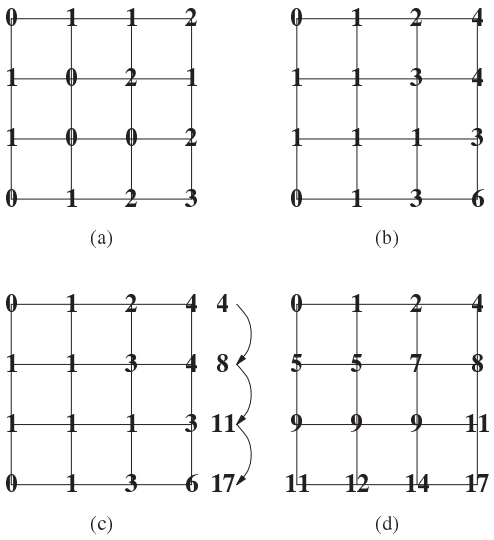
Négyzeten-prefix(X, Y) *párhuzamos eljárás*
Számítási modell: négyzet
Bemenet: $X = (x_1, x_2, \ldots, x_p)$ (az összeadandó elemek)
Kimenet: $Y = (y_1, y_2, \ldots, y_p)$ (a prefixek)

01 P_{i1} **in parallel for** $i \leftarrow 1$**to**a
02 **do** Lánc-prefix$(S_i, X[i], Y[i])$
03 Lánc-prefix$(O_i, Z[i])$
04 $P_{j,a}$ **in parallel for** $j \leftarrow 1$**to**$a - 1$
05 küldje el a kiszámolt prefixet $P_{j+1,a}$-nak
06 S_i **in parallel for** $j \leftarrow 1$**to**$a - 1$
07 **do** Üzenet-szór()
08 P_{ij} **in parallel for** $i \leftarrow 1$**to**$a - 1$, $j \leftarrow 1$**to** a
09 számolja ki és tárolja $z_{i,a} \oplus y_{i+1,j}$-t
10 számítsa ki és tárolja $z_{i-1} \oplus x_i$-t

A futási idő a következő.

3.7. tétel. *A* Négyzeten-prefix *algoritmus* $a \times a$ *méretű négyzeten sorfolytonos indexeléssel* $3a + 2 = O(a)$ *lépésben elvégzi a prefixszámítást.*

3.5. példa. *Prefixszámítás* 4×4 *méretű négyzeten.* A 3.11. ábra (a) részén 16 rendezendő szám látható.



3.11. ábra. Prefixszámítás négyzeten

Az első fázisban minden sorban kiszámítjuk a prefixeket – az eredményt a (b) rész mutatja. A második fázisban csak a negyedik oszlopban számolunk – az eredmény a (c) részen látható. Végül a harmadik fázisban aktualizáljuk a prefixeket – az eredményt a (d) rész mutatja.

3.3.3. Adatkoncentráció

Tegyük fel, hogy egy p processzoros hálózatban d ($d < p$) adat van – processzoronként legfeljebb egy. *Adatkoncentráció* az a feladat, hogy az adatokat egyesével helyezzük el az első d processzornál. Lánc esetében az adatokat a P_1, P_2, ..., P_d processzorokhoz kell mozgatni. Rács esetében tetszés szerinti indexelés alkalmazható.

3.8. tétel. *Az adatkoncentráció p processzoros láncon legfeljebb $2p$ lépésben, $a \times a$ méretű négyzeten $6a + \overline{O}(p^{1/4})$ lépésben megoldható.*

Bizonyítás. ■

3.3.4. Ritka rendezés

Ha a rendezendő kulcsok száma lényegesen kisebb, mint a rendező hálózat mérete, akkor *ritka leszámláló rendezésről* (♣) beszélünk.

RITKA-RENDEZ(X, Y) *párhuzamos eljárás*
Számítási modell: négyzet
Bemenet: X (a rendezendő kulcsok)
Kimenet: Y (a rendezett kulcsok)

01 P_i **in parallel for** $1 \le j \le a$
Szórja a k_j kulcsot a j-edik oszlopban
02 P_i **in parallel for** $1 \le i \le a$
Szórja a k_i kulcsot az i-edik oszlopban
03 P_i **in parallel for** $1 \le i \le a$
Kiszámítja k_i rangját az i-edik sorban
prefixszámítást végezve
04 P_i **in parallel for** $1 \le j \le a$
Elküldi a k_j kulcs rangját $P_{i,j}$-nek
05 P_r **in parallel for** $1 \le r \le a$
az r rangú kulcs elküldése a $P_{1,r}$ processzorhoz

3.9. tétel. (*Rendezés négyzeten.*) *Ha a rendezendő kulcsok száma legfeljebb a, akkor a* RITKA-RENDEZ *algoritmus rendezés egy négyzeten $O(a)$ lépés alatt befejeződik.*

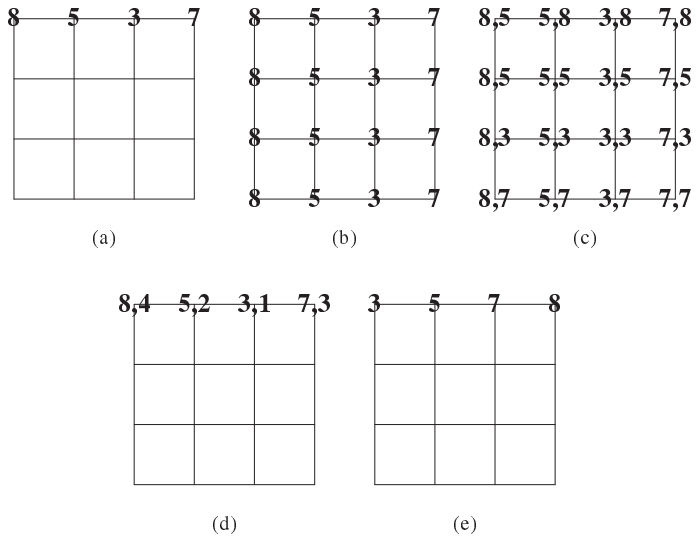
3.6. példa. 4 kulcs rendezése egy 4×4 méretű négyzeten

A 3.12. ábra mutatja a (8,5,3,7) kulcssorozat rendezését egy 4×4 méretű négyzeten. Az ábra (a) részében a bemenő adatok láthatók (a processzorok első soránál). Az első lépésben az algoritmus az oszlopokban szórja az üzenetet (a j-edik oszlopban k_j-t): az eredményt a (b) részen látjuk. A második lépésben soronként szórjuk az üzeneteket (az i-edik sorban k_i-t) – az eredmény a (c) részben látható. A 3. és 4. lépés utáni helyzetet – amikor a kulcsok és rangjaik az első sorban vannak – tükrözi a (d) ábrarész. Az (e) részben a kulcsok már rendezve vannak a processzorok első sorában.

3.4. Kiválasztás

A második fejezetben már foglalkoztunk a speciális és általános kiválasztási feladat megoldására alkalmas PRAM algoritmusokkal.

Négyzeten két változatot vizsgálunk. Az egyikben feltételezzük, hogy a kulcsok és a processzorok száma megegyezik. A második változat szerint a processzorok száma kisebb, mint a kulcsok száma. Ha a számítási modellünk PRAM, akkor az első feladatot megoldó algoritmus és a lassulási tétel segítségével olyan algoritmust kaphatunk, amely hasznosítja az elvégzett munkát. Rácsokra azonban nem ismert ilyen általános, a lassulásra vonatkozó állítás. Ezért a második esetet külön kell kezelni.



3.12. ábra. Ritka leszámláló rendezés négyzeten

3.4.1. Véletlenített algoritmus az $p = n$ esetre (⋆)

A párhuzamos gépre javasolt VÉLETLEN-HATÉKONY-KIVÁLASZT algoritmus módosítható úgy, hogy négyzeten is munkaoptimális legyen – így kapjuk a NÉGYZETEN-HATÉKONY-KIVÁLASZT algoritmust.

Erre az algoritmusra érvényes a következő állítás.

3.10. tétel. *A* NÉGYZETEN-HATÉKONY-KIVÁLASZT *algoritmus p kulcs esetében egy négyzeten $O(a)$ lépésben megoldja a kiválasztási feladatot.*

Bizonyítás. A *fokozat* a **while** ciklus magjának egyszeri végrehajtása. Korábban megmutattuk, hogy az algoritmus $\overline{O}(1)$ fokozat alatt véget ér.

A HATÉKONY-KIVÁLASZT algoritmus első szakasza rácson $O(1)$ lépést igényel. A 2–5. lépésekből álló szakaszban leírt prefixszámítás $O(a)$ lépés alatt befejeződik. A 2–6.lépésekben végzett adatkoncentrációhoz a 3.11. tétel szerint elég $O(\sqrt{p})$ lépés. A 3. és 6. szakaszban végzett ritka rendezéshez szintén elég $O(\sqrt{(p)})$ lépés. A 4. és 6. szakaszban végzett kiválasztás konstans lépésben elvégezhető, mivel rendezett sorozatból kell választani. ■

3.4.2. Véletlenített algoritmus a $p < n$ esetre (⋆)

Tegyük fel, hogy kevesebb processzorunk van, mint kulcsunk. Ha feltesszük, hogy alkalmas $c > 1$ konstanssal teljesül $n = p^c$, akkor a HATÉKONY-KIVÁLASZT algoritmus ennek a

feladatnak a megoldására is átalakítható.

Minden processzor $\frac{n}{p}$ kulccsal kezd. A **while** utasítás feltétele $N > D$ lesz (ahol D egy állandó). A második lépésben a processzorok minden hozzájuk tartozó kulcsot $\frac{1}{N^{1-(1/3c)}}$ valószínűséggel veszik a mintához. Ezért ez a lépés $\frac{n}{p}$ időt vesz igénybe. A mintában lévő kulcsok száma $\overline{O}N^{1/3c} = \overline{o}(\sqrt{p})$. A harmadik lépés változatlan és $O(\sqrt{p})$ időt vesz igénybe. Mivel a mintában csak $\overline{O}(N^{1/3c})$ kulcs van, a negyedik lépésben $O(\sqrt{p})$ idő alatt koncentrálhatók és rendezhetők. Az ötödik lépés $O(\sqrt{p})$ ideig tart, a hatodik és hetedik ugyancsak. Így minden fokozat $O(\frac{n}{p} + \sqrt{p})$ ideig tart.

Ennek az algoritmusnak a lépésszámára vonatkozik a következő tétel.

3.11. tétel. *Ha $c > 1$ állandó és $n = p^c$, akkor az n kulcs közül történő kiválasztás egy négyzeten $\overline{O}\left((\frac{n}{p} + \sqrt{p}) \lg \lg p\right)$ lépésben elvégezhető.*

Bizonyítás. A 2.3. lemmából következik, hogy az egyes fokozatokat túlélő kulcsok száma legfeljebb

$$2\sqrt{\alpha}N^{1-(1/6c)}\sqrt{\lg N} = \overline{O}(N^{1-(1/6c)}\sqrt{\lg N}, \tag{3.4}$$

ahol N az élő kulcsok száma az adott fokozat kezdetekor.

Ebből adódik, hogy az algoritmusnak csak $\overline{O}(\lg \lg p)$ fokozata van. ■

3.4.3. Determinisztikus algoritmus a $p < n$ esetre

Az algoritmus alapja, hogy az elemeket például ötös csoportokra bontja, minden csoportnak meghatározza a mediánját, majd kiszámítja ezen mediánok M mediánját. Ezután meghatározzuk M rangját (r_M), majd az $i \leq r_M$ összehasonlítás eredményétől függően elhagyjuk az M-nél nagyobb, illetve nem nagyobb elemeket. Végül a megmaradó kulcsok közül rekurzívan kiválasztjuk a megfelelőt.

Amikor ezt az algoritmust hálózatban hajtjuk végre, akkor célszerű arra törekedni, hogy a kulcsok egyenletesen legyenek a processzorok között elosztva. Ennek érdekében a mediánok M mediánját súlyozással számoljuk: minden csoportot az elemszámának megfelelő súllyal veszünk figyelembe.

Legyen $X = k_1, k_2, \ldots, k_n$ egy kulcssorozat, és legyen w_i a k_i kulcs súlya, továbbá legyen a súlyok összege W:

$$W = \sum_{i=1}^{n} w_i. \tag{3.5}$$

Ekkor az X sorozat *súlyozott mediánja* (♣) az a $k_j \in X$ kulcs, amelyre

$$\sum_{k_m \in X,\ k_m \leq k_j} w_{k_l} \geq \frac{W}{2} \tag{3.6}$$

és

$$\sum_{k_m \in X,\ k_m \geq k_j} w_{k_l} \geq \frac{W}{2}. \tag{3.7}$$

3.7. példa. Súlyozott médián meghatározása

A súlyozott médián meghatározásának egyik módja, hogy rendezzük a kulcsokat, és az így kapott $X' = k'_1, k'_2, \ldots, k'_n$ kulcssorozathoz tartozó $W' = w'_1, w'_2, \ldots, w'_n$ rendezett súlysorozatnak meghatározzuk a legkisebb olyan prefixét (az összeadás a művelet), amely már legalább $\frac{W}{2}$. Legyen $X = 1, 3, 5, 6, 4, 2$ és a megfelelő súlyok legyenek $1, 2, 3, 4, 5, 6$. Ekkor $W = 21$, $W' = 1, 6, 2, 5, 3, 4$ és a prefixek $1, 7, 9, 14, 17, 21$. Mivel 14 a legelső megfelelő prefix, ezért X – W-vel súlyozott – mediánja 4 (és ennek súlya 5).

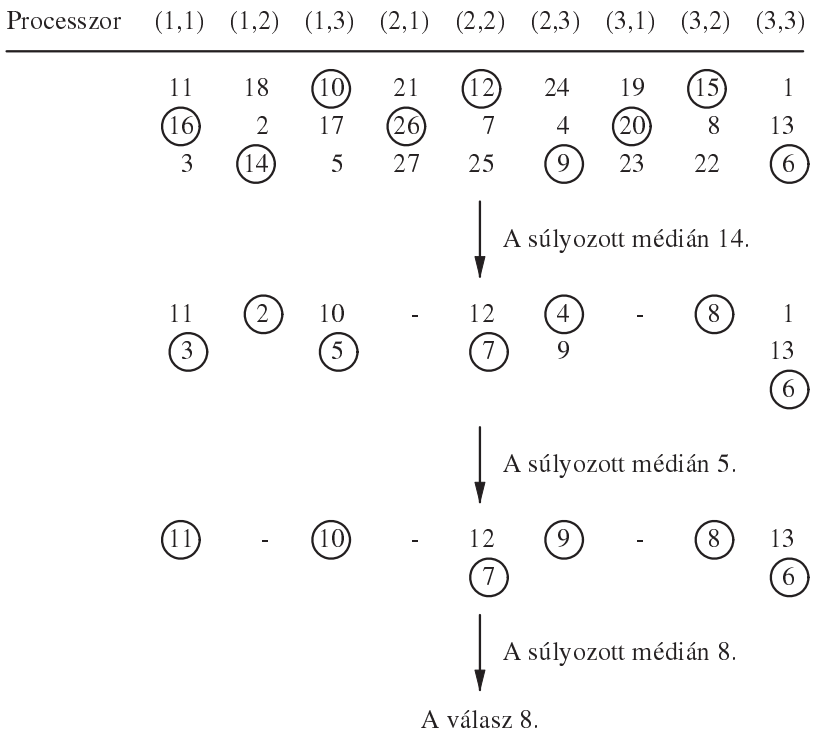
DET-NÉGYZETEN-KIVÁLASZT$(k_1, k_2, \ldots, k_p, i)$ *párhuzamos eljárás*
Számítási modell: négyzet
Bemenet: $K = k_1, \ldots, k_p$ (a kulcsok)
Kimenet: i (a kiválasztott kulcs indexe)

```
01  N := 0
02  if lg n/p ≤ lg lg p
        then minden processzor rendezzen
03      else minden processzor ossza fel a kulcsokat lg p egyenlő
             részre úgy, hogy a kulcsok minden részben
             legfeljebb akkorák, mint a tőle jobbra lévő részben.
04  while N > D
05          P_q in parallel for q ← 1 to p
                határozza meg a nála lévő kulcsok mediánját.
                Legyen M_q a médián és N_q a
                megmaradó kulcsok száma (81 ≤ q ≤ p).
06          határozzuk meg M_1, M_2, ... M_p súlyozott mediánját úgy,
            hogy M_q súlya N_q. Legyen M a súlyozott médián.
07          Meghatározzuk M rangját a maradék kulcsok között
            (legyen ez r_M) és ezt a rangot szétszórjuk.
08          if i ≤ r_M then eltávolítunk minden kulcsot,
                           amely M-nél nagyobb.
09          Kiszámítjuk és szórjuk az eltávolított kulcsok E számát.
10          if i > r_M
11            then i ← i − E
                   N ← N − E
12  return k_i
```

3.8. példa. *Kiválasztás 3×3 méretű rácson.* A 3.13. ábra determinisztikus kiválasztást mutat be.

A futási idő a következő.

3.12. tétel (kiválasztás négyzeten). *A* DET-NÉGYZETEN-KIVÁLASZT *algoritmus n kulcs közül a $\times$ a méretű négyzeten $O\left(\frac{n}{p} \lg \lg p\right) + a \lg n\Big)$ lépésben megoldja a kiválasztást.*



3.13. ábra. Determinisztikus kiválasztás 3×3 méretű négyzeten

3.5. Összefésülés

A második fejezetben több algoritmust is elemeztünk, amelyek PRAM segítségével oldották meg az összefésülési feladatot.

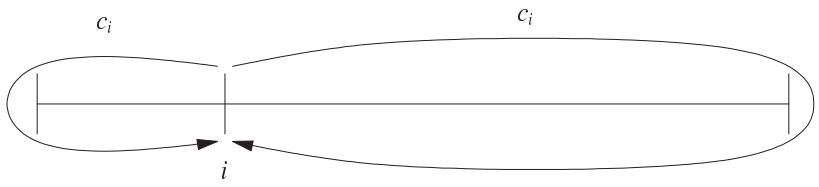
3.5.1. Rangon alapuló összefésülés láncon

A RANGSOROL rangsorolási algoritmus átalakítható úgy LÁNCON-ÖSSZEFÉSÜL algoritmussá, hogy a futási ideje lineáris legyen. Legyen $\mathcal{L}$ egy p processzoros lánc. A bemenő sorozatok legyenek $X_1 = k_1, k_2, \ldots, k_m$ és $X_2 = k_{m+1}, k_{m+2}, \ldots, k_{2m}$. Kezdetben a P_i processzornál két kulcs van: k_i és k_{i+m}.

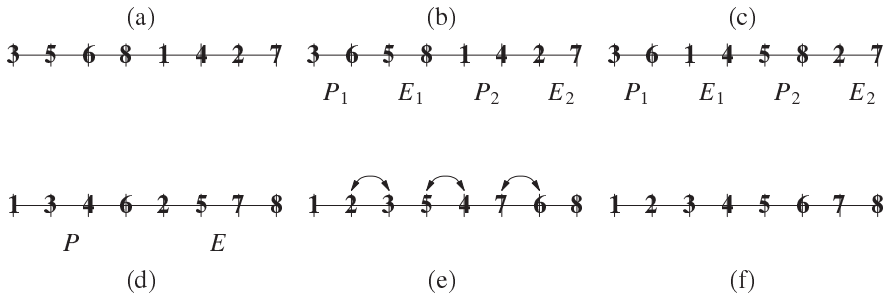
A 3.14. ábra mutatja, hogyan utaznak a c_i számlálót tartalmazó csomagok, amelyek végül visszatérnek a P_i processzorhoz.

3.13. tétel. (*Rangsorolásos összefésülés láncon.*) *Két p hosszúságú rendezett sorozat egy p processzoros láncon $O(p)$ lépésben összefésülhető.*

Bizonyítás. ■



3.14. ábra. Kulcsok rangjának számítása



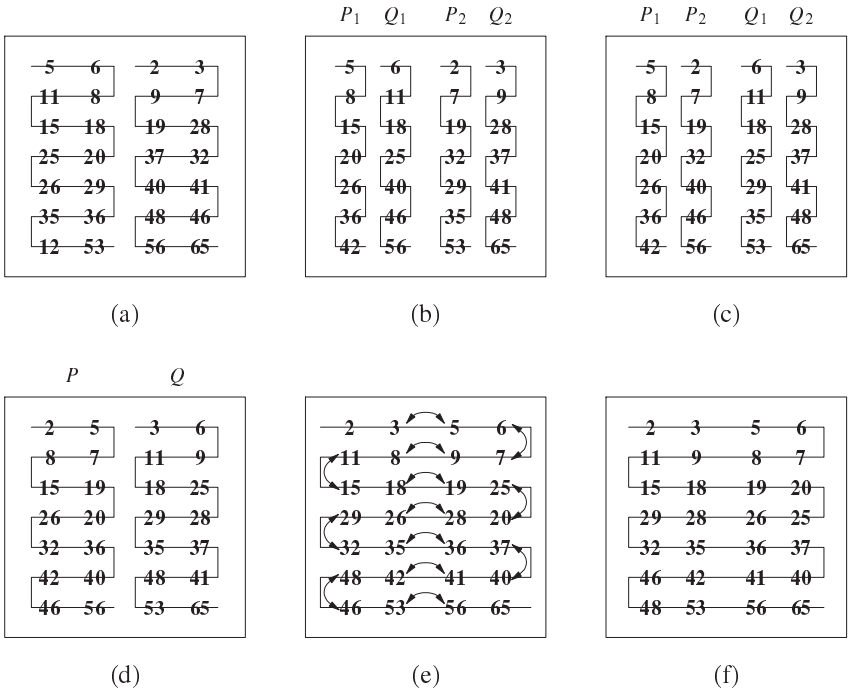
3.15. ábra. Páratlan-páros összefésülés láncon

3.5.2. Páratlan-páros összefésülés láncon

3.14. tétel. (*Páros-páratlan összefésülés láncon.*) *Két m hosszúságú sorozat egy $2m$ processzoros láncon $O(m)$ lépésben összefésülhető.*

3.9. példa. *Két 4 hosszúságú sorozat összefésülése 8 processzoron.* A 3.15. ábra mutatja, hogyan fésüli össze az algoritmus a rendezett (3,5,6,8) és (1,4,2,7) sorozatokat egy 8 processzoros láncon. Az ábra (a) része a bemenő adatokat mutatja. A (b) rész a bemenő sorozatok páros és páratlan indexű elemeket tartalmazó részsorozatokra való felbontását mutatja. A (c) rész az P_2 és Q_1 sorozatok felcserélésével kapott állapotot tükrözi.

A (d) ábrarész a P_1 és P_2, valamint a Q_1 és Q_2 sorozatok (rekurzív) összefésülésével kapott sorozatokat tartalmazza. A következő két ábrarész az összekeveréssel kapott sorozatot, illetve a cserélő-összehasonlítással kapott végeredményt ábrázolja.



3.16. ábra. Páratlan-páros összefésülés négyzeten

3.5.3. Páratlan-páros összefésülés négyzeten

Ebben a szakaszban négyzet a számítási modell. Feltesszük, hogy a 2 hatványa és a bemenő adatok két részre osztva, a 3.16. ábra (a) részének megfelelően *kígyószerűen* helyezkednek el az $1, 2, ..., a/2$, illetve az $a/2+1, a/2+2, \ldots, a$ oszlopindexű processzorok helyi memóriájában. Mindkét rész a oszlopból és $a/2$ sorból álló ***adatkígyó.***

Ennek a két rendezett sorozatnak az összefésülésére alkalmas a következő algoritmus.

NÉGYZETEN-PP-FÉSÜL(p) *párhuzamos rekurzív eljárás*
Számítási modell: négyzet
Bemenet: p (processzorszám, 2 páratlan hatványa), X_1 és X_2
(mindkettő $\frac{a}{2}$ hosszúságú rendezett sorozat)
Kimenet: Y (p hosszúságú rendezett sorozat)

01 **if** $l = 0$
then fésüljük össze a két kígyót
02 **else** cseréljük fel P_2-t és Q_1-et
03 rekurzívan fésüljük össze P_1-et és P_2-t

Most megmutatjuk, hogy ez az algoritmus $O(a)$ lépésben befejeződik.

3.15. tétel. (*Kígyók rendezése rácson.*) *A* RÁCSON-PP-FÉSÜL *algoritmus két* $a \times \frac{a}{2}$ *hosszúságú*

rendezett adatkígyót $O(a)$ lépésben összefésül egy $a \times a$ méretű négyzeten.

Bizonyítás. Az algoritmus lépésszámára az

$$M(l) \leq M\left(\frac{l}{2}\right) + 2l \tag{3.8}$$

adódik, amelynek a megoldása $M(l) = O(\sqrt{p})$. ■

3.6. Rendezés

A 2. fejezetben foglalkoztunk PRAM modellen rendező algoritmusokkal. Most lánc és rács lesznek a felhasznált számítási modellek.

3.6.1. Rendezés láncon

A LÁNCON-RANG-RENDEZ algoritmus először meghatározza a rendezendő kulcsok rangját, majd eljuttatja a kulcsokat a megfelelő helyre. A LÁNCON-BUBORÉK-RENDEZ algoritmus a soros buborékrendezéshez hasonlít.

Rangsoroló rendezés láncon
Ez az algoritmus $O(p)$ lépést tesz.

3.16. tétel. *A* LÁNCON-RANG-RENDEZ *algoritmus p kulcsot egy p processzoros láncon $O(p)$ lépésben rendez.*

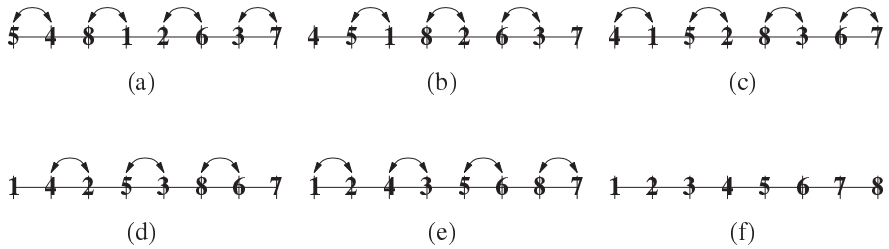
Páratlan-páros felcserélő rendezés láncon
A LÁNCON-PP-RENDEZ algoritmus alapötlete ugyanaz, mint a soros buborékrendezésé: a szomszédos kulcsokat összehasonlítjuk és szükség esetén felcseréljük.

LÁNCON-PP-RENDEZ(p) *párhuzamos eljárás*
Számítási modell: lánc
Bemenet: X (rendezendő kulcsok)
Kimenet: Y (p hosszúságú rendezett sorozat)

```
01  P_i in parallel for i ← 1 to p
02                  do if i páratlan
                          then hasonlítson össze és cseréljen
03                        else hasonlítson össze és cseréljen
```

Ez az algoritmus is $O(p)$ lépést tesz.

3.17. tétel. *A* LÁNCON-PP-RENDEZ *algoritmus egy p processzoros láncon p kulcsot $O(p)$ lépésben rendez.*



3.17. ábra. Páros-páratlan felcserélő rendezés láncon

Páratlan-páros összefésülő rendezés láncon

A LÁNCON-FÉSÜL-RENDEZ algoritmus alapötlete ugyanaz, mint a soros buborékrendezésé: a szomszédos kulcsokat összehasonlítjuk és szükség esetén felcseréljük.

Ez az algoritmus is $O(p)$ lépést tesz.

3.18. tétel. *A* LÁNCON-FÉSÜL-RENDEZ *algoritmus egy p processzoros láncon p kulcsot $O(p)$ lépésben rendez.*

3.6.2. Rendezés négyzeten

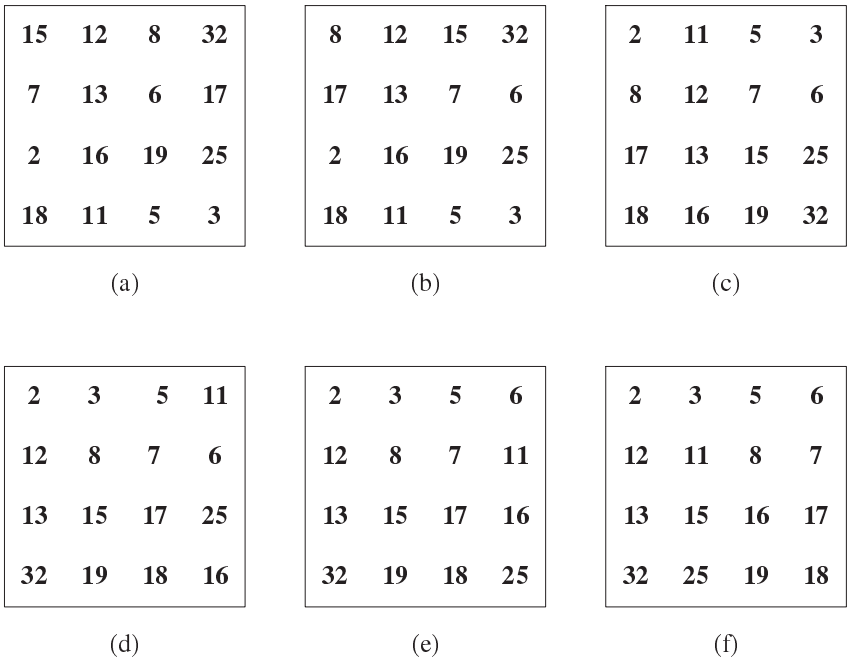
Két algoritmust vizsgálunk ebben az alfejezetben. A SHEARSON-RENDEZ $\Theta(a \lg a)$ lépést tesz, a másik viszont $O(a)$ lépésszámának köszönhetően munkahatékony.

Schearson rendező algoritmusa

SCHEARSON-RENDEZ(p) *párhuzamos eljárás*
Számítási modell: négyzet
Bemenet: X (rendezendő kulcsok)
Kimenet: Y (p hosszúságú rendezett sorozat

```
01  for i := 1 to n
02      if i páros then
03                      Rendezzük az oszlopokat
04                      Rendezzük az első sort növekvőleg
```

3.10. példa. *Schearson-rendezés.* A 3.18. ábra (a) része egy 4×4 méretű négyzeten elrendezett kulcsokat ábrázol. Az első fázisban a sorokat rendezzük – az egymást követő sorokat ellenkező módon (az első sort növekvőleg, a másodikat csökkenőleg stb.) Az első fázis eredményét mutatja az ábra (b) része. Az ábra következő részei rendre a második, . . ., ötödik fázis utáni helyzetet mutatják. Az ötödik fázis végén a négyzet rendezve van.



3.18. ábra. Példa Schearson-rendezésre

Páratlan-páros összefésülő rendezés

Most a páratlan-páros rendezési algoritmust négyzeten valósítjuk meg. Egy példát mutat a 3.19. ábra.

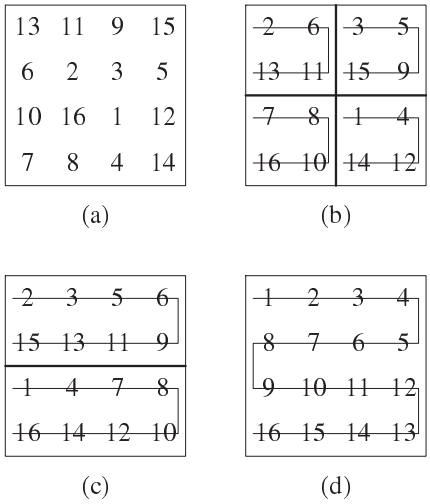
Az algoritmus futási ideje a következő.

3.19. tétel. (*Páratlan-páros összefésülő rendezés.*) *p elem $\sqrt{p} \times \sqrt{p}$ méretű négyzeten $O(\sqrt{p})$ lépésben rendezhető.*

3.11. példa. 16 elem rendezése páratlan-páros összefésüléssel A 3.19. ábra mutatja 16 szám rendezését négyzeten kígyószerű sorfolytonos sorrendbe.

3.7. Gráfalgoritmusok

Ebben az alfejezetben először néhány kockán futó gráfalgoritmust mutatunk be, majd négyzeten oldunk meg gráfokkal kapcsolatos feladatokat.



3.19. ábra. Páratlan-páros összefésülés négyzeten

3.7.1. Kocka

Ebben a szakaszban kocka lesz a számítási modell. A minmátrix, tranzitív lezárt, és az összefüggő komponensek számítását mutatjuk be.

Újra alkalmazzuk a második fejezetben a tranzitív lezárt, összefüggő komponensek és legrövidebb utak meghatározására kidolgozott formalizmust.

Először a minmátrix számítására mutatunk egy algoritmust, amely $n \times n \times n$ méretű kockán $O(n \lg n)$ lépést tesz. Ezután a tranzitív lezárt, a legrövidebb utak és a konvex burok meghatározására mutatunk be négyzeten működő algoritmusokat.

Minmátrix számítása

Az M mátrixból az $\overline{M}$ mátrixot egy kockán $O(n \lg n)$ lépésben meg tudjuk határozni.

Egy $n \times n \times n$ méretű ***kocka elemeit*** úgy definiáljuk, hogy $(i, *, *)$ azokat a processzorokat jelöli, amelyeknek az első koordinátája i.

3.20. tétel. *(Minmátrix számítása kockán.) Egy $n \times n$ méretű mátrix minmátrixa egy $n \times n \times n$ méretű kockán $O(n \lg n)$ lépésben kiszámítható.*

Irányított gráf tranzitív lezártja

Az előzőek alapján adódik a KOCKA-LEZÁR algoritmus, melynek lépésszáma a következő.

3.21. tétel. *(Tranzitív lezárt számítása kockán.) Egy n csúcsú irányított gráf tranzitív lezárt mátrixa egy $n \times n \times n$ méretű kockán $O(n \lg n)$ lépésben kiszámítható.*

Összefüggő komponensek meghatározása

3.22. tétel. (*Összefüggő komponensek számítása rácson.*) *Egy n csúcsú irányított gráf összefüggő komponensei egy $n \times n \times n$ méretű kockán $O(n \lg n)$ lépésben kiszámíthatók.*

Bizonyítás. A minmátrixra vonatkozó tétel segítségével. ■

3.7.2. Négyzet

Ebben az alfejezetben a tranzitív lezárt és a legrövidebb utak számítására determinisztikus, a teljes párosítás keresésére (páros gráfban és általános gráfban) pedig véletlenített algoritmust mutatunk be.

Tranzitív lezárt

Egy n-csúcsú gráf tranzitív lezártja meghatározható úgy, hogy $n \times n$ méretű mátrixokkal $\lceil \lg n \rceil$ szorzást végzünk. A következő tétel ezen szorzások négyzeten való gyors elvégezhetőségét mondja ki.

3.23. tétel. (*Mátrixok szorzása négyzeten.*) *Az $n \times n$ méretű $A = a[i, j]$ és $B[i, j]$ mátrixok egy $n \times n$ méretű négyzeten $O(n)$ idő alatt összeszorozhatók.*

Ennek a tételnek a segítségével belátható a következő állítás.

3.24. tétel. (*Tranzitív lezárt rácson.*) *Adott irányítatlan gráf tranzitív lezárt mátrixa egy $n \times n$ méretű négyzeten $O(n \lg n)$ lépésben meghatározható.*

Bizonyítás. A tétel bizonyítását a 3.20. ábra és a 3.21. ábra segítségével szemléltetjük. ■

Legrövidebb utak

A legrövidebb utak minden csúcspárra való meghatározására vonatkozik a következő tétel.

3.25. tétel. (*Legrövidebb utak meghatározása négyzeten.*) *A legrövidebb utak egy gráf minden csúcspárjára egy négyzeten $O(n \lg n)$ lépésben meghatározhatók.*

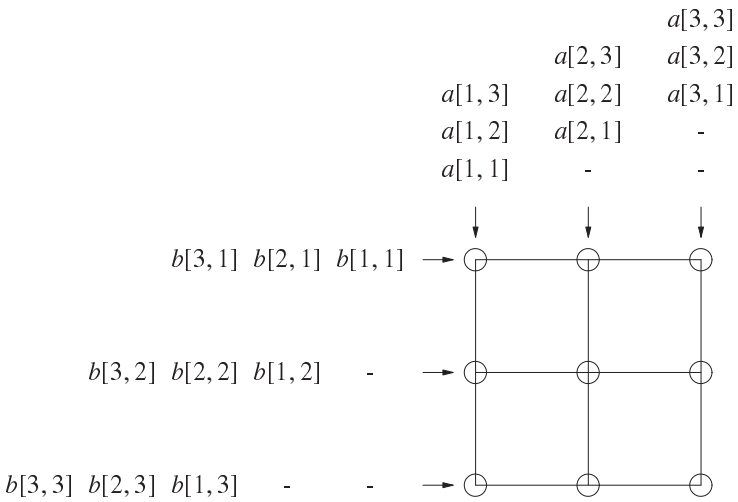
Konvex burok

n síkbeli pont konvex burka egy n-processzoros láncon $O(n)$ idő alatt meghatározható (ld. 3.10. gyakorlat). Ez az idő nagy valószínűséggel lényegesen csökkenthető.

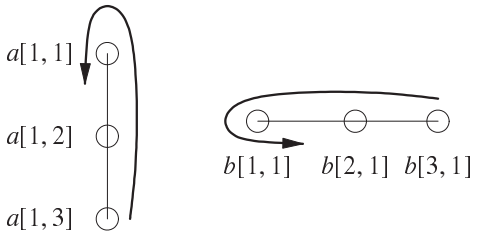
Legyen H_1 és H_2 a síkbeli pontok két felső burka.

3.26. lemma. (*Érintő számítása.*) *Legyen H_1 és H_2 két olyan felső burok, amelynek legfeljebb n pontja van. Ha P pontja H_1-nek, akkor a H_2-vel bezárt szöge $O(\sqrt{n})$ lépésben meghatározható.*

3.27. lemma. (*Két felső burok közös érintője.*) *Ha H_1 és H_2 két felső burok legfeljebb n ponttal, akkor közös érintőjük $O(\sqrt{n} \lg n)$ lépésben meghatározható.*



3.20. ábra. Két mátrix szorzása

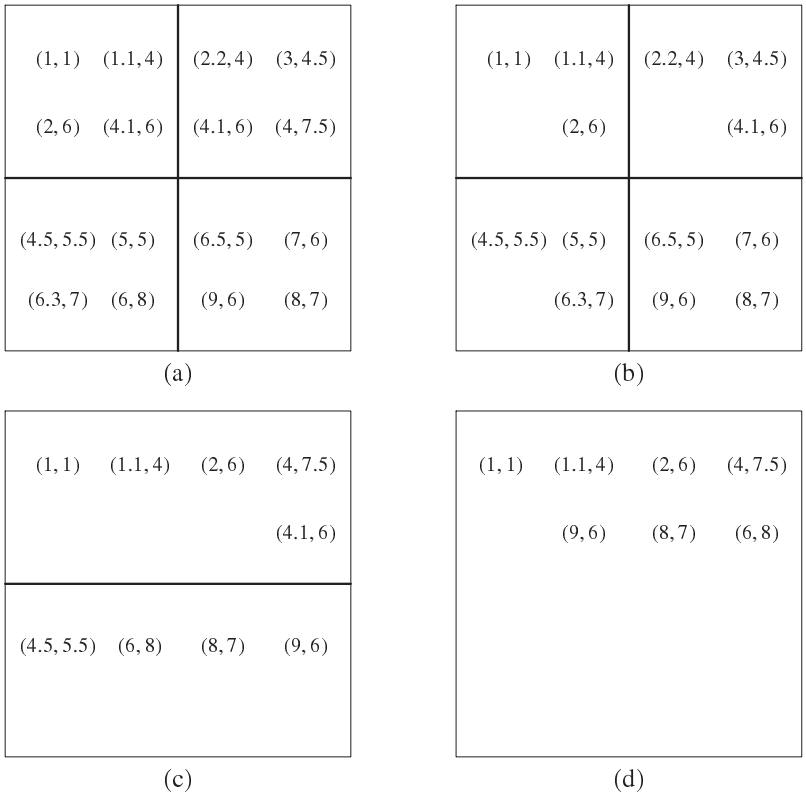


3.21. ábra. Az adatáram szimulálása

A két lemmából adódik a következő tétel.

3.28. tétel. (*Konvex burok számítása.*) *n síkbeli pont közös konvex burka egy $\sqrt{n}\times\sqrt{n}$ méretű négyzeten $O(\sqrt{n}\lg^2 n)$ lépésben meghatározható.*

3.12. példa. *16 pont konvex burka.* A 3.22. ábra (a) része 16 pont adatait mutatja. Ezek az adatok egy 4×4 méretű négyzetnek megfelelően vannak elrendezve. A 4 részre tagolt adatok minden negyedrészben az x-koordináták szerint rendezve tartalmazzák az adatokat (azonban nem sorfolytonosan, hanem kígyószerű sorrendben). A felső burok (rekurzív) kiszámításának eredményét mutatja az ábra (b) része. A két felső, és a két alsó negyedrész összefésülésének eredményét mutatja az ábra (c) része. Végül az ábra felső részén, illetve alsó részén ábrázolt burok összefésülésével kapjuk az ábra (d) részén bemutatott végeredményt.

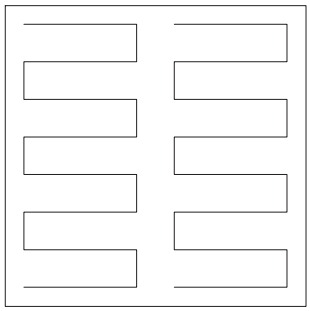


3.22. ábra. Felső burok számítása

Az összefésülés gyorsabb megoldásával csökkenthetjük az algoritmus lépésszámát. A FELSŐ-BURKOT-FÉSÜL algoritmus összesen legfeljebb a^2 pontot tartalmazó felső burkokat $O(a)$ lépésben összefésül.

FELSŐ-BURKOT-FÉSÜL(a, u, v) *párhuzamos rekurzív eljárás*
Számítási modell: négyzetrács
Bemenet: B_1 és B_2 (mindkettő legfeljebb $\frac{a^2}{2}$ pontot tartalmazó felső burok)
Kimenet: Y (p hosszúságú rendezett sorozat)

01 **if** $l = 1$ **then**
legyen u a baloldali pont és v a jobboldali pont
02 Legyen p a H_1 középső pontja. Keressük meg az érintőnek H_2-vel
03 közös pontja. Állapítsuk meg u és p relatív helyzetét.
04 Hagyjuk el H_1-nek azt a felét, amely nem tartalmazza u-t.
05 Hasonlóképpen hagyjuk el H_2 felét is.
06 Ismételjük ezt a lépést addig, amíg H_1 és H_2 negyedrésze marad
07 Rendezzük át H_1 és H_2 megmaradó pontjait úgy, hogy az $l/2 \times l/2$



3.23. ábra. Két felső burok összefésülése

08 méretű részrácsot a 3.23. ábra szerint foglalják el
09 Rekurzívan dolgozzunk a négyzeten

3.29. lemma. *A* Felső-burkot-fésül *algoritmus egy $a \times a$ méretű rácson $\overline{O}(a)$ lépésben összefésül két konvex burkot.*

Innen adódik, hogy a konvex burkot nagy valószínűséggel az előbbinél gyorsabban is meg tudjuk határozni.

3.30. tétel. (*Konvex burok négyzeten.*) *A* Konvex-burok-négyzeten *algoritmus egy $a \times a$ méretű négyzeten $\overline{O}(a)$ lépésben meghatározza a^2 pont konvex burkát.*

Gyakorlatok

3.7-1. Mennyi a FOF és a LIFO elsőbbségi algoritmusok lépésszáma a 3.1. példa adatai esetében?

3.7-2. Egy p processzoros lánc minden processzoránál két csomag van. Feltesszük, hogy p páros. A P_i processzornál lévő csomagok rendeltetési helye a $P_{p/2+i}$ ($i = 1, 2, \ldots, \frac{p}{2} + i$). Minden csomag a számára legrövidebb úton jut el a célba. Határozzuk meg a csomagok utazási idejét a FOF, FDF, FIFO és LIFO elsőbbségi módszerek esetében.

3.7-3. Egy $a \times a$ méretű rácsban minden processzortól legfeljebb $k \geq 1$ csomag indul és minden processzorhoz legfeljebb k csomag érkezik. Tervezzünk algoritmust, amely legfeljebb $\frac{(k+1)p}{2}$ lépésben megoldja a csomagirányítási feladatot.

3.7-4. Mutassuk meg, hogy egy p processzoros gyűrűben a PPR probléma $\frac{p}{2}$ lépésben megoldható.

3.7-5. Hogyan oldható meg a szuffixszámítási feladat egy $\sqrt{p} \times \sqrt{p}$ méretű rácson $O(\sqrt{p})$ lépésben?

3.7-6. p elem közül kell a maximálisat megkeresni egy $\sqrt{p} \times \sqrt{p}$ méretű rács segítségével. Az $\mathcal{A}$ algoritmus $T(\sqrt{p})$ lépésben meghatározza p elem súlyozott mediánját. Hogyan használható fel $\mathcal{A}$ a keresésre és milyen lépésszámot kapunk?

3.7-7. Legyen

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0. \tag{3.9}$$

Tervezzünk algoritmust, amely láncon és rácson meghatározza az $f(x)$ polinom értékét egy $x = x_0$ helyen. Mennyi lesz a tervezett algoritmus lépésszáma?

3.7-8. Tekintsünk egy $\sqrt{p} \times \sqrt{p}$ méretű négyzetet, melynek minden processzorának memóriájában van egy, a $[0,\ p^{\epsilon}]$ intervallumból vett egész szám, ahol ϵ a $(0, 1)$ intervallumból vett állandó. Tervezzünk algoritmust a legnagyobb kulcs kiválasztására. Mennyi lesz e tervezett algoritmus lépésszáma?

3.7-9. Valósítsuk meg a rangsorolási algoritmust egy $\sqrt{p} \times \sqrt{p}$ méretű rácson.

3.7-10. Mutassuk meg, hogy p pont konvex burka $O(p)$ lépésben meghatározható egy p processzoros láncon.

3.7-11. Tervezzünk algoritmust, amely láncon és rácson meghatározza, hogy adott n pont között van-e 3 olyan pont, amelyek egy egyenesre esnek. Mit mondhatunk algoritmusaink lépésszámáról és a felhasznált processzorok számáról?

Feladatok

3-1. Csomagok irányítása közeli célokhoz

Egy $a \times a$ méretű rácsban minden processzor legfeljebb egy üzenetet küld és legfeljebb d távolságra. Tervezzünk csomagirányítási algoritmust, amely $\overline{O}(d)$ lépést tesz.

3-2. Csomagirányítás tóruszon

Módosítsuk úgy a Három-fázisú algoritmust, hogy tóruszon $1.5\sqrt{p} + \overline{o}(p)$ lépést tegyen.

3-3. Palindrómák ellenőrzése

Egy adott Σ ábécé feletti $w = x_1 x_2 \ldots x_p$ szót akkor nevezünk *palindrómának*, ha $w = x_p x_{p-1} \ldots x_1$ teljesül. Tervezzünk párhuzamos algoritmust, amely egy p processzoros láncon $O(p)$ lépésben eldönti, hogy egy p hosszúságú szó palindróma-e.

3-4. Gyors Fourier-transzformált kiszámítása

Tervezzünk algoritmust, amely egy p processzoros láncon $O(p)$ lépésben kiszámítja egy p hosszúságú vektor FFT-jét (gyors Fourier-transzformáltját). Tervezzünk rácsot használó algoritmust az FFT kiszámítására. Milyen lépésszám érhető el?

3-5. Egycsomagos csomagirányítás

Tegyük fel, hogy egy $\sqrt{p} \times \sqrt{p}$ méretű rácsban minden processzor pontosan egy csomagot küld és pontosan egy csomagot fogad. Tervezzünk olyan csomagirányító algoritmust, melynek lépésszáma $O(\sqrt{p})$, az igényelt sorhosszúsága pedig $O(1)$.

3-6. Csoportokba rendezés láncon

Tegyük fel, hogy egy $\lg n$ processzoros lánc minden processzorának memóriájában $n/\lg n$ kulcs van. Tervezzünk algoritmust, amely biztosítja, hogy P_1 memóriájába kerüljön a legkisebb $n/\lg n$ kulcs, P_2 memóriájába a következő $n/\lg n$ legkisebb kulcs, és így tovább, a

legnagyobb indexű processzor memóriájába kerüljön a $n/\lg n$ legnagyobb kulcs. Mutassuk meg, hogy ez a *rendezés* megoldható $O(n)$ lépésben.

3-7. Soronkénti és oszloponkénti rendezés
Mutassuk meg, hogy ha egy $\sqrt{p}\times\sqrt{p}$ méretű rácsban minden processzor memóriájában van egy kulcs és ezeket a kulcsokat először soronként, azután oszloponként rendezzük, akkor a sorok és oszlopok is rendezettek lesznek.

3-8. Prefix számítása bináris fával
Processzorok bináris fája (röviden: bináris fa) olyan teljes bináris fa, melynek minden csúcsában van egy processzor és az élek adatátviteli vonalak. A 4.8. ábra egy 4 levelű bináris fát mutat. A bemenő adatok rendszerint a fa levelein jelennek meg. A n levelű bináris fában $2n-1$ processzor van és a fa magassága $\lceil\lg n\rceil$. Ha minden levélen van egy szám, ezen számok összegét kiszámíthatjuk a következőképpen. Először minden levél elküldi a nála lévő számot a szülőjének. Ha egy belső processzor kap két számot alulról, akkor összeadja őket és az összeget elküldi a szülőjének. Ilymódon $\lg n$ lépés után a gyökérben megjelenik az összeg.

Oldjuk meg a prefixszámítási problémát egy n levelű bináris fával. Kezdetben minden levélnél van egy elem. A prefixek értékét a levelekről lehet kivinni. Hogyan oldható meg a feladat $O(n)$ lépésben?

3-9. Topologikus rendezés rácson
Tervezzünk algoritmust, amely rács segítségével topologikusan rendez.

3-10. Körmentesség ellenőrzése
Tervezzünk algoritmust, amely rácson ellenőrzi, hogy adott irányítatlan gráf tartalmaz-e kört?

3-11. Minimális feszítőfa 0-1 súlyok esetében
Tegyük fel, hogy az irányított gráfok éleinek súlya nulla vagy egy lehet. Tervezzünk rácsalgoritmust, amely meghatározza a minimális feszítőfát.

3-12. Háromszög mátrix invertálása négyzeten
Mutassuk meg, hogyan lehet egy $a\times a$ méretű háromszög mátrixot $O(a)$ lépésben invertálni egy $a\times a$ méretű négyzeten.

3-13. Háromátlós mátrix invertálása négyzeten
Mutassuk meg, hogyan lehet egy $a\times a$ méretű háromátlós mátrixot $O(a)$ lépésben invertálni egy $a\times a$ méretű négyzeten.

3-14. Konvex burok területe
Tervezzünk olyan algoritmust, amely egy $a\times a$ méretű négyzeten $\overline{O}(a)$ lépésben meghatározza a^2 pont konvex burkának területét.

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*[*szomszéd*]. Az algoritmusok neveit kis kapitális betűkkel írtuk, mint például Kiválaszt. Az algoritmusok kódjában a programozási alapszavakat félkövéren szedtük, mint például **if, then, else, in parallel for, do**.

Az algoritmusok nevében kiskötőjelet használtunk, viszont a változók neveiben alsó kötőjelet, mint például PP-összefésül és *bal_szomszéd*. Az egyes fogalmak meghatározásának helyére a tárgymutató dőlt oldalszámmal utal.

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

Névmutató

Tartalomjegyzék