2. Párhuzamos gépek

Ebben a fejezetben először két alapvető módszert (a prefixszámítást és a tömbelemek rangsorolását), (2.1. alfejezet), azután az összefésülést (2.2. alfejezet), kiválasztást (2.3. alfejezet), rendezést (2.4. alfejezet), végül pedig néhány gráfalgoritmust (2.5. alfejezet) mutatunk be.

2.1. Alapvető módszerek

Ebben az alfejezetben két, a feladatok párhuzamos megoldásában gyakran használt módszert mutatunk be. Az asszociatív műveletek gyors elvégzését a prefixek számítására, a mutatóugrást pedig a listarangsorolási feladat megoldására használjuk fel.

2.1.1. Prefixszámítás

Legyen Σ egy alaphalmaz, melyen definiáltuk a $\oplus$ ***bináris asszociatív operátort.*** Feltesszük, hogy a művelet egy lépéssel elvégezhető és a Σ halmaz zárt erre a műveletre nézve.

Egy $\oplus$ bináris operátor ***asszociatív*** a Σ alaphalmazon, ha minden $x, y, z \in \Sigma$ esetében teljesül

$$(x \oplus y) \oplus z = x \oplus (y \oplus z) \ . \tag{2.1}$$

Legyenek az $X = \langle x_1, x_2, \ldots, x_p \rangle$ sorozat elemei a Σ alaphalmaz elemei. Ekkor a prefixszámítás bemenő adatai az X sorozat elemei, a ***prefixszámítási feladat*** pedig az $x_1,\ x_1 \oplus x_2,\ \ldots,\ x_1 \oplus x_2 \oplus x_3 \oplus \cdots \oplus x_p$ elemek meghatározása. Ezeket a meghatározandó elemeket ***prefixeknek*** hívjuk.

Érdemes megjegyezni, hogy más területeken inkább az X sorozat $x_1, x_2, \ldots, x_k$ kezdősorozatait hívják prefixeknek.

2.1. példa. *Asszociatív műveletek.* Ha Σ az egészek halmaza, $\oplus$ az összeadás

és a bemenő adatok sorozata a 3, -5, 8, 2, 5, 4, akkor a prefixek 3, -2, 6, 8, 13, 17. Ha az ábécé és a bemenő adatok ugyanazok, de a művelet a szorzás, akkor a kimenő adatok (prefixek) 3, -15, -120, -240, -1200, -4800. Ha a művelet a minimum (ez is asszociatív), akkor a prefixek 3, -5, -5, -5, -5, -5. Az utolsó prefix megegyezik a bemenő számok legkisebbikével. ♠

A prefixszámítás sorosan megoldható $O(p)$ lépéssel. Bármely A soros algoritmusnak $N(p, \mathrm{A}) = \Omega(p)$ lépésre szüksége van. Vannak olyan párhuzamos algoritmusok, amelyek különböző párhuzamos modelleken megoldják a munkahatékony prefixszámítást.

Először a CREW-PREFIX CREW PRAM algoritmust mutatjuk be, amely p processzoron $\Theta(\lg p)$ lépéssel számítja ki a prefixeket.

Azután az EREW-PREFIX algoritmus következik, amelynek menynyiségi jellemzői hasonlóak az előző algoritmuséhoz, azonban EREW PRAM számítási modellen is végrehajtható.

Ezekkel az algoritmusokkal a prefixszámítást a soros megoldás $\Theta(p)$ lépésszámánál lényegesen gyorsabban meg tudjuk oldani, de sok az elvégzett munka.

Ezért is érdekes az OPTIMÁLIS-PREFIX CREW PRAM algoritmus, amely $\lceil p/\lg p \rceil$ processzort használ és $\Theta(\lg p)$ lépést tesz. Ennek elvégzett munkája csak $O(p)$, ezért a hatékonysága $\Theta(1)$ és az algoritmus munkaoptimális. Ennek az algoritmusnak a gyorsítása $\Theta(n/\lg n)$.

A továbbiakban – a jelölések egyszerűsítése érdekében – a $\lceil p/\lg p \rceil$ típusú kifejezések helyett általában csak $(p/\lg p)$-t írunk.

Az algoritmusok tervezéséhez az *oszd-meg-és-uralkodj* elvet alkalmazzuk. A bemenő adatok legyenek $X = x_1, x_2, \ldots, x_p$. Az általánosság megszorítása nélkül feltehető, hogy p a 2 egész kitevőjű hatványa.

Prefixszámítás CREW PRAM modellen

Először egy p processzoros algoritmust mutatunk be, melynek lépésszáma $\Theta(\lg p)$.

CREW-PREFIX(p, X) *párhuzamos rekurzív eljárás*
Számítási modell: PRAM
Bemenet: p (a bemenő sorozat hossza) és
$X[1 : p] = x_1, x_2, \ldots, x_p$ (p hosszúságú sorozat)
Kimenet: $Y[1 : p] = y_1, y_2, \ldots, y_p$ (p hosszúságú sorozat, amelynek elemei a prefixek)

```
01 if p = 1
```

02 **then** $y_1 \leftarrow x_1$
03 **return** Y
04 **if** $p > 1$
05 **then** Az első $p/2$ processzor rekurzívan számítsa az $x_1, x_2, \ldots, x_{p/2}$-höz tartozó prefixeket – legyenek ezek $y_1, y_2, \ldots, y_{p/2}$ (ezek adják a végeredmény első felét). Ugyanakkor a többi processzor rekurzívan számítsa ki az $x_{p/2+1}, x_{p/2+2}, \ldots, x_p$-hez tartozó prefixeket – legyenek ezek $y_{p/2+1}, y_{p/2+2}, \ldots, y_p$.
06 A processzorok másik fele párhuzamosan olvassa ki a globális memóriából $y_{p/2}$-t és az $y_{p/2} \oplus y_{p/2+1}, y_{p/2} \oplus y_{p/2+2}, \ldots, y_{p/2} \oplus y_p$ prefixeket számítva állítsa a végeredmény másik felét.

2.2. példa. *8 elem prefixeinek számítása 8 processzoron.* Legyen $n = 8$ és $p = 8$. A prefixszámítás bemenő adatai 12, 3, 6, 8, 11, 4, 5 és 7, az asszociatív művelet az összeadás. Az első szakaszban az első 4 processzor 12, 3, 6, 8 bemenethez a 12, 15, 21, 29 prefixeket számolja ki. A másik 4 processzor pedig a 11, 4, 5, 7 bemenethez számolja ki a 11, 15, 20, 27 prefixeket.

A második szakaszban az első 4 processzor nem dolgozik, a második 4 pedig 29-et ad minden prefixhez és a 40, 44, 49, 56 eredményt kapja. ♠

Mi ennek az algoritmusnak a $T(p)$ lépésigénye? Az első lépés $T(p/2)$ ideig, a második pedig $O(1)$ ideig tart. Ezért a következő rekurziót kapjuk:

$$T(p) = T\left(\frac{p}{2}\right) + O(1)\,, \tag{2.2}$$

$$T(1) = 1\,. \tag{2.3}$$

Ennek a rekurzív egyenletnek a megoldása $T(p) = O(\lg p)$.

2.1. tétel. *A* CREW-PREFIX *algoritmus p* CREW PRAM *processzoron $\Theta(\lg p)$ lépésben számítja ki p elem prefixeit.*

Ez az algoritmus nem munkaoptimális, mivel $\Theta(p \lg p)$ munkát végez, és ismert olyan soros algoritmus, amely $O(p)$ lépést tesz. Munkaoptimális algoritmust kaphatunk például úgy, ha a felhasznált processzorok számát lecsökkentjük $(p/\lg p)$-re, miközben a lépésszám nagyságrendje ugyanaz marad. A processzorszámot úgy csökkentjük, hogy a bemenet méretét

csökkentjük $(p/\lg p)$-re, alkalmazzuk az előző algoritmust, majd végül minden prefixet kiszámolunk.

Prefixszámítás EREW PRAM modellen

A következő algoritmusban a párhuzamos olvasás helyett elég a soros olvasás lehetősége.

EREW-PREFIX(p, X, Y) *párhuzamos rekurzív eljárás*
Számítási modell: EREW PRAM
Bemenet: p (a bemenő sorozat hossza) és $X[0 : p] = x_0,\ x_1,\ x_2,\ \ldots,\ x_p$ (p hosszúságú sorozat)
Kimenet: $Y[1 : p] = y_1,\ y_2,\ \ldots,\ y_p$ (p hosszúságú sorozat, melynek elemei a prefixek)

```
01  Y[1] ← X[1]
02  P_i in parallel for i ← 2 to p
03                       Y[i] ← X[i − 1] ⊕ X[i]
04  k ← 2
05  while k < p
06        P_i in parallel for i ← k + 1 to p
07                          do Y[i] ← Y[i − k] ⊕ Y[i]
08        k ← k + k
09  return Y
```

2.2. tétel. *Az* EREW-PREFIX *algoritmus p* EREW PRAM *processzoron* $\Theta(\lg p)$ *lépésben számítja ki p elem prefixeit.*

Bizonyítás. A lépésszám nagyságrendjét az határozza meg, hányszor hajtódik végre az utolsó sorban lévő utasítás. Az 1–4. és 9. lépések $O(1)$ idő alatt, az 5–8. lépések $\Theta(\lg p)$ idő alatt hajtódnak végre. ∎

Prefixszámítás munkaoptimálisan

OPTIMÁLIS-PREFIX(p, X) *párhuzamos rekurzív eljárás*
Számítási modell: CREW PRAM
Bemenet: p (a bemenő sorozat hossza) és $X[1\,..\,p] = \langle x_1,\ x_2, \ldots,\ x_p\rangle$ (p hosszúságú sorozat)
Kimenet: $Y[1\,..\,p] = \langle y_1,\ y_2, \ldots,\ y_p\rangle$ (p hosszúságú sorozat, melynek elemei a prefixek)

01 P_i processzor ($i = 1, 2, \ldots, p/\lg p$) sorosan számolja a hozzárendelt $\lg p$ darab $x_{(i-1)\lg p+1}, x_{(i-1)\lg p+2}, \ldots, x_{i\lg p}$ elem prefixeit.
Legyen az eredmény $z_{(i-1)\lg p+1}, z_{(i-1)\lg p+2}, \ldots, z_{i\lg p}$.
02 Összesen $\frac{p}{\lg p}$ processzor együtt alkalmazza a CREW-PREFIX algoritmust a $\frac{p}{\lg p}$ darab elem, $z_{\lg p}, z_{2\lg p}, z_{3\lg p}, \ldots, z_p$ prefixeinek számítására. Legyen az eredmény $w_{\lg p}, w_{2\lg p}, w_{3\lg p}, \ldots, w_p$.
03 Minden processzor aktualizálja az első lépésben kiszámolt értéket: a P_i processzor ($i = 2, 3, \ldots, p/\lg p$) számolja ki a $w_{(i-1)\lg p} \oplus z_{(i-1)\lg p+1}, w_{(i-1)\lg p} \oplus z_{(i-1)\lg p+2}, \ldots, w_{(i-1)\lg p} \oplus z_{i\lg p}$ prefixeket, majd az első processzor változtatás nélkül adja ki a $z_1, z_2, \ldots, z_{\lg p}$ prefixeket.

Az algoritmus lépésszáma logaritmikus. Ennek belátását megkönnyíti a következő két képlet:

$$z_{(i-1)\lg p+k} = \sum_{j=(i-1)\lg p+1}^{i\lg p} x_j \quad (k = 1,\ 2,\ \ldots,\ \lg p) \tag{2.4}$$

és

$$w_{i\lg p} = \sum_{j=1}^{i} z_{j\lg p} \ (i = 1,\ 2,\ \ldots)\ , \tag{2.5}$$

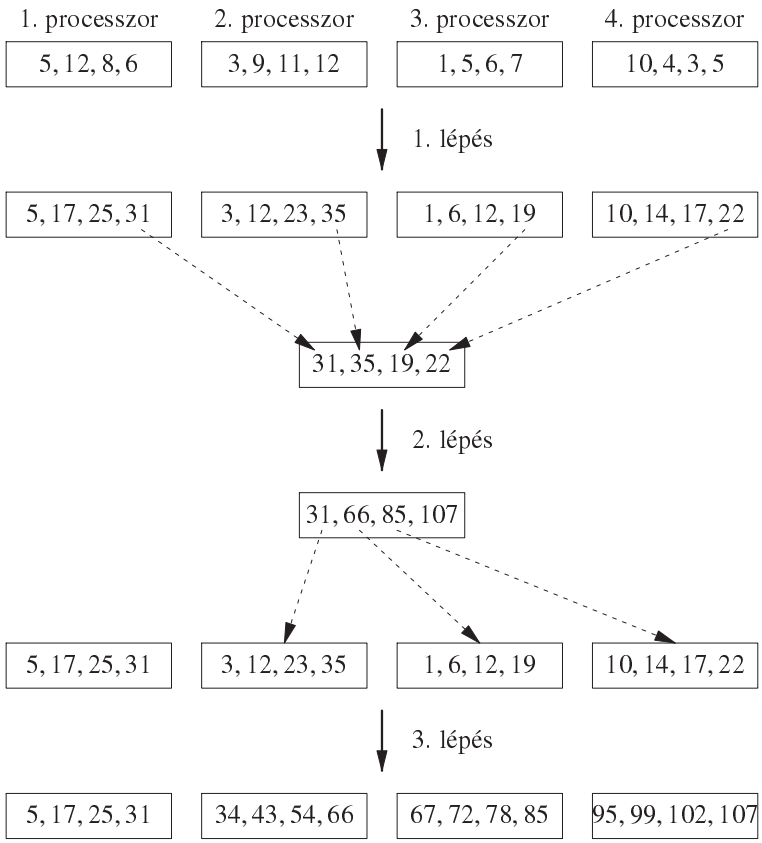
ahol az összegzés a megfelelő asszociatív művelet segítségével történik.

2.3. tétel (párhuzamos prefixszámítás $\Theta(\lg p)$ lépéssel). *Az* OPTIMÁLIS-PREFIX *algoritmus* ($p/\lg p$) CREW PRAM *processzoron* $\Theta(\lg p)$ *lépéssel számítja ki* p *elem prefixeit.*

Bizonyítás. Az algoritmus az első szakasza $O(\lg p)$ ideig, a második szakasza $O(\lg(p/\lg)) = O(\lg p)$ lépésig tart. Végül a harmadik szakasz ugyancsak $O(\lg p)$ lépést igényel. ∎

A tételből következik, hogy az OPTIMÁLIS-PREFIX algoritmus munkaoptimális.

2.3. példa. *16 elem összeadása.* Legyen 16 elemünk: 5, 12, 8, 6, 3, 9, 11, 12, 1, 5, 6, 7, 10, 4, 3, 5. Az asszociatív művelet az összeadás. Ekkor $\lg n = 4$.



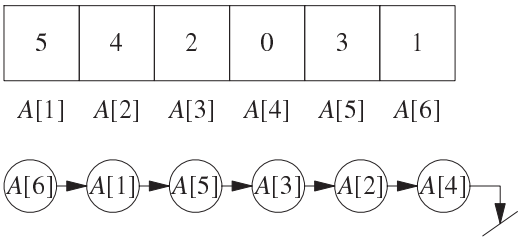
2.1. ábra. 16 elem prefixeinek számítása az OPTIMÁLIS-PREFIX algoritmussal.

Ekkor az első párhuzamos lépésben a processzorok 4-4 elem prefixeit számolják. A második lépésben a helyi összegekből globális összegeket számolunk, majd azokkal a harmadik lépésben frissítjük a helyi eredményeket. A számítás menetét mutatja a 2.1. ábra. ♠

2.1.2. Tömb elemeinek rangsorolása

A ***tömbrangsorolási feladat*** bemenő adata egy p elemű tömbben ábrázolt lista: minden elem tartalmazza jobb oldali szomszédjának az indexét (és esetleges további adatokat). A feladat az elemek ***rangjának*** (jobb oldali szomszédai számának) meghatározása.

Mivel az adatokra nincs szükség a megoldáshoz, feltesszük, hogy az ele-



2.2. ábra. Tömbrangsorolási probléma bemenő adatai és a megfelelő tömb.

mek csak a szomszéd indexét tartalmazzák. A jobb szélső elem index mezője nulla. Az indexet a továbbiakban mutatónak hívjuk.

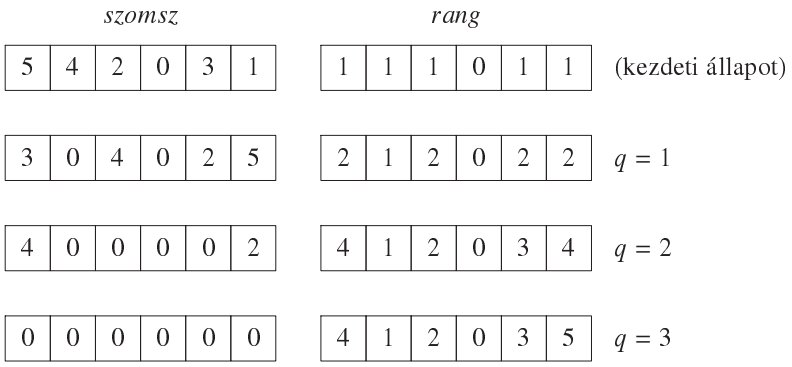
2.4. példa. *Tömbrangsorolás bemenő adatai.* Legyen $A[1:6]$ a 2.2. ábra felső sorában bemutatott tömb. Ekkor az $A[1]$ elem jobboldali szomszédja $A[5]$, $A[2]$ jobboldali szomszédja $A[4]$. $A[4]$ az utolsó elem, ezért rangja 0. $A[2]$ rangja 1, mivel csak $A[4]$ van tőle jobbra. $A[5]$ rangja 3, mivel az $A[3], A[2]$ és $A[4]$ elemek vannak tőle jobbra. Az elemek sorrendjét (balról jobbra haladva) mutatja az ábra alsó része. ♠

A tömbrangsorolás sorosan elvégezhető lineáris lépésszámmal. Először meghatározzuk a ***tömb fejét***– az egyetlen olyan i értéket ($1 \le i \le p$), melyre $A[j] \neq i$ teljesül minden $1 \le j \le n$ értékre. Legyen $A[i]$ a tömb feje. A fejtől kiindulva pásztázzuk a tömböt és az elemekhez rendre hozzárendeljük a $p-1, \ldots, 1, 0$ rangokat.

Ebben a részben két párhuzamos algoritmust ismertetünk. Az egyik a DET-RANGSOROL, amely egy p processzoros EREW PRAM modellen $\Theta(\lg p)$ futási időt igényel, a másik a VÉL-RANGSOROL, amely egy $(p/\lg p)$ processzoros véletlenített EREW PRAM algoritmus $\overline{O}(\lg p)$ lépésszámmal. Mindkettő relatív sebessége $\Theta(p/\lg p)$. Az első algoritmus hatékonysága $(\Theta(p)/(\Theta(p \lg p)) = \Theta(1/\lg p)$, míg a másodiké $\Theta(1.)$ Ezért az első algoritmus csak munkahatékony, a második algoritmus viszont munkaoptimális is.

Determinisztikus tömbrangsorolás

Ezekben az algoritmusokban az egyik alapvető ötlet a *mutatóugrás*. DET-RANGSOROL szerint először mindegyik elem a jobb oldali szomszédjának indexét tartalmazza, és ennek megfelelően a rangja – a jobb oldali szomszéd-



2.3. ábra. A DET-RANGSOROL algoritmus működése a 2.4. példa adataival.

jához viszonyítva – 1 (kivétel a lista utolsó eleme, melynek rangja 0. Ezt a kezdeti állapotot mutatja a 2.3. ábra első sora.

Ezután módosítjuk a csúcsokat úgy, hogy mindegyik a jobb oldali szomszédjának a jobb oldali szomszédjára mutasson (ha nincs, akkor a lista végére). Ezt tükrözi a 2.3. ábra második sora.

Ha p processzorunk van, akkor ez $O(1)$ lépéssel elvégezhető.

Most minden csúcs (kivéve az utolsót) olyan csúcsra mutat, amelyik eredetileg 2 távolságra volt. A mutatóugrás következő lépésében a csúcsok olyan csúcsra mutatnak, amelyek eredetileg 4 távolságra voltak tőlük (ha ilyen csúcs nincs, akkor a lista végére) – amint azt az ábra harmadik sora mutatja.

A következő lépésben a csúcsok (pontosabban a mutató részük) a 8 távolságú szomszédra mutatnak (ha van ilyen – ha nincs, akkor a lista végére), az 2.3. ábra utolsó sora szerint.

Minden csúcs minden lépésben információt gyűjt arról, hány csúcs van közte és azon csúcs között, amelyre most mutat. Ehhez kezdetben legyen a csúcsok rang mezőjében 1 – kivéve a jobboldali csúcsot, melyre ez az érték legyen 0. Legyen $rang[i]$ és $szomsz[i]$ az i csúcs rang, illetve szomszéd mezője. A mutatóugrás során $rang[i]$-t általában $rang[i] + rang[szomsz[i]]$-re módosítjuk – kivéve azokat a csúcsokat, melyekre $szomsz[i] = 0$. Ezután $szomsz[i]$-t úgy módosítjuk, hogy $szomsz[szomsz[i]]$-re mutasson. A teljes DET-RANGSOROL algoritmus a következő.

DET-RANGSOROL($szomsz[1 : p], rang[1 : p]$) *párhuzamos eljárás*
Számítási modell: EREW PRAM

Bemenet: $szomsz[1:p]$ (az elemek jobboldali szomszédainak indexei)
Kimenet: $rang[1:p]$ (az elemek rangjai)

```
01 P_i in parallel for i ← 1 to p
02                     if szomsz[i] = 0
03                       then rang[i] ← 0
04                       else rang[i] ← 1
05 for j ← 1 to ⌈lg p⌉
06     do P_i in parallel for i ← 1 to p
07                             if szomsz[i] ≠ 0
08                               then rang[i] ← rang[i] + rang[szomsz[i]]
09                                    szomsz[i] ← szomsz[szomsz[i]]
```

A 2.3. ábra mutatja, hogyan működik DET-RANGSOROL a 2.6. példa adataival.

Kezdetben minden csúcs rangja 1, kivéve a 4. csúcsot. Amikor $q = 1$, akkor például az 1. csúcs *rang* mezőjét kettőre változtatjuk, mert jobboldali szomszédjának (ez az 5. csúcs) rangja 1. Az 1. csúcs *szomsz* mezőjét az 5. csúcs szomszédjának indexére, azaz 3-ra változtatjuk.

2.4. tétel. *A* DET-RANGSOROL *algoritmus egy* EREW PRAM *modellen p processzoron* $\Theta(\lg p)$ *lépésben határozza meg egy p elemű tömb elemeinek rangját.*

Mivel a A DET-RANGSOROL algoritmus $\Theta(p \lg p)$ munkát végez, ezért nem munkaoptimális.

A listarangsorolási probléma megfelel a lista prefix összege számításának, ahol minden csúcs súlya 1, kivéve a jobboldalit, melynek súlya 0. A DET-RANGSOROL algoritmus könnyen módosítható úgy, hogy kiszámítsa egy lista prefixeit – a processzorszámra és a lépésszámra vonatkozó hasonló korlátokkal.

Véletlenített listarangsorolás (⋆)

Most egy munkahatékony véletlen listarendező algoritmus következik. Minden processzor $\lg p$ csúcs rangját számolja. A P_i processzort az $A[(i-1)\lg p+1], A[(i-1)\lg p+2], \ldots, A[i \lg p]$ csúcsokhoz rendeljük. Az algoritmus meneteket hajt végre. Minden menetben kiválasztja és ***kiemeli*** a csúcsok egy részét. Amikor egy csúcsot kiemelünk, akkor a rá vonatkozó információt tároljuk úgy, hogy később a rangját meg tudjuk határozni. Ha már csak 2 csúcs

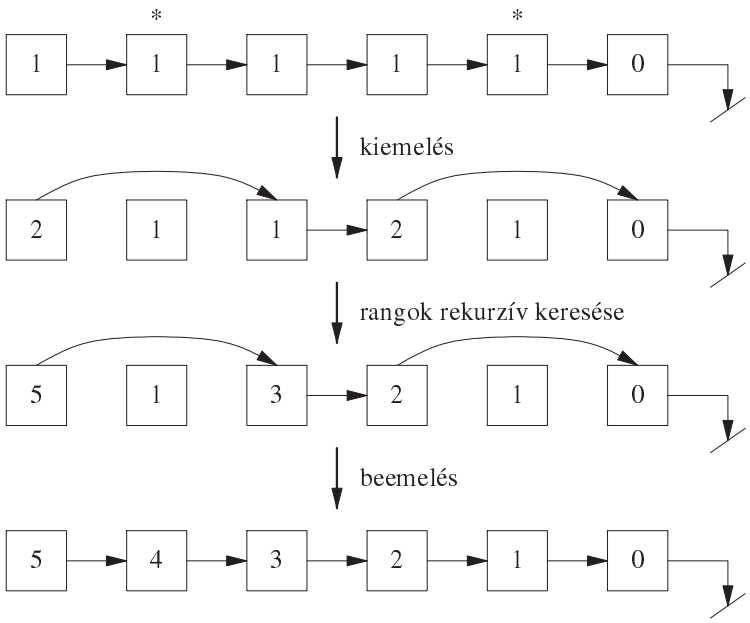
marad, a listarendezési probléma egyszerűen megoldható. A következő menetben a kiemelt csúcsokat ***beemeljük.***

Amikor egy csúcsot beemelünk, akkor a helyes rangját is meghatározzuk. A beemelés sorrendje a kiemelési sorrend fordítottja. A KIEMEL algoritmus a következő.

KIEMEL(p) *párhuzamos eljárás*
Számítási modell: EREW PRAM
Bemenet: p (a rangsorolandó elemek száma) és $A[1:p]$
(a rangsorolandó elemeket tartalmazó tömb)
Kimenet: $rang[1:p]$ (az elemek rangjai)

01 Láncoljuk a listát kettősen. Legyen az $A[i]$ csúcs bal oldali szomszédja $bal\_szomsz[i]$, jobb oldali szomszédja $jobb\_szomsz[i]$.
A csúcsok rang mezőit kezdetben így adjuk meg:
$[1] \to [1] \to [1] \to [1] \to [1] \to [1] \to [0] \Downarrow$.
02 **while** A megmaradó csúcsok száma legalább 3
03 **do** P_i **in parallel for** $i \leftarrow 1$ **to** $p/\lg p$
vizsgálja meg a következő hozzátartozó kiemeletlen csúcsot (legyen ez x). Ezután dobjon fel egy kétoldalú érmét. Ha az eredmény ÍRÁS, akkor P_i maradjon tétlen a menet hátralévő részében. A következő menetben próbálja meg ismét kiemelni x-et. Másrészt, ha a dobás eredménye FEJ, akkor P_i vizsgálja meg, vajon az elem jobb oldali szomszédját vizsgálta-e a megfelelő processzor. Ha a megfelelő processzor vizsgálta és a dobásának eredménye ugyancsak FEJ, akkor P_i adja fel és a menet hátralévő részében maradjon tétlen. Ha nem, akkor P_i emelje ki x-et.
04 Amikor egy x csúcsot kiemelünk, tároljuk a menetszámot, valamint a bal_szomsz mutatót és $rang[bal\_szomsz[x]]$-et. Az utóbbi ebben a pillanatban a bal_szomsz és x közötti csúcsok számát tartalmazza. P_i a $rang[bal\_szomsz[x]] \leftarrow rang[bal\_szomsz[x]] + rang[x]$ értékadást is elvégzi. P_i végül beállítja a $jobb\_szomsz[bal\_szomsz[x]] \leftarrow jobb\_szomsz[x]$ és $bal\_szomsz[jobb\_szomsz[x]] \leftarrow bal\_szomsz[x]$ értékeket.

A kettős láncolás p processzorral $O(1)$ lépéssel elvégezhető. A P_i processzorhoz az $A[i]$ $(1 \le i \le p)$ csúcsot rendeljük. Egy lépésben



A * kiemelt csomópontokat jelöl.

2.4. ábra. A csúcsok kiemelése és beemelése.

P_i a $szomsz[i]$ memóriarekeszbe ír úgy, hogy a következő lépésben az $A[szomsz[i]]$ csúccsal összekapcsolt processzor ismerni fogja bal oldali szomszédját. A lassulási lemma segítségével belátható, hogy $n/\lg n$ processzoron p elem kettős láncolása $\lg p$ lépésben elvégezhető.

A csúcsok beemelése (lásd 2.4. ábra) ugyancsak menetekben történik. Amikor az x csúcsot beemeljük, helyes rangját így határozzuk meg: ha kiemelésekor a $bal\_szomsz[x]$ mutatót tároltuk, akkor x rangját úgy kapjuk, hogy $bal\_szomsz[x]$ aktuális rangjából levonjuk azt a rangot, amit x kiemelésekor tároltunk. A mutatókat ugyancsak frissítjük, figyelembe véve azt a tényt, hogy x-et már beemeltük. Ezt mutatja a **??**. ábra, amelyen a csúcsoknak csak a jobb oldali mutatója szerepel.

Most megmutatjuk, hogy a kiemelések s összes száma $\overline{O}(\lg p)$. Ha egy csúcsot a i-edik menetben kiemelünk, akkor a $(2s-i+1)$. menetben fogjuk beemelni. Így az algoritmus szerkezete a következő. Az $1, 2, \ldots, s$ menetekben egymás után kiemeljük a csúcsokat. Az s menetekben az addigra megmaradt 2 csúcs egyikét kiemeljük. Az $(s+1)$-edik menetekben beemeljük az s-edik menetben kiemelt csúcsot. Az $(s+2)$-edik menetben az $s = 1$ me-

netben kiemelt csúcsot emeljük be és így tovább. Az utolsó menet után az eredeti lista minden csúcsának rangját tudjuk.

Mivel P_i minden menetben csak egy csúcsot vizsgál, ezért a hozzá tartozó csúcsok közül legfeljebb egyet emelünk ki. Az is minden esetben fennáll, hogy a lista szomszédos csúcsait egyszerre nem emeljük ki: ugyanis a FEJ után egy processzor csak akkor próbálja kiemelni a választott csúcsot, ha a jobb oldali szomszéd processzor nem *fej*-et dobott. Ezért a processzorok menetenként csak $O(1)$ lépést igényelnek.

Az algoritmus lépésszámához csak s-et kell meghatározni. Ehhez megbecsüljük a menetenként kiemelt csúcsok számát. Minden P_i processzorra igaz, hogy a kiválasztott x csúcsot legalább 1/4 valószínűséggel kiemeli: ugyanis P_i 1/2 valószínűséggel dob *fej*-et, és legalább 1/2 annak a valószínűsége, hogy x jobb oldali szomszédját (legyen ez a csúcs y) nem választjuk ki, vagy kiválasztjuk ugyan, de y processzora írást dob.

Minden processzor $\lg p$ csúccsal kezdi az algoritmust és minden menetben legalább 1/4 annak valószínűsége, hogy kiemelünk egy csúcsot. Ezért s várható értéke legfeljebb $4 \lg p$. Most alkalmazzuk az (1.51) Csernov-egyenlőtlenséget, amely p darab q-paraméterű Bernoulli-kísérlet esetében minden $0 < \epsilon < 1$ számra igaz. A $12\alpha \lg p$ és 1/2 paraméterekkel az $\epsilon = 2/3$ esetben azt kapjuk, hogy

$$P(s \leq 12\alpha \lg p) > 1 - p^{-\alpha} \tag{2.6}$$

minden $\alpha \geq 1$ értékre. Tehát beláttuk a következő tételt.

2.5. tétel (lista rendezése $\overline{O}(\lg p)$ lépéssel). *A* KIEMEL *algoritmus egy p hosszúságú lista rangsorolását $O(p/\lg p)$ EREW PRAM processzoron $\overline{O}(\lg p)$ lépéssel elvégezi.*

2.2. Kiválasztás

Adott $n \geq 2$ kulcs és egy i ($1 \leq i \leq n$) egész szám. A feladat az i-edik legkisebb kulcs ***kiválasztása.*** Mivel a kiválasztáshoz minden elemet meg kell vizsgálni, ezért $N(n) = \Omega(n)$. Erre a feladatra ismert olyan $\mathcal{A}$ soros algoritmus, amelyikre $W(n, \mathcal{A}) = O(n)$, tehát $\mathcal{A}$ aszimptotikusan optimális.

Ehhez hasonló a ***keresési feladat,*** amelyben azt kell eldönteni, hogy adott elem előfordul-e a vizsgált sorozatban – és ha igen, milyen indexszel. Ennél a feladatnál tagadó válasz is lehetséges és egy elemről tulajdonságai alapján eldönthető, megfelel-e a keresési feladatnak.

Először 3 speciális esetet vizsgálunk, majd egy munkahatékony véletlenített algoritmust ismertetünk.

2.2.1. Kiválasztás n^2 processzoron

Legyen $i = n$, azaz a legnagyobb kulcsot keressük. Ez a feladat a következő NÉGYZETES-KIVÁLASZT algoritmussal n^2 CRCW processzoron $O(1)$ lépéssel elvégezhető.

NÉGYZETES-KIVÁLASZT(K, y) *párhuzamos eljárás*
Számítási modell: CRCW PRAM
Bemenet: $\mathbf{k} = k_1, k_2, \ldots, k_n$ (n különböző kulcs)
Kimenet: y (a maximális kulcs értéke)

01 **if** $n = 1$
02 **then** $y \leftarrow k_1$
03 **return** y
04 P_{ij} **in parallel for** $i \leftarrow 1$ **to** n, $j \leftarrow 1$ **to** n
számítsa ki az $x_{ij} = k_i < k_j$ értéket
05 Az n^2 processzort n csoportba $(G_1, \ldots, G_n)$ osztjuk úgy, hogy a G_i csoportba a $P_{i,1}, \ldots, P_{i,n}$ processzorok kerüljenek. Mindegyik csoport logikai VAGY műveletet végez az $x_{i1}, \ldots, x_{in}$ logikai változókkal.
06 Ha a G_i csoport számítási eredménye az 5. lépésben HAMIS, akkor a csoport P_{i1} processzora megadja $(y = k_i)$-t kimenetként.

Legyenek a bemenő adatok $k_1, \ldots, k_n$. Az összehasonlításokat párhuzamosan végezzük a P_{ij} $(1 \le i, j \le n)$ processzorokon úgy, hogy P_{ij} az $x_{ij} = (k_i < k_j)$ logikai értéket számítja ki. Feltehetjük, hogy a kulcsok különbözőek. Ha mégse, k_i helyett a (k_i, i) párt alkalmazva különbözővé tehetők: ehhez minden kulcshoz egy $(\lg n)$-bites számot kell hozzáadni. Ekkor egyetlen olyan kulcs van, amelyikre minden összehasonlítás eredménye HAMIS. Ez a kulcs egy logikai VAGY művelettel azonosítható.

2.6. tétel (kiválasztás $O(1)$ lépéssel). *n kulcs közül a maximális $O(1)$ lépéssel meghatározható n^2 CRCW közös PRAM processzoron.*

Bizonyítás. Az első és a harmadik szakasz egységnyi ideig tart. A második szakasz $O(1)$ lépéssel elvégezhető. ∎

Ennek az algoritmusnak a relatív sebessége $\Theta(n)$. Az elvégzett munka $\Theta(n^2)$. Ezért a hatékonyság $\Theta(n)/n^2) = \Theta(1/n)$. Tehát az algoritmus nem munkahatékony.

2.2.2. Kiválasztás p processzoron

Most megmutatjuk, hogy a maximális elem p közös CRCW processzoron $O(\lg\lg p)$ lépéssel meghatározható. A technika az *oszd-meg-és-uralkodj*. Az egyszerűség kedvéért feltesszük, hogy p négyzetszám.

Legyenek a bemenő adatok $X = \langle k_1, k_2, \ldots, k_p\rangle$. Legyen az algoritmusunk lépésszáma $T(p)$. A bemenő adatokat $\sqrt{p} = a$ csoportra osztjuk úgy, hogy minden csoportban a elem legyen. Minden csoporthoz rendeljünk a processzort – ekkor a csoportok maximális eleme párhuzamosan számítható. Mivel csoportonként a elem és ugyanannyi processzor van, a csoport maximális eleme $T(a)$ lépéssel meghatározható. Legyenek $M_1, M_2, \ldots, M_a$ a csoportok maximális elemei. Ezek maximuma lesz az algoritmus kimenete. Mivel most csak a elemünk van, az összes processzort alkalmazhatjuk.

A következő CRCW algoritmus $O(\lg\lg p)$ lépést tesz.

GYÖKÖS-KIVÁLASZT(X, p) *párhuzamos rekurzív eljárás*
Számítási modell: CRCW PRAM
Bemenet: $\langle k_1,\ k_2,\ \ldots,\ k_p\rangle$ (p különböző kulcs)
Kimenet: y (a maximális kulcs értéke)

```
01 if p = 1
02    then y ← k_1
03         return y
04 Osszuk a bemenetet a részre (K_1, K_2, ..., K_a) úgy, hogy K_i a
     k_{(i-1)a+1}, k_{(i-1)a+2}, ..., k_{ia} elemeket tartalmazza. Hasonlóképpen
     csoportosítsuk a processzorokat úgy, hogy a P_i (1 ≤ i ≤ a)
     csoportba a P_{(i-1)a+1}, P_{(i-1)a+2}, ..., P_{ia} processzorok tartozzanak.
     A P_i csoport rekurzívan határozza meg a K_i csoport maximális elemét.
05 Ha a csoportok maximális elemei M_1, M_2, ..., M_a, ezek maximumát határozzuk meg
     az előző NÉGYZETES-KIVÁLASZT algoritmussal és ez lesz az eredmény.
```

2.7. tétel (kiválasztás $O(\lg\lg p)$ lépéssel). *A* GYÖKÖS-KIVÁLASZT *algoritmus p közös* CRCW PRAM *processzoron $O(\lg\lg p)$ lépéssel meghatározza p kulcs közül a legnagyobbat.*

Bizonyítás. Ennek az algoritmusnak az első lépése $T(\sqrt{p})$, második lépése $O(1)$ ideig tart. Ezért $T(p)$ kielégíti a

$$T(p) = T(\sqrt{p}) + O(1) \tag{2.7}$$

rekurzív egyenletet, melynek megoldása $O(\lg\lg p)$. ■

A GYÖKÖS-KIVÁLASZT algoritmus összes munkája $\Theta(p \lg\lg p)$, ezért hatékonysága $(\Theta(p)/\Theta(p \lg\lg p) = \Theta(1/\lg\lg p)$, így ez az algoritmus sem munkahatékony.

2.2.3. Kiválasztás egész számok között

Legyen a feladat ismét n kulcs maximumának meghatározása. Ha a kulcsok egyetlen bitből állnak, akkor a maximum keresése visszavezethető a logikai VAGY műveletre és ezért $O(1)$ lépéssel meghatározható. Ebből adódik a kérdés: mekkora intervallumban lehetnek a kulcsok ahhoz, hogy p processzoron konstans lépéssel meg tudjuk határozni a maximális elemet?

Legyen c adott konstans, a kulcsok pedig legyenek a $[0, n^c]$ intervallumban. Ekkor a kulcsok legfeljebb $c \lg n$ bites bináris számok. Az egyszerűség kedvéért feltesszük, hogy pontosan ennyi bitesek (a számok elejére szükség esetén nullákat írunk).

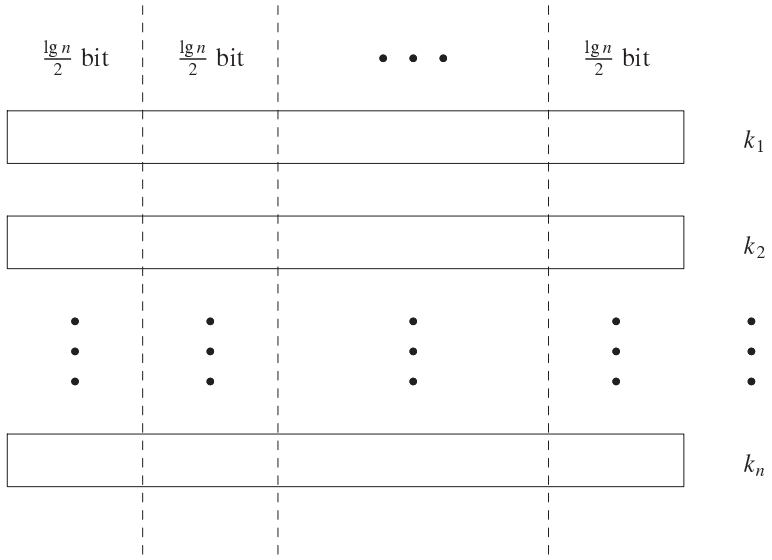
A következő CRCW algoritmus $O(1)$ lépést tesz.

Az alapötlet az, hogy a számok $b_1, b_2, \ldots, b_{2c}$ bitjeit $\frac{\lg n}{2}$ hosszúságú *részekre* bontjuk. Az i-edik rész a $b_{(i-1)+1}, b_{(i-1)+2}, \ldots, b_{(i-1)+b_{(i-1)+(\lg n)/2}}$ biteket tartalmazza, a részek száma $2c$.

Ezt a helyzetet mutatja a 2.5. ábra: először az ábra első oszlopában lévő bitek alapján keressük a maximális kulcsot.

EGÉSZET-KIVÁLASZT(X) *párhuzamos eljárás*
Számítási modell: CRCW PRAM
Bemenet: $X = k_1, k_2, \ldots, k_n$ (n különböző kulcs – egész számok)
Kimenet: y (a maximális kulcs értéke)

```
01 for i ← 1 to 2c
02     do Határozzuk meg a megmaradt kulcsok maximumát i-edik
       részük alapján. Legyen a maximum M.
03     Hagyjuk el azokat a kulcsokat, melyek i-edik része kisebb, mint M.
04 y legyen a megmaradt kulcsok egyike
05 return y
```



2.5. ábra. Maximális egész szám kiválasztása.

2.8. tétel (kiválasztás egész számok közül). *Ha a kulcsok a* $[0, n^c]$ *intervallumból vett egész számok, akkor* p *kulcs közül a maximális* $O(1)$ *lépéssel meghatározható* p CRCW PRAM *processzoron tetszőleges* c *konstans esetében.*

Bizonyítás. Tegyük fel, hogy a kulcsok maximumát a $\frac{\lg n}{2}$ legfontosabb bit alapján határozzuk meg.

Legyen az első részben a maximum M. Ekkor azok a kulcsok, melyek legfontosabb bitjei nem M-et adnak, biztosan nem maximálisak. Ezt az alaplépést megismételjük $2c$-szer, azaz minden $(\lg p)/2$ bitre pontosan egyszer. Legalább egy kulcs megmarad az utolsó lépés után is – az lesz az eredmény. Az utolsó rész lehet rövidebb, mint $(\lg p)/2$ bit.

Ha egy kulcs legfeljebb $(\lg n)/2$ bites, akkor az értéke legfeljebb $\sqrt{n} - 1$. Ezért az EGÉSZET-KIVÁLASZT első lépésében a $[0, \sqrt{n} - 1]$ intervallumba eső egész kulcsok maximumát kell meghatározni. Rendeljünk minden kulcshoz egy processzort és használjunk $\sqrt{n}$ közös memóriarekeszt $(M_1, M_2, \ldots, M_{\sqrt{n}-1})$, melyek tartalma kezdetben $-\infty$. Egy párhuzamos lépésben a P_i processzor k_i-t ír az M_{k_i} memóriarekeszbe. Ezután az n kulcs maximuma a $\sqrt{n}$ memóriarekesz tartalmából n processzorral a 2.9. tétel alapján

konstans idő alatt meghatározható. ■

2.2.4. Az általános kiválasztási feladat megoldása $n^2/\lg n$ processzoron

Tegyük fel, hogy az $X = \langle k_1, k_2, \ldots, k_n \rangle$ sorozat különböző kulcsokat tartalmaz és az i-edik legkisebb kulcsot akarjuk kiválasztani. Legyen most az x_i ***kulcs rangja*** eggyel nagyobb, mint a nála kisebb kulcsok száma (ez a definíció eggyel nagyobb értéket ad, mint a korábban használt).

Ezt a rangot a 2-5. gyakorlat szerint $\frac{n}{\log n}$ CREW PRAM processzoron bármely kulcsra $O(\lg n)$ lépésben meg tudjuk határozni.

Ha $n^2/\lg n$ processzorunk van, akkor azokat $C_1, C_2, \ldots, C_n$ csoportokba oszthatjuk úgy, hogy minden csoportban $n/\lg n$ processzor legyen. A C_j $(1 \leq j \leq n)$ csoport $O(\lg n)$ lépésben meghatározza a k_j kulcs rangját X-ben. Annak a csoportnak egyik processzora, amelyik az i rangot határozta meg, adja a kimenetet. Az így kapott algoritmus neve legyen Ált-kiválaszt.

2.9. tétel (általános kiválasztás). *Az* Ált-kiválaszt *algoritmus $n^2/\lg n$ processzoron n különböző kulcs közül $\Theta(\lg n)$ lépésben meghatározza az i-edik legkisebbet.*

Nem nehéz belátni, hogy az Ált-kiválaszt algoritmus munkája $\Theta(n^2)$, tehát ez az algoritmus sem munkahatékony.

2.2.5. Munkaoptimális véletlenített algoritmus (⋆)

Ebben a pontban $n/\log n$ közös CRCW processzort alkalmazunk arra, hogy $\overline{O}(\lg n)$ lépésben megoldjuk az i-edik legkisebb elem kiválasztását.

A Vél-kiválaszt algoritmus a bemenő kulcsok $X = k_1, k_2, \ldots, k_n$ sorozatából kiválaszt egy $n^{1-\epsilon}$ méretű S mintát és S két elemét elválasztó elemként. Például $\epsilon = 0.6$ megfelelő érték.

Legyen e_1 és e_2 a két elválasztó elem. Az elválasztó elemek olyanok lesznek, hogy a kiválasztandó elem nagy valószínűséggel a két elválasztó elem közé fog esni. Továbbá X-nek az elválasztó elemek közé kevés eleme esik: $\overline{O}(n^{(1+\epsilon)/2}\sqrt{n})$.

Ha már kiválasztottuk a két elválasztó elemet, X-et az $X_1 = \{x \in X | x < e_1\}$, $X_2 = \{x \in X | e_1 \leq x \leq e_2\}$ és $X_3 = \{x \in X | x > e_2\}$ diszjunkt részekre bontjuk. A felbontás során meghatározzuk az egyes részek elemszámát. Ha $|X_1| < i \leq |X_1| + |X_2|$, akkor a kiválasztandó elem X_2 eleme. Ebben az esetben tovább megyünk, míg ellenkező esetben újra kezdjük.

A mintavételezési és kiküszöbölési műveleteket addig ismételjük, amíg el nem érjük, hogy a megmaradó kulcsok száma legfeljebb $n^{0.4}$ legyen. Ezután a megmaradó kulcsok közül az ÁLT-KIVÁLASZT algoritmus segítségével végezzük el a kiválasztást.

Az algoritmus pszeudokódja a következő. A pszeudokódban a N munkaváltozó az élő kulcsok számát adja meg. Kezdetben legyen $N = n$ és minden kulcs legyen ***élő.*** Minden processzorra $\lg n$ kulcs jut. A 3. és 6. lépésekben a koncentráció azt jelenti, hogy a megfelelő kulcsokat összegyűjtjük és a közös memória egymást követő rekeszeiben helyezzük el őket.

VÉL-KIVÁLASZT(X, i, y, n) *párhuzamos eljárás*
Számítási modell: közös CRCW PRAM
Bemenet: $X = k_1,\ k_2,\ \ldots,\ k_p$ (p különböző kulcs)
Kimenet: y (az i-edik legkisebb kulcs értéke)

01 $N \leftarrow n$
02 $\epsilon \leftarrow 0.6$
03 **while** $N > n^{1-\epsilon}$
04 Minden élő kulcs $\frac{1}{N^\epsilon}$ valószínűséggel kerül az M mintába.
05 P_j **in parallel for** $j \leftarrow 1$ **to** $\frac{n}{\lg n}$
P_j meghatározza a minta q méretét és azt minden processzorhoz eljuttatja. Ha nem teljesül $0.5N^{1-\epsilon} \le q \le 1.5N^{1-\epsilon}$, akkor folytatja a 01-es lépésnél.
06 Koncentráljuk és rendezzük a mintában lévő kulcsokat.
07 Legyen e_1 az M minta $\lfloor \frac{iq}{N} \rfloor - d\sqrt{q \lg N}$ és e_2 a minta $\lfloor \frac{iq}{N} \rfloor + d\sqrt{q \lg N}$ rangú eleme, ahol d egy $\sqrt{3\alpha}$-nál nagyobb állandó. Szórjuk e_1 és e_2 értékét minden processzorhoz.
08 Számoljuk meg a $[e_1, e_2]$ intervallumba eső élő kulcsok I, valamint az e_1-nél kisebb kulcsok K számát. Minden processzorhoz juttassuk el ezt a két számot. Ha $i \notin (I, I + K)$ vagy $I \ne (N^{(1+\epsilon)/2}\sqrt{N}$, akkor folytassuk az 1-es lépéssel – egyébként hagyjuk el az e_1-nél kisebb és az e_2-nél nagyobb kulcsokat és legyen $i \leftarrow i - K$ és $N \leftarrow I$.
09 Koncentráljuk és rendezzük a mintában lévő kulcsokat. Határozzuk meg y értékét.

Nevezzük *menetnek* a **while** ciklus egyszeri lefutását. A minták száma minden menetben N és $N^{-\epsilon}$ paraméterekkel rendelkező binomiális eloszlású. Ezért a mintákban lévő kulcsok számának várható értéke $N^{-\epsilon}$. A Csernov-

egyenlőtlenség segítségével belátható, hogy

$$|M| = \overline{O}(N^{1-\epsilon}) \ . \tag{2.8}$$

Legyen M egy m elemű minta, amelyet egy n elemű X halmazból állítottunk elő. Legyen *kiválaszt*(j, M) az M minta j-edik legkisebb eleme és $r_j = rang(kiválaszt(j, M), X)$. Bizonyítás nélkül említjük a következő lemmát.

2.10. lemma. *Minden α számra*

$$P\left(|r_j - j\frac{n}{m}| > \sqrt{3\alpha}\frac{n}{\sqrt{m}\sqrt{\lg n}}\right) < n^{-\alpha} \ . \tag{2.9}$$

Ennek a lemmának a segítségével bizonyítható a következő állítás.

2.11. tétel. *A* Vél-kiválaszt *algoritmus $n/\lg n$ processzoron $\overline{O}(\lg n)$ lépésben kiválasztja n különböző kulcs közül az i-edik legkisebbet.*

Ebből a tételből és a kiválasztás lineáris lépésigényéből következik, hogy a Vél-kiválaszt algoritmus nagy valószínűséggel munkahatékony.

2.3. Összefésülés

Adott 2 csökkenőleg (vagy növekvőleg) rendezett sorozat, melyek együtt p elemet tartalmaznak. A feladat ennek a sorozatnak egy csökkenő (vagy növekvő) sorozattá való rendezése.

Ez a feladat egy soros processzoron $O(p)$ lépéssel megoldható. Mivel legrosszabb esetben minden elemet meg kell vizsgálni és a helyére kell tenni, ezért a feladat megoldásának lépésszámigénye $\Omega(p)$.

2.3.1. Logaritmikus idejű algoritmus

Legyen $X_1 = \langle k_1, k_2, \ldots, k_m \rangle$ és $X_2 = \langle k_{m+1}, k_{m+2}, \ldots, k_{2m} \rangle$ a két bemenő sorozat. Az egyszerűség kedvéért legyen m 2 hatványa és a kulcsok különbözzenek.

Az összefésüléshez elég az összes kulcs rangjának kiszámítása. Ha a rangokat ismerjük, akkor $p = 2m$ processzoron egy lépésben beírhatjuk az i rangú kulcsot az i-edik memóriarekeszbe.

2.12. tétel. *A* Log-összefésül *algoritmus két m hosszúságú kulcssorozatot $\Theta(\lg m)$ idő alatt fésül össze $2m$ CREW PRAM processzoron.*

Bizonyítás. A k kulcs rangja legyen r_k^1 (r_k^2) X_1-ben (X_2-ben). Ha $k = k_j \in X_1$, akkor legyen $r_k^1 = j$. Ha egy külön π processzort rendelünk k-hoz, akkor az bináris kiválasztással $\Theta(\lg m)$ lépéssel meghatározza azon X_2-beli elemek q számát, amelyek kisebbek, mint k. Ha q ismert, akkor π kiszámíthatja k ($X_1 \cup X_2$)-beli rangját: ez $j+q$ lesz. Ha k X_2-höz tartozik, hasonlóképpen járhatunk el.

Összegezve: ha elemenként egy, azaz összesen $2m$ processzorunk van, akkor két m hosszúságú rendezett sorozat $O(\lg m)$ lépéssel összefésülhető. Az ezt megoldó algoritmus neve LOG-ÖSSZEFÉSÜL. ■

Ez az algoritmus nem munkaoptimális, de munkahatékony.

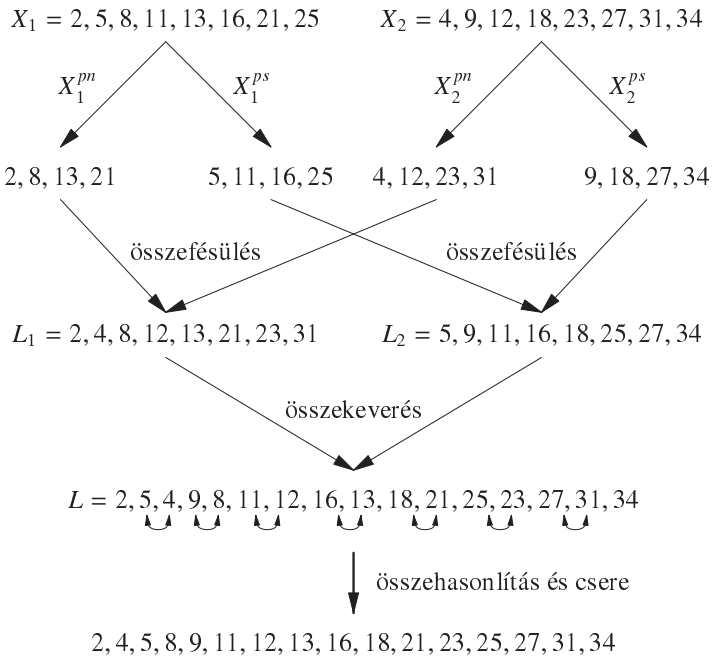
2.3.2. Páratlan-páros összefésülő algoritmus

Ez az algoritmus a klasszikus *oszd meg és uralkodj* elvet alkalmazza.

Legyen $X_1 = k_1, k_2, \ldots, k_m$ és $X_2 = k_{m+1}, k_{m+2}, \ldots, k_{2m}$ a két bemenő sorozat. Az egyszerűség kedvéért legyen m kettő hatványa és a kulcsok legyenek különbözőek.

PÁRATLAN-PÁROS-ÖSSZEFÉSÜL(X_1, X_2) *párhuzamos rekurzív eljárás*
Számítási modell: EREW PRAM
Bemenet: X_1 és X_2 (két rendezett sorozat)
Kimenet: Y (az összefésült sorozat)

01 **if** $m = 1$
then fésüljük össze a sorozatokat egyetlen összehasonlítással
02 **return** Y
03 **if** $m > 1$
04 **then** Bontsuk fel X_1-et és X_2-t páros és páratlan részre, azaz legyen
$X_1^{ptn} = k_1, k_3, \ldots, k_{m-1}$ és $X_1^{prs} = k_2, k_4, \ldots, k_m$.
Hasonlóképpen bontsuk fel X_2-t is X_2^{ptn} és X_2^{prs} részekre.
05 Rekurzívan fésüljük össze X_1^{ptn}-t és X_2^{ptn}-t m processzoron. Legyen az eredmény $L_1 = l_1, l_2, \ldots, l_m$. Ugyanakkor fésüljük össze X_1^{prs}-t és X_2^{prs}-t
másik m processzoron: az eredmény legyen $L_2 = l_{m+1}, l_{m+2}, \ldots, l_{2m}$.
06 Fésüljük össze az L_1, L_2 sorozatokat, azaz legyen
$L = l_1, l_{m+1}, l_2, l_{m+2}, \ldots, l_m, l_{2m}$.
07 Hasonlítsuk össze az $(l_{m+i},\ l_{i+1})$ $(i = 1, 2, \ldots, m-1)$ párokat és szükség esetén cseréljük fel őket. Az eredmény lesz a kimenő sorozat.



2.6. ábra. 16 szám rendezése a PÁRATLAN-PÁROS-ÖSSZEFÉSÜL algoritmussal.

2.5. példa. *Kétszer nyolc szám összefésülése.* Legyen $X_1 = 2, 5, 8, 11, 13, 16, 21, 25$ és $X_2 = 4, 9, 12, 18, 23, 27, 31, 34$. A 16 szám rendezését mutatja a következő 2.6. ábra. ♠

Az összefésülő algoritmusok helyessége a ***nulla-egy elv*** segítségével bizonyítható.

Egy összehasonlítás alapú rendező algoritmus ***egyszerű,*** ha az összehasonlítandó elemek sorozata előre meg van határozva (ekkor a következő összehasonlítás elemei nem függnek a mostani eredménytől).

Formálisan ez azt jelenti, hogy adott az összehasonlítandó elempárok indexeinek (i_1, j_1), (i_2, j_2), $\ldots$, (i_m, j_m) sorozata.

2.13. tétel (nulla-egy elv). *Ha egy egyszerű összehasonlításos rendező algoritmus helyesen rendez egy n hosszúságú nulla-egy sorozatot, akkor tetszőleges kulcsokból álló n hosszúságú sorozatot is helyesen rendez.*

Bizonyítás. Legyen $\mathcal{A}$ egy egyszerű összehasonlításos (növekvőleg) ren-

dező algoritmus és legyen S egy olyan kulcssorozat, melyet az adott algoritmus rosszul rendez. Ekkor a rosszul rendezett S sorozatban van olyan kulcs, amely az i-edik ($1 \leq i \leq n-1$) helyen van annak ellenére, hogy S-ben legalább i nála kisebb elem van.

Legyen k S legelső (legkisebb indexű) ilyen kulcsa. A bemenő sorozatban írjunk a k-nál kisebb elemek helyére nullát, a többi elem helyére egyest. Ezt a módosított 0-1 sorozatot $\mathcal{A}$ helyesen rendezi, ezért a k helyére írt egyest a rendezett sorozatban legalább i darab nulla megelőzi.

Most kihasználjuk, hogy $\mathcal{A}$ egyszerű. A bemenő sorozatban színezzük pirosra a k-nál kisebb (nulla) elemeket, és kékre a többit (egyeseket). Indukcióval megmutatjuk, hogy az eredeti és a 0-1 sorozatnak megfelelő színes sorozatok minden összehasonlítás után azonosak. A színek szerint háromféle összehasonlítás van: kék, piros vagy különböző színű elemek összehasonlítása. Ha azonos színű elemeket hasonlítunk össze, akkor a színek sorozata egyik esetben sem változik. Ha viszont különböző színű elemeket hasonlítunk össze, akkor mindkét esetben a piros elem kerül a kisebb, és a kék elem a nagyobb indexű helyre. Eszerint k-t legalább i nála kisebb elem megelőzi a rendezett sorozatban. Az ellentmondás az állítás helyességét mutatja. ■

2.6. példa. *Egy nem összehasonlításos rendező algoritmus.* Legyen $\langle k_1, k_2, \ldots, k_n \rangle$ egy bitsorozat. Rendezhetjük úgy, hogy megszámoljuk a nullák z számát, majd leírunk előbb z nullát, majd $n-z$ egyest. Erre az elv nem alkalmazható, mert ez nem összehasonlításos rendezés. ♠

Az összefésülés viszont rendezés, és a páros-páratlan összefésülés egyszerű.

2.14. tétel. *A* Páratlan-páros-összefésül *algoritmus helyesen rendez tetszőleges számokból álló sorozatokat.*

Bizonyítás. Legyenek X_1 és X_2 rendezett 0-1 sorozatok, melyek közös hossza m. Legyen q_1 (q_2) az X_1 (X_2) elején álló nullák száma. Az X_1^{ptn}-ban lévő nullák száma $\lceil q_1/2 \rceil$, és az X_1^{prs}-ban lévő nullák száma $\lfloor q_1/2 \rfloor$. Így az L_1-beli nullák száma $z_1 = \lceil q_1/2 \rceil + \lceil q_2/2 \rceil$ és az L_2-beli nullák száma $z_2 = \lfloor q_1/2 \rfloor + \lfloor q_2/2 \rfloor$.

z_1 és z_2 különbsége legfeljebb 2. Ez a különbség pontosan akkor kettő, ha q_1 és q_2 is páratlan. Egyébként a különbség legfeljebb 1. Tegyük fel, hogy $|z_1 - z_2| = 2$ (a többi eset hasonló). Most L_1-ben kettővel több nulla van.

Amikor ezeket a harmadik lépésben összekeverjük, akkor L elején nullák vannak, azután 1,0, majd egyesek. A rendezetlen (*piszkos*) rész csak az 1,0. Amikor a harmadik lépésben az utolsó összehasonlítás és csere megtörténik, az egész sorozat rendezetté válik. ■

2.15. tétel (összefésülés $O(\lg m)$ lépéssel). *A* Páratlan-páros-összefésül *algoritmus két m hosszúságú kulcssorozatot $O(\lg m)$ lépéssel összefésül $2m$* EREW PRAM *processzoron.*

Bizonyítás. Legyen az algoritmus lépésszáma $M(n)$. Az 1. lépés $O(1)$ ideig tart. A 2. lépés $\frac{m}{2}$ ideig tart. Innen az

$$M(m) = M\left(\frac{m}{2}\right) + O(1) \tag{2.10}$$

rekurzív egyenlőséget kapjuk, melynek megoldása $M(m) = O(\lg m)$. ■

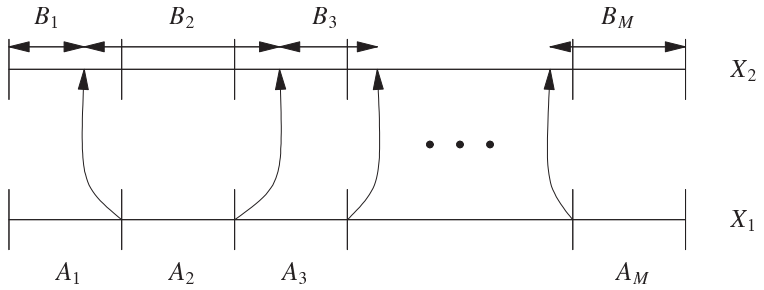
2.3.3. Munkaoptimális algoritmus

Most $\lceil 2m/\lg m \rceil$ processzoron $O(\lg m)$ lépéssel végezzük az összefésülést. Ez az Optimálisan-összefésül algoritmus az eredeti problémát $O(m/\lg m)$ részre osztja úgy, hogy mindegyikben $O(\lg m)$ hosszúságú rendezett sorozatokat kell összefésülni. Ezek a részproblémák soros algoritmussal $O(\lg m)$ lépéssel megoldhatók.

Legyen $X_1 = \langle x_1, x_2, \ldots, x_m \rangle$ és $X_2 = \langle x_{m+1}, x_{m+2}, \ldots, x_{m+m} \rangle$ a két bemenő sorozat. Osszuk X_1-et $\lceil \frac{m}{\lg m} \rceil$ részre: ekkor mindegyikben legfeljebb $\lceil \lg m \rceil$ kulcs lesz. A részek legyenek $A_1, A_2, \ldots, A_M$, ahol $M = \frac{m}{\lg m}$. Az A_i-beli legnagyobb kulcs legyen l_i ($i = 1, 2, \ldots, M$). Rendeljünk egy-egy processzort ezekhez az l_i elemekhez. Ezek a processzorok bináris kiválasztással meghatározzák l_i X_2-beli (rendezés szerinti) helyét. Ezek a helyek felbontják X_2-t M részre (ezek között üres részek is lehetnek – lásd a következő 2.7. ábrát). Jelöljük ezeket a részeket B_1, B_2, …, B_M-mel. B_i-t az A_i-nek X_2-ben megfelelő részhalmaznak nevezzük.

Ekkor X_1 és X_2 összefésülését megkaphatjuk úgy, hogy rendre összefésüljük A_1-et B_1-gyel, A_2-t B_2-vel és így tovább, majd ezeket a sorozatokat egyesítjük.

2.16. tétel. *Az* Optimálisan-összefésül *két m hosszúságú rendezett kulcssorozatot $O(\lg m)$ lépésben összefésül $\lceil \frac{2m}{\lg m} \rceil$* CREW PRAM *processzoron.*



2.7. ábra. Munkaoptimális összefésülő algoritmus.

Bizonyítás. Az előző algoritmust alkalmazzuk. Az A_i részek hossza $\lg m$, a B_i részek hossza azonban nagy is lehet. Ezért még egyszer alkalmazzuk a felbontást. Legyen A_i, B_i tetszőleges pár. Ha $|B_i| = O(\lg m)$, akkor A_i és B_i egy processzoron $O(\lg m)$ lépésben alatt összefésülhető. Ha viszont $|B_i| = \omega(\lg m)$, akkor osszuk B_i-t $\frac{|B_i|}{\lg m}$ részre – ekkor minden rész legfeljebb $\lg m$ egymást követő kulcsot tartalmaz. Mindegyik részhez rendeljünk egy processzort, és az keresse meg az ennek a sorozatnak megfelelő részhalmazt A_i-ben: ehhez $O(\lg \lg m)$ lépés elegendő. Így A_i és B_i összefésülése $\frac{|B_i|}{\lg m}$ részproblémára redukálható, ahol minden részprobléma két $O(\lg m)$ hosszúságú sorozat összefésülése.

A felhasznált processzorok száma $\sum_{i=1}^{M} \lceil |B_i| / \lg m \rceil$,, ami legfeljebb $m/\lg m + M$, és ez legfeljebb $2M$. ∎

2.3.4. Egy $O(\lg \lg m)$ idejű algoritmus

Ha az előző algoritmust kiegészítjük az *oszd meg és uralkodj* elvvel, akkor még gyorsabb algoritmust kapunk.

GYORSAN-ÖSSZEFÉSÜL(X_1, X_2, Y) *párhuzamos eljárás*
Számítási modell: CREW PRAM
Bemenet: X_1 és X_2
Kimenet: Y

01 Bontsuk fel X_1-et $\sqrt{m} = b$ részre: ekkor minden részben b elem lesz. Legyen A_i-ben a legnagyobb kulcs l_i $(i = 1, 2, \ldots, b)$. Minden l_i-hez rendeljünk b processzort. Ezek a processzorok b-áris keresést végeznek X_2-ben, hogy megtalálják l_i X_2-beli helyét. Ezzel X_2 b részre való felbontását kapjuk: legyenek ezek a részek

$B_1, B_2, \ldots, B_b$. A B_i részhalmaz az A_i-nek X_2-ben megfelelő részhalmaz.

02 Most X_1 és X_2 összefésüléséhez elegendő A_i és B_i $(i = 1, 2, \ldots, b)$ összefésülése. Az A_i-k mérete adott, viszont a B_i-k nagyon nagyok (és nagyon kicsik) is lehetnek. Ezért újra felbontunk.

Legyen A_i és B_i tetszőleges pár. Ha $|B_i| = O(b)$, akkor a két sorozat $O(1)$ lépésben összefésülhető $\sqrt{m}$-áris kereséssel (ahol ϵ tetszőleges pozitív szám). Ha viszont $|B_i| = \omega(b)$, akkor B_i-t $\left\lceil \frac{|B_i|}{b} \right\rceil$ részre osztjuk, ahol B_i-nek minden részben legfeljebb b egymást követő eleme van. Rendeljünk minden részhez b processzort, hogy megtalálják az ehhez a halmazhoz tartozó részhalmazt A_i-ben: ehhez $O(1)$ lépés elég. Így A_i és B_i összefésülésének problémája $\left\lceil \frac{|B_i|}{b} \right\rceil$ részproblémára redukálható, ahol minden részprobléma két $O(b)$ hosszúságú sorozat összefésülése.

A felhasznált processzorok száma $\sum_{i=1}^{b} b\lceil |B_i|/b \rceil$, ami legfeljebb $2m$.

2.17. tétel (összefésülés $O(\lg \lg m)$ lépésben). *Két m hosszúságú rendezett kulcssorozat $O(\lg \lg m)$ lépésben összefésülhető $2m$* CREW PRAM *processzoron.*

Bizonyítás. Legyen X_1 és X_2 a két adott sorozat. Legyenek a kulcsok különbözők és legyen $\sqrt{m} = b$. Az algoritmus a feladatot $N \leq 2b$ részfeladatra redukálja, melyek mindegyike két $O(b)$ hosszúságú rendezett sorozat összefésülése. A redukció m processzoron $O(1)$ lépésben elvégezhető. Ha az algoritmus lépésszáma $2m$ processzoron $T(m)$, akkor $T(m)$ kielégíti a

$$T(m) = T(O(b)) + O(1)$$

rekurzív egyenletet, melynek megoldása $O(\lg \lg m)$. ∎

Ez az algoritmus nem munkaoptimális, bár $\Theta(m)/\Theta(\lg \lg m) = \Theta(m/\lg \lg m)$ gyorsítása nagyon közel van m-hez. Hatékonysága csak $\Theta(1/\lg \lg m)$.

2.4. Rendezés

Adott $n \geq 2$ kulcs. A feladat ezek csökkenő vagy növekvő sorrendbe való rendezése.

Ismert, hogy ha a megengedett művelet a szokásos összehasonlítás, akkor

minden $\mathcal{A}$ soros algoritmusnak $N(n,\mathcal{A}) = \Omega(n \lg n)$ lépésre van szüksége, másrészt vannak $O(n \lg n)$ lépésszámú összehasonlítás alapú algoritmusok, amelyek tehát aszimptotikusan optimálisak.

Más műveletek vagy a rendezendő kulcsok speciális tulajdonságai esetében a rendezés $O(n)$ lépéssel is megoldható. Ha legrosszabb esetben minden kulcsot meg kell vizsgálni, akkor természetesen a lépésszám $N(n) = \Omega(n)$.

Tehát mind az összehasonlítás alapú, mind pedig a speciális esetben ismert aszimptotikusan optimális soros algoritmus.

Vizsgáljuk meg a következő kérdéseket. Hány rendező algoritmus van? Ezek közül hány egyszerű, hány optimális (aszimptotikusan, szigorúan)?

Ezekre a kérdésekre nem könnyű válaszolni – például először pontosan definiálnunk kell, mi is az a rendező algoritmus.

Szűkítsük a kérdést: hány összehasonlításra van szükség n elem rendezéséhez? jelöljük ezt a számot $c(n)$-nel. Ismert, hogy

$$\left\lceil \sum_{i=1}^{n} \lg i \right\rceil \le c(n) \le \sum_{i=1}^{n} \lceil \lg i \rceil \tag{2.11}$$

és hogy

$$c(n) \le n \lg n - (n-1)\ . \tag{2.12}$$

Az alsó becslés döntési fákkal vagy információelméleti eszközökkel igazolható (lásd az alsó korlátokról szóló 1.7. alfejezetet), a felső becslések pedig a bináris beszúró, illetve az összefésüléses rendező jellemző adatai.

4 elemre az alsó és a felső becslések egyaránt ötöt adnak. 5 elemre $5! = 120$ miatt az alsó becslés 7, viszont az előbbi algoritmusoknak 8 összehasonlításra van szüksége.

n^2 processzoron a kulcsok rangja $O(\lg n)$ lépéssel meghatározható. Ha a rangokat ismerjük, akkor a rendezés egy párhuzamos írással megoldható.

Tehát igaz a következő tétel.

2.18. tétel (rendezés $O(\lg n)$ lépésben). *n kulcs n^2 CRCW PRAM processzoron rendezhető $O(\lg n)$ lépéssel.*

Mivel a kulcsok meghatározása $\Omega(\lg n)$ ideig tart, ez a módszer $\Theta(n^2 \lg n)$ munkát igényel, azaz nem munkahatékony.

2.4.1. Páratlan-páros algoritmus

Ez az algoritmus a klasszikus *oszd meg és uralkodj* elvet alkalmazza. Az egyszerűség kedvéért legyen n kettő hatványa és a kulcsok legyenek különbözők.

A következő EREW PRAM algoritmus $O(\lg^2 n)$ lépést tesz.

PÁRATLAN-PÁROS-RENDEZ(X, Y) *párhuzamos rekurzív eljárás*
Számítási modell: EREW PRAM
Bemenet: X (a rendezendő kulcsok)
Kimenet: Y (a rendezett kulcsok)

```
01 if n = 1
02       then Y ← X
03            return X
04 if n > 1
05       then Osszuk az X bemenetet két részre: ezek legyenek
              X'_1 = k_1, k_2, ..., k_{n/2} és X'_2 = k_{n/2+1}, k_{n/2+2}, ..., k_n.
06            Rendezze n/2 processzor rekurzívan X'_1-t. Az eredmény legyen X_1.
              Ugyanakkor rendezze n/2 processzor rekurzívan X'_2-t. Az eredmény legyen X_2.
07            Fésüljük össze X_1-et és X_2-t 2m processzoron
              a PÁRATLAN-PÁROS-ÖSSZEFÉSÜL algoritmussal.
```

Ez az algoritmus hasonlít a soros összefésüléses algoritmusra. Ott azonban a sorozatok legkisebb elemeit hasonlítjuk össze és a kisebb elem az összehasonlítás eredménye.

2.19. tétel (rendezés $O(\lg^2 n)$ lépéssel). *n kulcs n EREW PRAM processzoron rendezhető $O(\lg^2 n)$ lépéssel.*

Bizonyítás. Legyen $T(n)$ az algoritmus lépésszáma. Az 1. lépés $O(1)$ ideig tart, a 2. lépés $T(\frac{n}{2})$ ideig, a 3. lépés pedig $O(\lg n)$ ideig. Ezért $T(n)$ kielégíti a

$$T(n) = O(1) + T\left(\frac{n}{2}\right) + O(\lg n) \tag{2.13}$$

rekurzív egyenlőséget, melynek megoldása $T(n) = O(\lg^2 n)$. ■

2.7. példa. *Rendezés 16 processzorral.* Rendezzük 16 processzoron a következő számokat: 25, 21, 8, 5, 2, 13, 11, 16, 23, 31, 9, 4, 18, 12, 27, 34. Az első lépésben a páros és páratlan részeket kapjuk meg, majd a másodikban az első 8 processzor a páratlan részből kapja az $X_1 = \langle 2, 5, 8, 11, 13, 16, 21, 25\rangle$-öt, a második 8 processzor pedig az $X_2 = \langle 4, 9, 12, 18, 23, 27, 31, 34\rangle$-et. A

harmadik lépésben kapjuk az eredményt. ♠

Ez az algoritmus $\Theta(n \lg^2 n)$ munkát végez. Hatékonysága $\Theta(\frac{1}{\lg n})$, gyorsítása $\Theta\left(\frac{n}{\lg n}\right)$.

2.4.2. Egy véletlenített algoritmus (⋆)

Az $O(\lg^2 n)$ lépésszám véletlenített algoritmussal is elérhető.

2.20. tétel (rendezés $\overline{O}(\lg^2 n)$ lépéssel). *n kulcs n CREW PRAM processzoron rendezhető $\overline{O}(\lg^2 n)$ lépéssel.*

Bizonyítás. Ismert, hogy n elem közül $n/\lg n$ processzoron a kiválasztás $\overline{O}(\lg n)$ lépéssel megoldható. Legyen n processzorunk. Ekkor n adott kulcs k mediánja $\overline{O}(\lg n)$ lépéssel megtalálható. Osszuk X-et 2 részre: X_1' tartalmazza a k-nál nem nagyobb kulcsokat, X_2' pedig a többi kulcsot.

Az X_1' és X_2' részeket rekurzívan rendezzük $n/2$ processzoron, majd az eredményeket láncoljuk össze.

Ha a rendezés ideje $T(n)$, akkor

$$T(n) = T\left(\frac{n}{2}\right) + \overline{O}(\lg n)\ , \tag{2.14}$$

melynek megoldása $T(n) = \overline{O}(\lg^2 n)$. ■

2.4.3. Preparata algoritmusa

Több processzorral a lépésszám csökkenthető: Preparata algoritmusa $n \lg n$ CREW PRAM processzoron $\lg n$ párhuzamos lépést végez.

PREPARATA(X, Y) *párhuzamos rekurzív eljárás*
Számítási modell: CREW PRAM
Bemenet: X (rendezendő kulcsok)
Kimenet: Y (rendezett kulcsok)

```
01 if n ≤ 20
02    then rendezzük a bemenetet tetszőleges rendező algoritmussal
03         return Y
04 Osszuk az adott n kulcsot lg n részre úgy, hogy mindegyikben n/lg n kulcs legyen.
```

Rendezzük a részeket külön, rekurzívan, mindegyik részhez n processzort rendelve. A rendezett sorozatok legyenek $S_1, S_2, \ldots, S_{\lg n}$.

05 Fésüljük össze S_i-t és S_j-t $(1 \le i, j \le \lg n)$ párhuzamosan.

06 Rendeljünk $\lg n$ processzort a kulcsok eredeti sorozatra vonatkozó rangjának meghatározásához. Végül a kulcsok a rangok sorrendjében adják a kimenetet.

Ez az algoritmus *oszd meg és uralkodj* elvű. A kulcssorozatot $\lg n$ részre osztjuk, majd a részeket páronként összefésülve minden kulcsnak minden részre nézve meghatározzuk a rangját. Ezután a kulcsok tényleges rangja az előbbi rangok összege.

Ha a harmadik lépésben minden (i, j) párhoz $n/\lg n$ processzort rendelünk, akkor az összefésülés $O(\lg\lg n)$ lépéssel elvégezhető.

A negyedik lépésben a rang számítása párhuzamosan végezhető a harmadik lépésben kapott $\lg n$ rang összeadásával: ez a prefixet számító algoritmussal $O(\lg\lg n)$ lépéssel megoldható.

2.21. tétel (rendezés $O(\lg n)$ lépéssel). *A* Preparata *algoritmus n elemet $n \lg n$ CREW PRAM processzoron $O(\lg n)$ lépéssel rendez.*

Bizonyítás. Legyen a Preparata-algoritmus futási ideje $T(n)$. Az első lépés időigénye $T(n/\lg n)$, a második és harmadik lépésé együtt $O(\lg\lg n)$. Ezért

$$T(n) = T\left(\frac{n}{\lg n}\right) + O(\lg\lg n)\ . \tag{2.15}$$

melynek megoldása (helyettesítéses módszerrel) $T(n) = O(\lg n)$. ∎

A lassulásra vonatkozó tétel segítségével kapjuk a következő állítást.

2.22. következmény (rendezés $\frac{n\lg n}{t}$ processzoron). *Tetszőleges $t \ge 1$ egész számra n tetszőleges kulcs rendezhető $O(t \lg n)$ lépésben $\frac{n\lg n}{t}$ CREW PRAM processzoron.*

A Preparata-algoritmus munkája ugyanannyi, mint a páros-páratlan rendező algoritmusé, viszont a gyorsítása jobb: $\Theta(n)$. Mindkét algoritmus hatékonysága $\Theta(1/\lg n)$.

2.4.4. Reischuk véletlenített algoritmusa (⋆)

Reischuk algoritmusa n processzoron $\overline{O}(\lg n)$ párhuzamos lépést tesz, azaz nagy valószínűséggel munkahatékony.

Alapja Preparata algoritmusa és a következő tétel.

2.23. tétel (korlátos egészek rendezése). *Ha n kulcs mindegyike $[0,\ n(\lg n)^c]$ intervallumból vett egész szám (ahol c tetszőleges konstans), akkor ezek a kulcsok $\frac{n}{\lg n}$ CREW PRAM processzoron rendezhetők $\overline{O}(\lg n)$ lépésben.*

Az algoritmus a következő.

REISCHUK(X) *párhuzamos eljárás*
Számítási modell: CREW PRAM
Bemenet: X (rendezendő kulcsok)
Kimenet: Y (rendezett kulcsok)

01 Legyen $X = k_1, k_2, \ldots, k_n$ a rendezendő kulcssorozat.
$N = \frac{n}{\lg^4 n}$ processzor mindegyike véletlenül kiválaszt egy kulcsot.
02 Rendezzük az N kiválasztott kulcsot Preparata algoritmusával.
Az eredmény legyen $l_1, l_2, \ldots, l_N$.
03 Legyen $K_1 = \{k \in X | k \le l_1\}$;
$K_i = \{k \in X | l_{i-1} < k \le l_i\}$, $i = 2, 3, \ldots, N$;
és $K_{N+1} = \{k \in X | k > l_N\}$. Párhuzamos bináris kiválasztással
bontsuk fel X-et K_i részekre.
04 Rendezzük a K_i részeket a PREPARATA algoritmussal.
05 Legyen Y a rendezett K_1, rendezett K_2, …, rendezett K_{N+1}.
06 **return** Y

2.24. tétel (tetszőleges kulcsok gyors rendezése). *A* REISCHUK *algoritmus n tetszőleges kulcsot n CREW PRAM processzoron $\overline{O}(\lg n)$ lépésben rendez.*

Bizonyítás. Reischuk algoritmusa szerint véletlenül kiválasztunk $N = n/\lg^4 n$ kulcsot és ezeket Preparata algoritmusával rendezzük. Ez a rendezett minta az eredeti sorozatot $N+1$ – közel azonos hosszúságú – részsorozatra bontja, melyeket Preparata algoritmusával rendezünk: ezek a rendezett részsorozatok a megfelelő sorrendben véve megadják az eredményt.

A REISCHUK algoritmus második lépése $N \lg N \le N \lg n$ processzoron

$O(\lg N) = O(\lg n)$ idő alatt elvégezhető.

A harmadik lépésben X felbontását bináris kereséssel és az egészeket rendező algoritmussal végezzük: minden kulcshoz tartozik egy processzor, amely az $l_1, l_2, \ldots, l_N$ sorozatban végzett bináris kereséssel megadja, hogy az adott kulcs a K_i részhalmazok melyikéhez tartozik.

Mivel a részek sorszámai a $[0, N+1]$ intervallumból vett egészek, az előző tétel alapján legfeljebb n processzoron $\overline{O}(\lg n)$ lépéssel rendezhetők. A K_i $(1 \le i \le N)$ részekben nagy valószínűséggel nem lesz több elem, mint $O(\lg^5 n)$. Ugyanennyi processzoron és lépéssel a $|K_i|$ elemszámokat is meg tudjuk határozni minden részre.

A negyedik lépésben minden K_i rész rendezhető $|K_i| \lg |K_i|$ processzoron $\lg |K_i|$ lépéssel. Ezért a negyedik lépés n processzoron

$$(\max_i \lg |K_i|)^2) \tag{2.16}$$

lépéssel elvégezhető. Ha $\max_i |K_i| = O(\lg^5 n)$, akkor a negyedik lépés $O((\lg \lg n)^2)$ lépéssel elvégezhető. ■

2.5. Gráfalgoritmusok

Legyen M egy $n \times n$ méretű mátrix, amelynek az elemei nemnegatív egész számok. Az $\tilde{M}$ mátrixot M ***minmátrixának*** nevezzük és a következőképpen definiáljuk:

$$\tilde{M}(i,i) = 0 \quad (1 \le i \le n) \tag{2.17}$$

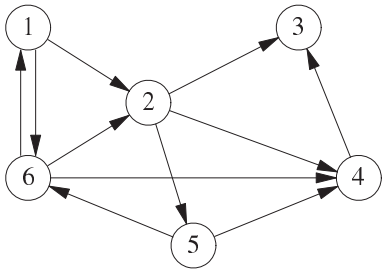
és

$$\tilde{M}(i,j) = \min\{M_{i_0 i_1} + M_{i_1 i_2} + \ldots + M_{i_{k-1} i_k}\} \quad (i \ne j)\,, \tag{2.18}$$

ahol a minimumot az olyan – az $\{1, 2, \ldots, n\}$ halmaz elemeiből képezett – $i_0, i_1, \ldots, i_k$ permutációkra nézve képezzük, amelyekre $i_0 = i$ és $i_k = j$.

Legyen G egy irányítatlan gráf, amelynek n csúcsa van. Ennek reflexív tranzitív lezártját A^*-gal jelöljük és úgy definiáljuk, hogy ha $i = j$ vagy V_i és V_j G-nek ugyanahhoz az összefüggő komponenséhez tartoznak, akkor $A^*(i,j) = 1$, egyébként $A^*(i,j) = 0$.

2.8. példa. *Irányított gráf és minmátrixa.* Legyen $G(V,E)$ egy irányított gráf, melynek csúcsait az $1, 2, \ldots, n$ számokkal címkézzük. Legyen $M[i,j] = 0$, ha $i = j$ vagy a gráf tartalmaz i-ből j-be vezető élet. Egyébként $M[i,j]$ legyen



M

0	0	1	1	1	0
1	0	0	0	0	1
1	1	0	1	1	1
1	1	0	0	1	1
1	1	1	0	0	0
0	0	1	0	1	0

$\tilde{M}$

0	0	0	0	0	0
0	0	0	0	0	0
1	1	0	1	1	1
1	1	0	0	1	1
0	0	0	0	0	0
0	0	0	0	0	0

2.8. ábra. Egy gráf és annak M-je és $\tilde{M}$-je.

1. Ekkor $\tilde{M}$ (i, j) pontosan akkor nulla, amikor van az i csúcsból a j csúcsba vezető út.

A 2.8. ábra egy 6 csúcsú irányított gráfot, valamint a hozzá tartozó M mátrixot és annak minmátrixát ábrázolja.

♠

2.5.1. Minmátrix

Több – gráfokkal kapcsolatos – feladat megoldását megkönnyíti a minmátrixok felhasználása.

A MINMÁTRIX algoritmus meghatározza adott M mátrix minmátrixát.

MINMÁTRIX$(M, \tilde{M})$ *párhuzamos eljárás*
Számítási modell: CREW PRAM
Bemenet: M (egy $n \times n$ méretű mátrix)

Kimenet: $\tilde{M}$ (az M mátrix minmátrixa)

01 P_{ij} **in parallel for** $i \leftarrow 1$ **to** n, $j \leftarrow 1$ **to** n
do $m[i,j] \leftarrow M[i,j]$
02 rendezzük az N kiválasztott kulcsot Preparata algoritmusával
Az eredmény legyen $l_1, l_2, \ldots, l_N$.
03 **for** $i \leftarrow 1$ **to** n
04 **do** P_{ijk} **in parallel for** $(i,j,k) \leftarrow 1$ **to** n
05 $q[i,j,k] \leftarrow m[i,j] + m[j,k]$
06 P_{ij} **in parallel for** $(i,j) \leftarrow$ **to** n
07 **do** $m[i,j] \leftarrow \min\{q[i,1,j], q[i,2,j], \ldots, q[i,n,j]$
08 P_i **in parallel for** $i \leftarrow 1$ **to** n
do $\tilde{M}\,(i,j) \leftarrow 0$
09 P_{ij} **in parallel for** $(i,j) \leftarrow 1$ **to** n
do $\tilde{M}\,(i,j) \leftarrow m(i,j)$

Ezzel kapcsolatos a következő állítás.

2.25. tétel (minmátrix számítása). *Ha ϵ tetszőleges pozitív szám, akkor a* Min-mátrix *algoritmus $n^{3+\epsilon}$ CRCW PRAM processzoron $O(\lg n)$ lépésben meghatározza egy $n \times n$ méretű mátrix minmátrixát.*

2.5.2. Tranzitív lezárt

A 2.25. tétel segítségével belátható a következő állítás.

2.26. tétel (tranzitív lezárt számítása). *Ha ϵ tetszőleges pozitív szám, akkor $n^{3+\epsilon}$ CRCW PRAM processzoron $O(\lg n)$ lépésben meghatározható egy n-csúcsú irányított gráf tranzitív lezártja.*

Bizonyítás. Ha M-et a 2.8. példa szerint definiáljuk, akkor egy G gráf tranzitív lezártja a minmátrix segítségével könnyen meghatározható. ∎

2.5.3. Összefüggő komponensek

Ugyancsak a 2.25. tétel segítségével bizonyíthatjuk a következő állítást.

2.27. tétel (összefüggő komponensek számítása). *Ha ϵ tetszőleges pozitív szám, akkor $n^{3+\epsilon}$ CRCW PRAM processzoron $O(\lg n)$ lépésben meghatározhatók egy n-csúcsú gráf összefüggő komponensei.*

Bizonyítás. Legyen $M[i, j] = 0$, ha $i = j$ vagy i és j össze vannak kötve egy éllel. Egyébként $M[i, j]$ legyen 1. Az i és j csúcsok pontosan akkor vannak ugyanabban az összefüggő komponensben, ha $\tilde{M}[i, j] = 0$. ■

2.5.4. Minimális feszítőfa

A soros Kruskal-algoritmus párhuzamosításával belátható a következő állítás.

2.28. tétel (minimális feszítőfa). . *Ha ϵ tetszőleges pozitív szám, akkor $n^{5+\epsilon}$* CRCW PRAM *processzoron $O(\lg n)$ lépésben meghatározható egy n-csúcsú élsúlyozott gráf minimális feszítőfája.*

2.5.5. Konvex burok

Gyakorlatok

2.5-1. A globális memória M_1 rekeszében van bizonyos adat. Másoljuk át ezt az adatot az $M_2, M_3, \ldots, M_n$ rekeszekbe. Mutassuk meg, hogyan lehet ezt megvalósítani $O(\lg n)$ lépéssel n EREW PRAM processzor felhasználásával.

2.5-2. Adjunk meg egy olyan algoritmust, amely $n/\lg n$ EREW PRAM processzor felhasználásával $O(\lg n)$ lépéssel megoldja az előző gyakorlatot.

2.5-3. Legyen $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. Adjunk $O(1)$ idejű CREW PRAM algoritmust a polinom értékének adott x helyen való kiszámítására. Mennyi processzort igényel a javasolt algoritmus?

2.5-4. Adjunk meg egy $O(\lg\lg n)$ idejű algoritmust, amely $n/\lg\lg n$ közös CRCW PRAM processzoron $O(\lg\lg n)$ lépésben megadja n tetszőleges szám maximumát.

2.5-5. Legyen A egy n kulcsot tartalmazó tömb. Mutassuk meg, hogy $n/\lg n$ CREW PRAM processzoron $O(\lg n)$ lépésben meghatározható tetszőleges $k \in A$ kulcs rangja.

2.5-6. Tervezzünk egy $O(1)$ lépésszámú algoritmust, amely n közös CRCW PRAM processzoron eldönti, hogy adott $A[1..n]$ tömb elemei között előfordul-e az 5, és ha igen, megadja a legnagyobb olyan i indexet, amelyre $A[i] = 5$.

2.5-7. Tervezzünk algoritmust, amely n^2 CREW PRAM processzoron $O(1)$ lépésben összefésül két n hosszúságú rendezett sorozatot.

2.5-8. Határozzuk meg a fejezetben tárgyalt algoritmusok relatív sebességét,

összes lépésszámát és hatékonyságát.

Feladatok

2-1. Közös elem
Tervezzünk algoritmust annak eldöntésére, hogy adott $A[1\,..\,n]$ és $B[1\,..\,n]$ tömböknek van-e közös eleme:

a. ha n^2 CRCW PRAM processzorunk van, akkor $O(1)$ lépésszámút;

b. *b.* ha n CRCW PRAM processzorunk van, akkor $\overline{O}(1)$ lépésszámút.

2-2. Minimális feszítőfa
Párhuzamosítsuk a minimális feszítőfák meghatározására szolgáló Kruskal-algoritmust és Prim-algoritmust. Tervezzünk algoritmust arra a speciális esetre, amikor az élek súlya csak 0 vagy 1 lehet.

2-3. Összes csúcspár távolsága
Párhuzamosítsuk a gráfok összes csúcspárjának távolságát meghatározó Bellman–Ford-algoritmust.

2-4. Körmentesség
Tervezzünk párhuzamos algoritmust annak eldöntésére, hogy adott irányítatlan gráf tartalmaz-e kört. Elemezzük a különböző nagyságrendű processzorszám esetében elérhető $W(n, p, \mathrm{P})$ futási időket.

Névmutató

Ez a névindex

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*[*szomsz*]. 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**.

Az algoritmusok nevében kiskötőjelet használtunk, viszont a változók neveiben alsó kötőjelet, mint például PP-ÖSSZEFÉSÜL 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.

Tartalomjegyzék

Megoldások