4. Hiperkocka

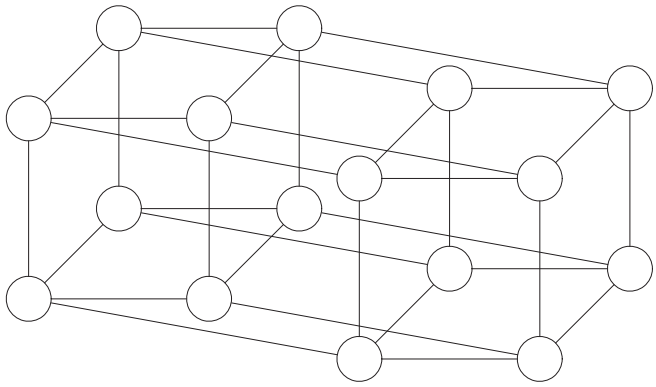
Ebben a fejezetben először a felhasznált számítási modelleket mutatjuk be, azután speciális hálózatos (mint a csomagirányítás, üzenetszórás, adatkoncentráló), végül tipikus „soros” (mint a kiválasztás, összefésülés, rendezés, gráfokkal kapcsolatos problémák) feladatokat megoldó algoritmusokat elemzünk.

4.1. Számítási modellek

Ebben az alfejezetben egyrészt a hiperkocka és pillangó számítási modelleket, másrészt hálózatok egymásba ágyazását tárgyaljuk.

4.1.1. Hiperkocka

Az első fejezetben szereplő definíció szerint egy d dimenziós hiperkocka, amelyet $\mathcal{H}_d$-vel jelölünk, 2^d processzorból áll. $\mathcal{H}_d$ minden processzora megcímkézhető egy d bites bináris számmal. A harmadik fejezetben lévő **??**. ábra egy 3 dimenziós, a 4.1. ábra pedig egy 4 dimenziós hiperkockát ábrázol.



4.1. ábra. 4 dimenziós hiperkocka.

A processzort és címkéjét ugyanazon szimbólummal jelöljük.

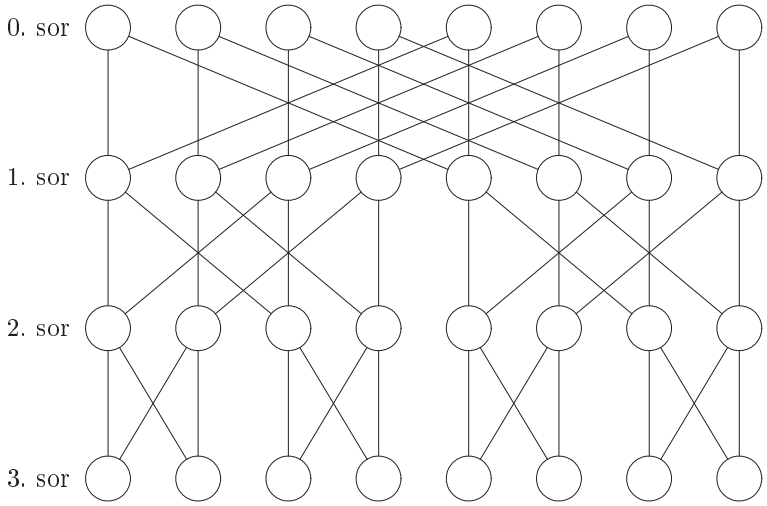
Ha v egy d bites bináris szám, akkor v első bitjét tekintjük legmagasabb helyiértékűnek. Jelölje $v(i)$ azt a d bites bináris számot, amely v-től csak az i-edik bitjében tér el. $\mathcal{H}_d$ minden P_v processzorára igaz, hogy az pontosan a $P_{v(i)}$ $(i = 1, 2, \ldots, d)$ processzorokkal van összekapcsolva. A $(v, v(i))$ kapcsolatot *i-edik szintű kapcsolatnak* nevezzük. Mivel $\mathcal{H}_d$ minden processzora pontosan d másikkal van összekötve, így $\mathcal{H}_d$ d-reguláris és a fokszáma d.

Az u és v bináris számok Hamming-távolsága azon bitpozíciók száma, amelyeken a két bináris szám eltér. Jele $H(u, v)$. A $\mathcal{H}_d$ hiperkocka bármely P_u és P_v processzora között van $H(u, v)$ hosszúságú út. Ha ugyanis u és v két processzor $\mathcal{H}_d$-ben, akkor egy közöttük vezető út megadható a következő módon. Legyenek $i_1, i_2, \ldots, i_k$ azon bitpozíciók (növekvő sorrendben) amelyeken P_u és P_v eltérnek. Ekkor létezik a következő útvonal: $u = w_0, w_1, w_2, \ldots, w_k = v$, ahol $w_j = w_{j-1}(i, j)$ $(1 \leq j \leq k)$. Ebből következik, hogy egy d-dimenziós hiperkocka átmérője pontosan d. A hiperkocka minden processzora egy helyi memóriával rendelkező RAM, amely minden alapvető műveletet (összeadás, kivonás, szorzás, osztás, összehasonlítás vagy a hozzáférés a helyi memóriához stb.) egységnyi idő alatt végez el. A processzorok közötti kommunikáció a processzorokat összekötő kapcsolatok mentén történik. Ha két processzor között nincs közvetlen kapcsolat, akkor a kommunikáció az egyik processzortól a másikig vezető út mentén lehetséges, ám az adatátvitel időigénye egyenesen arányos az út hosszával.

A kommunikáció egyidejűsége szempontjából kétféle hiperkockát különböztethetünk meg. Az első típus a ***soros*** vagy egyportos hiperkocka, amelynél egy processzor egységnyi idő alatt egyetlen szomszédjával képes kommunikálni. Ezzel szemben a ***párhuzamos*** vagy többportos hiperkocka egy processzora egységnyi idő alatt mind a d szomszédjával képes adatot cserélni. A továbbiakban mindig jelölni fogjuk, hogy melyik kommunikációs modellt használjuk. Mindkét megoldás szinkron processzormőködést tételez fel, azaz minden időegységben a processzorok mindegyike pontosan egy utasítást hajt végre. A hiperkockáknak több – számunkra kedvező – tulajdonsága is van. Az egyik a kicsi átmérő. Egy p processzoros hiperkockának az átmérője $\lg p$, míg az azonos számú processzort tartalmazó rács átmérője legalább $2(\sqrt{p} - 1)$. Egy másik kedvező tulajdonság, hogy $\mathcal{H}_{d+1}$ felépíthető rekurzívan. Vegyük ugyanis $\mathcal{H}_d$ két példányát, $\mathcal{H}'$-t és $\mathcal{H}''$-t. Egészítsük ki $\mathcal{H}'$ processzorainak címkéjét a 0 prefixszel, $\mathcal{H}''$-ét pedig az 1 prefixszel. Ezután $\mathcal{H}'$ minden P_v processzorát kössünk össze $\mathcal{H}''$ $P_{v(1)}$ processzorával. Ez a tulajdonság megfordítva azt is jelenti, hogy $\mathcal{H}_{d+1}$ $\mathcal{H}_d$-nek két példányát tartalmazza. Például azon processzorok, amelyek címkéjének első bitje 0, ha csak a közöttük futó kapcsolatokat vesszük figyelembe, pontosan kiadják $\mathcal{H}_d$-t. Sőt, tetszőleges $1 \leq q \leq d$-re, azon processzorok, amelyek címkéjének q-adik bitje azonos, kiadják $\mathcal{H}_d$-t. Még általánosabban, ha rögzítjük i $(1 \leq i \leq d + 1)$ bit értékét, a megadott feltételt kielégítő processzorok $\mathcal{H}_{(d+1)-i}$-t alkotnak.

4.1.2. Pillangó hálózat

A pillangóhálózat közeli kapcsolatban áll a hiperkockákkal. A pillangóhálózatokra tervezett algoritmusok könnyedén átültethetők hiperkockákra és fordítva. Valójában gyakran könnyebb az adott probléma megoldását pillangóhálózaton elkészíteni, majd onnan átültetni hiperkockára. Egy d-dimenziós pillangóhálózat, amelyet $\mathcal{B}_d$-vel fo-

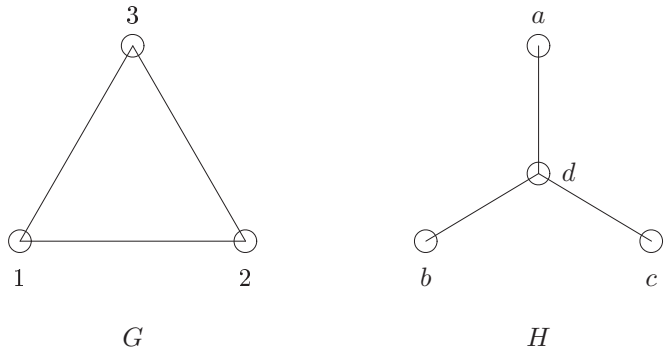


4.2. ábra. 4 sorban 32 processzort tartalmazó pillangó hálózat.

gunk jelölni, $p = (d+1)2^d$ processzorból és $d2^{d+1}$ élből áll. $\mathcal{B}_d$ minden processzora egy $\langle r, l\rangle$ párral jellemezhető, ahol $0 \leq r \leq 2^d - 1$ és $0 \leq l \leq d$. Az r változót a processzor ***sorindexének*** nevezzük, míg l a processzor *szintje*. Egy $P_u = \langle r, l\rangle$ processzor $(0 \leq l < d)$ $\mathcal{B}_d$-ben két, az $(l+1)$-edik szinten levő processzorral van összekötve, a $P_v = \langle r, l+1\rangle$ és $w = \langle r(l+1), l+1\rangle$ processzorokkal. Az (u, v) kapcsolatot ***közvetlen kapcsolatnak,*** (u, w)-t pedig ***kereszt kapcsolatnak*** nevezzük. Mindkét típust $(l+1)$-*edik szintű* kapcsolatnak mondjuk. Mivel minden processzor pontosan négy másikkal van összekötve, így $\mathcal{B}_d$ csúcsainak fokszáma (d-től, illetve p-től függetlenül) négy, ezért a d-dimenziós pillangó hálózat 4-reguláris és a fokszáma 4. Ha P_u egy 0-adik szintű processzor és P_v egy d-edik szintű, akkor létezik (és egyértelműen meg van határozva) egy d hosszúságú út P_u és P_v között. Legyen $P_u = \langle r, 0\rangle$ és $P_v = \langle r', d\rangle$. Ekkor az út $\langle r, 0\rangle, \langle r_1, 1\rangle, \langle r_2, 2\rangle, \ldots, \langle r', d\rangle$, ahol r_i-nek az első i bitje megegyezik r'-vel, a többi pedig r-rel $(1 \leq i \leq d-1)$. Vegyük észre, hogy ez az út létezik a pillangókapcsolatok definíciója miatt. Ezt az utat ***mohó útnak*** nevezzük. A 4.2. ábrán $\mathcal{B}_3$ látható. A vastagított vonal a $P_u = \langle 100, 0 >$ és $P_v = \langle 010, 3\rangle$ közötti mohó utat jelöli.

A fentiekből következik, hogy $\mathcal{B}_d$-ben bármely két processzor távolsága legfeljebb $2d$. A korlát éles, hiszen például a $P_u = \langle 0, d\rangle$ és $P_v = \langle 2d-1, d\rangle$ processzorok távolsága pontosan ennyi, így $\mathcal{B}_d$ átmérője $2d$. A pillangóhálózat is rendelkezik a hiperkockához hasonló rekurzív tulajdonsággal. Ha $\mathcal{B}_d$-ből eltávolítjuk a 0-adik szinten lévő processzorokat a hozzájuk csatlakozó élekkel együtt, akkor a hálózat két $(d-1)$-dimenziós pillangóhálózatra esik szét.

$\mathcal{H}_d$ és $\mathcal{B}_d$ között szoros kapcsolat van. Ha $\mathcal{B}_d$ minden sorát egyetlen processzorba fogjuk össze, megtartva a kimenő éleket, akkor az eredményül kapott gráf $\mathcal{H}_d$ (a keletkező többszörös éleket egyetlen példánnyal helyettesítve). Ennek következményeként kapjuk a következő lemmát.



4.3. ábra. Példa beágyazásra.

4.1. lemma (pillangó hálózat szimulálása). *$\mathcal{B}_d$ minden végrehajtási lépése szimulálható egy párhuzamos $\mathcal{H}_d$ egyetlen lépésével vagy egy soros $\mathcal{H}_d$ d lépésével.*

Egy $\mathcal{B}_d$ algoritmust ***normálisnak*** nevezünk, ha minden időpillanatban legfeljebb egy szint processzorai vesznek részt a kiszámításban.

4.2. lemma (normális algoritmus szimulálása). *Egy normális $\mathcal{B}_d$ egy lépése szimulálható egy soros $\mathcal{H}_d$ egyetlen lépésével.*

4.1.3. Hálózatok beágyazása

Egy hálózatnak egy másikra történő leképezését *beágyazásnak* nevezzük. Formálisan, a $G(V_1, E_1)$ hálózat beágyazása a $H(V_2, E_2)$ hálózatba, egy leképezés V_1-ről V_2-re. G-nek H-ra való leképezését felhasználva a G hálózatra tervezett algoritmusok lefuttathatók a H hálózaton. A 4.3. ábra bal oldalán látható G hálózat egy lehetséges beágyazása H-ba a következő leképezés: $1 \to b, 2 \to c, 3 \to a$.

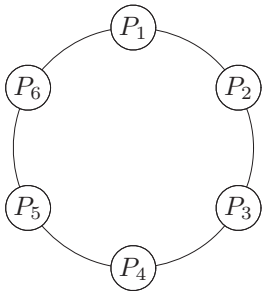
A beágyazás ***felfúvódásának*** a $|V_2|/|V_1|$ hányadost nevezzük. A beágyazás ***késleltetése***a leghosszabb út hossza, amelyre G-nek egy éle leképződött. A H hálózat egy éle ***torlódásának*** nevezzük az adott élt használó olyan utak számát, amelyekre G valamely élét leképeztük. A ***beágyazás torlódása*** a H élei torlódásának maximuma.

A 4.3. ábrán látható példában a felfúvódás mértéke 4/3, a késleltetés 2, mivel minden él egy-egy kettő hosszúságú útra képződött le. Hasonlóan a H összes élének torlódása 2, így az egész leképezés torlódása is 2.

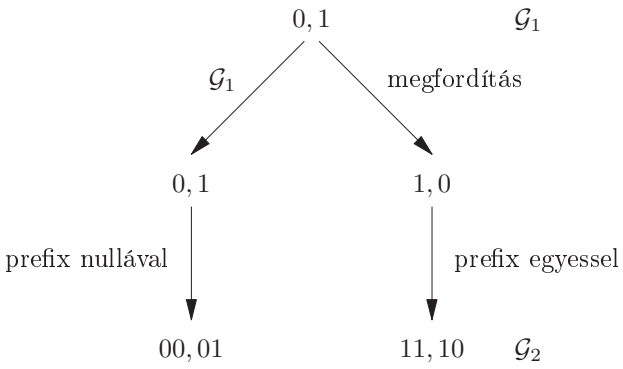
Gyűrű beágyazása

A p processzoros gyűrű egy síkhálózat, amelyben a P_i $(1 \leq i \leq p)$ processzor a P_{i+1} és a P_{i-1} processzorokkal van összekötve – ahol az indexeket (mod p) vesszük. A 4.4. ábra egy 6 processzoros gyűrűt ábrázol.

Megmutatjuk, hogyan lehet egy 2^d processzoros gyűrűt beágyazni $\mathcal{H}_d$-be. $\mathcal{H}_d$ processzorait d bites bináris számokkal címkézzük. Ha a gyűrű processzorait 0-tól $(2^d - 1)$-ig indexeljük a gyűrű mentén, akkor a 0-s processzor $\mathcal{H}_d$ $000\ldots00$ jelű



4.4. ábra. 6 processzoros gyűrű.



4.5. ábra. Gray-kód létrehozása.

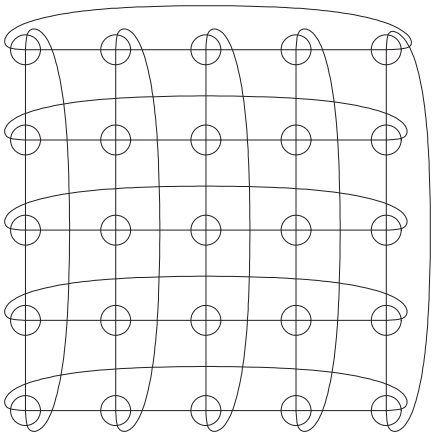
processzorára fog leképződni, a többi processzor megfelelőjét a Gray-kód segítségével határozhatjuk meg.

A ***k-ad rendű Gray-kód*** – jele $\mathcal{G}_k$ – a k-bites bináris számok egy adott permutációját definiálja. Az elsőrendű Gray-kód (G_1) a következő: 0, 1. $\mathcal{G}_k$-t ($k > 1$)-re rekurzívan definiáljuk a következőképpen: $0[\mathcal{G}_{k-1}], 1[\mathcal{G}_{k-1}]R$, ahol $0[\mathcal{G}_{k-1}]$ a $(k-1)$-edrendű Gray-kód elemeit jelenti úgy, hogy mindegyikükhöz hozzáragasztunk egy 0 prefixet. Hasonlóképpen $1[\mathcal{G}_{k-1}]R$ is a $(k-1)$-edrendű Gray-kód elemeit jelenti, ám fordított sorrendben és 1-es prefixszel.

A 4.5. ábra azt mutatja, hogyan származtatható $\mathcal{G}_2$ a $\mathcal{G}_1$ kódból. $\mathcal{G}_0$ elemei a nullás prefixszel a 00, 01 sorozatok adják, $\mathcal{G}_1$ elemeinek megfordítása az egyes prefixszel pedig az 11, 10 sorozatot eredményezi. Tehát $0[\mathcal{G}_2] = 00, 11, 11, 10$.

A $\mathcal{G}_k$ i-edik elemét $(0 \leq i \leq 2^k - 1)$ $g(i, k)$-val jelöljük.

A Gray-kód egyik tulajdonsága, hogy a szomszédos elemek pontosan egy bitben térnek el egymástól. Ebből az következik, hogy $\mathcal{G}_d$ a $\mathcal{H}_d$ processzorainak egy olyan permutációja, hogy a sorban egymást követő processzorok össze vannak kötve éllel a hiperkockában, azaz a gyűrű i-edik processzorát $(0 \leq i \leq 2^d - 1)$ a hiperkocka $g(i, d)$ processzorára képezzük le.



4.6. ábra. 5×5 méretű tórusz.

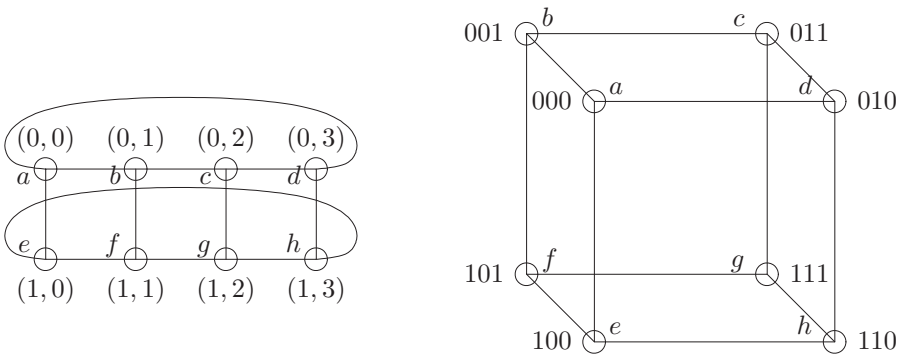
4.3. tétel (gyűrű beágyazása hiperkockába). *Egy 2^d processzorból álló gyűrű beágyazható $\mathcal{H}_d$-be úgy, hogy a felfúvódás, a késleltetés és a torlódás mindegyike* 1.

Tórusz beágyazása

Egy $m \times n$ tórusz származtatható egy $m \times n$ méretű rácsból úgy, hogy a rács minden sorának első és utolsó processzorát, valamint minden oszlopának első és utolsó processzorát is összekötjük: tehát a rácsot kiegészítjük a $P_{i,1}$-$P_{i,m}$ $(1 \leq i \leq m)$ és $P_{1,j}$-$P_{1,n}$ $(1 \leq j \leq n)$ élekkel.

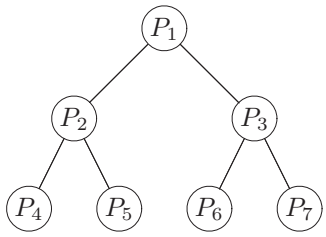
Már a 3×3 méretű tórusz ábrázolásához is 3 dimenzióra van szükségünk. A 4.6. ábra egy 5×5 méretű tóruszt ábrázol.

Legyen M egy $2^r \times 2^c$ méretű tórusz. Megmutatjuk, hogy M beágyazható úgy egy hiperkockába, hogy a felfúvódás, a késleltetés és a torlódás mindegyike 1 legyen. Ez az állítás egyszerűen adódik az előző fejezet végén kimondott lemmából. Van 2^r sor és 2^c oszlop M-ben. Korábban már láttuk, hogy ha egy d-dimenziós hiperkockában a d bitből valamely q-t rögzítjük $(1 \leq q \leq d-1)$, akkor a feltételnek megfelelő processzorok egy $\mathcal{H}_{d-q}$ alkockát alkotnak $\mathcal{H}_d$-ben. Ha egy $(r+c)$-bites bináris számnak rögzítjük az r ***legnagyobb helyi értékű bitjét*** (**M**ost **S**ignificant **B**its = MSB) és a maradék c bitet tetszőlegesen variáljuk, a kapott 2^c szám egy alkockát határoz meg $\mathcal{H}_{r+c}$-ben. A fenti lemma értelmében ebbe az alkockába beágyazható egy c processzorból álló gyűrű. Az r MSB minden lehetséges megválasztásához megvan a megfelelő $\mathcal{H}_c$, így M minden sorát leképezhetjük egyre. Egészen pontosan az i-edik sort azon $\mathcal{H}_c$-re képezzük, amelyet úgy kapunk, hogy az r MSB értékét pontosan $g(i,r)$-re állítjuk. Ebből következik, hogy általában a tórusz $P_{i,j}$ processzora a $\mathcal{H}_{r+c}$ $P_{g(i,r),g(j,c)}$ processzorára képződik. Így minden sor szomszédos processzorai a hiperkocka szomszédos processzoraira képződnek. A fentihez hasonló gondolatmenettel belátható, hogy a megadott leképezés az oszlopok szomszédos elemeit is szomszédos processzorokra képezi. Ebből következik, hogy mind a felfúvódás a késleltetés és a torlódás 1.



(a) (b)

4.7. ábra. Tórusz beágyazása hiperkockába.



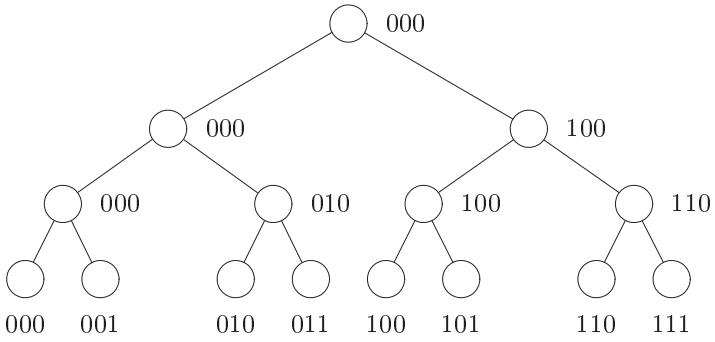
4.8. ábra. 3 szintes bináris fa hálózat.

A 4.7. ábra egy 2×4 méretű tórusznak a $\mathcal{H}_3$ pillangó hálózatba való beágyazását mutatja (a processzoroknak csak az indexe szerepel). Az ábra (a) részén ábrázolt tórusznak 2 sora (a nulladik és az első), valamint négy oszlopa (nulladik, első, második és harmadik) van. Például a tórusz $P_{1,2}$ csúcsát a hiperkocka $P_{g(1,1)g(2,2)} = P_{1,1,1,}$ csúcsára képezzük le. Az ábrán mind a két csúcsot g betűvel jelöltük.

Bináris fa beágyazása

Egy d szintes (teljes) bináris fában $p = 2^d - 1$ processzor van: P_1, $P_2, \ldots, P_p$. Az adatszerkezetekkel kapcsolatos terminológia szerint a P_1 processzort ***gyökérnek,*** a $P_{(p+1)/2}$, $P_{(p+1)/2+1}$, $\ldots$, P_p processzorokat ***levélnek*** a többi processzort ***belső processzornak*** nevezzük. Ha P_i nem levél, akkor össze van kötve a ***gyerekeinek*** nevezett P_{2i} és P_{2i+1} processzorokkal. Ha P_j nem a gyökér, akkor össze van kötve a *szülőjének* nevezett $P_{\lfloor j/2 \rfloor}$ processzorral. A 4.8. ábra egy 3 szintes bináris fa hálózatot ábrázol.

Bináris fák sokféle módon beágyazhatók hiperkockába. A következőkben megmutatjuk, hogy egy F p levelű (teljes, bináris) fa (ahol $p = 2^d$) beágyazható $\mathcal{H}_d$-be. Mivel a p levelű fának $2p-1$ processzora van, így a leképezés nem lehet kölcsönösen egyértelmű. Ha a leveleket $0, 1, \ldots, p-1$ jelöli, akkor képezzük az i-edik levelet $\mathcal{H}_d$ i-edik processzorára. F belső processzorait pedig képezzük arra a processzorra,



4.9. ábra. Bináris fa beágyazása hiperkockába.

amelyre az adott processzor legbaloldalibb leszármazottját képeztük. A 4.9. ábrán egy 8 levelű bináris fa csúcsainak leképezését láthatjuk.

A fa csúcsai melletti bitsorozat a $\mathcal{H}_3$ hiperkocka megfelelő csúcsának címkéje. Például a hiperkocka 0,0,0 csúcsára a fa 4 csúcsát képeztük le, míg az 1,1,1 csúcsra csak a fa egyetlen csúcsát.

A megadott beágyazás segítségével fa algoritmusokat hatékonyan szimulálhatunk soros hiperkockákkal. Ha a fa algoritmusban a számítás egy lépésében a fának legfeljebb egy szintjén lévő processzorok vesznek részt, akkor az a lépés a hiperkocka egyetlen lépésével szimulálható.

4.2. Csomagirányítás

A probléma: minden processzor legfeljebb egy csomagot küld és minden processzor legfeljebb egy csomag címzettje. Juttassuk el a csomagokat a feladótól a címzettekhez.

4.2.1. Mohó algoritmus

Tekintsük a csomagirányítási problémát először egy pillangóhálózaton, ahol kezdetben minden csomag a nulladik szinten helyezkedik el, a címzett processzorok pedig a d-ediken, azaz a címzett sorok a küldő sorok egy ***parciális permutációját*** alkotják. A mohó megoldás a csomagirányítási problémára ekkor az, hogy minden csomagot a mohó úton küldünk címzettjéhez. Ekkor minden csomag pontosan d hosszúságú utat tesz meg, így a továbbiakban az algoritmus vizsgálatához csak azt kell megvizsgálnunk, hogy mennyi késleltetést szenvedhet el egy csomag útja során. Legyen $u = \langle r, l \rangle$ egy tetszőleges processzor $\mathcal{B}_d$-ben. Ekkor legfeljebb 2^l csomag mehet keresztül u-n, hiszen a hálózat minden processzorának (a nulladik szintet kivéve) pontosan két szomszédja van a felette levő szinten. Hasonlóan a processzoron átmenő minden csomag címzettje az u-ból elérhető 2^{d-l} d-edik szinten levő processzor valamelyike. Ebből az következik, hogy az u processzorba befutó bármelyik l-edik szintű élen

legfeljebb $\min(2^{l-1}, 2^{d-l})$ csomag osztozik. Legyen most p egy tetszőleges csomag. p egy l-edik szintű élen áthaladva legfeljebb $\min(2^{l-1}, 2^{d-l})$ késleltetést szenved el $(l = 1, 2, \ldots, d)$. Így a maximális késleltetés, ami összesen egy csomagot érhet:

$$D = \sum_{l=1}^{d} \min\{2^{l-1}, 2^{d-l}\} \; . \tag{4.1}$$

Az általánosság megszorítása nélkül feltehetjük, hogy d páros. Ekkor D felírható a következőképpen:

$$\begin{aligned} D &= \sum_{l=1}^{d/2} 2^{l-1} + \sum_{l=d/2+1}^{d} 2^{d-l} && (4.2)\\ &= 2 * 2^{d/2} - 2 && (4.3)\\ &= \Theta(2^{d/2}) \; . && (4.4) \end{aligned}$$

Az $O(2^{d/2})$ érték felső korlát a maximális sorhosszúságra, ugyanis – mint láttuk – egy l-edik szintű processzoron legfeljebb $\min(2^{l-1}, 2^{d-l})$ csomag haladhat át. Ezen érték maximuma ($l = 1, 2, \ldots, d$-re) pedig $2^{d/2}$.

4.4. tétel (mohó algoritmus lépésszáma). *A mohó algoritmus $\mathcal{B}_d$-n $O(2^{d/2})$ időben fut, a maximális sorhosszúság szintén $O(2^{d/2})$.*

4.2.2. Véletlenített algoritmus

A véletlenítés, mint oly sokszor korábban, itt is számottevően javíthatja az előbb említett mohó algoritmus tulajdonságait. Már a rácsokon értelmezett csomagirányítási feladatoknál is alkalmaztuk azt a megoldást, hogy a csomagot először egy véletlenszerűen választott közbülső állomásra küldjük, majd onnan továbbítjuk eredeti céljához. Ezzel a megoldással elérhetjük, hogy a csomagok nagy valószínűséggel nem találkoznak egyetlen másik csomaggal sem útjuk során, aminek következtében az egyes élek menti torlódás kialakulásának valószínűsége jelentősen csökken. Ezt a stratégiát követjük a pillangóhálózatok esetében is.

A probléma tehát az eredeti, a javasolt megoldás pedig a következő Háromfázisú algoritmus:

1. fázis. A csomagot egy véletlenszerűen választott közbülső címre irányítjuk a d-edik szinten.

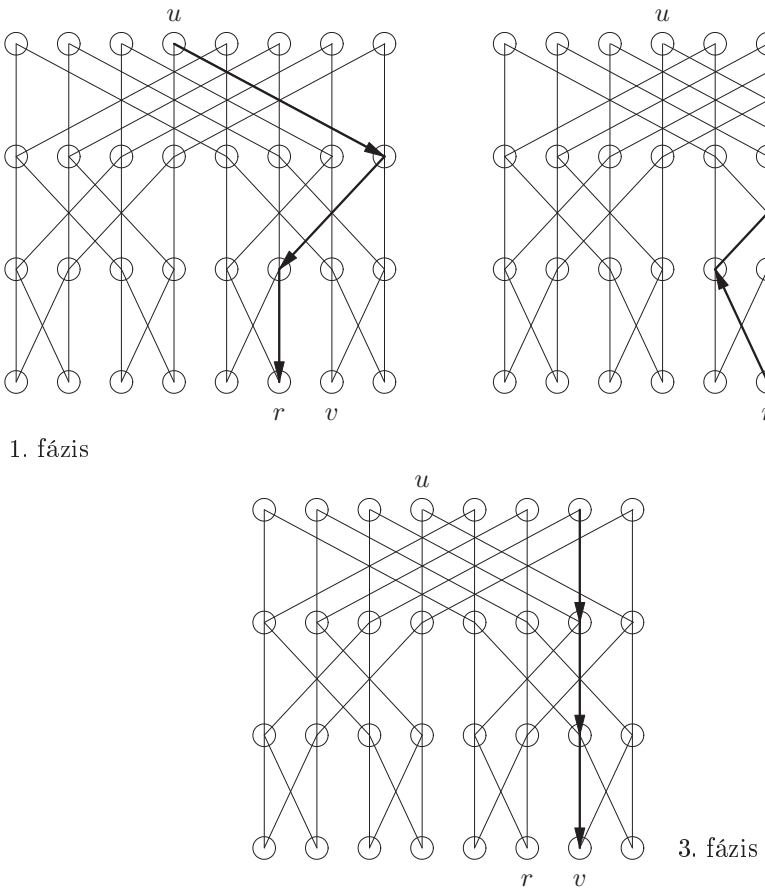
2. fázis. A csomagot az eredeti céljának megfelelő sorba irányítjuk, ám a nulladik szinten.

3. fázis. A csomagok elérik tényleges céljukat a d-edik szinten.

A 4.10. ábra szemlélteti az algoritmus 3 fázisát.

Az ábrán r a d-edik szint egy véletlenül választott csúcsa. u és v a vizsgált csomag kezdeti, illetve végső helye. A vastagított élek mutatják, hogy r az első fázisban eljut a $d$edik szintre, onnan a második fázisban a nulladik szintre, végül a harmadik fázisban a csomag célba ér.

A harmadik fázisban a csomagok végig a közvetlen éleken közlekednek, így akkor torlódás nem léphet fel, ezért a harmadik fázis végrehajtása pontosan d lépést



4.10. ábra. Véletlenített csomagirányítás 3 fázisa.

igényel. A második fázis az első fázis *inverze*, tulajdonságai megegyeznek az első fázisával, így a továbbiakban elegendő azt vizsgálni, hogy megkapjuk a teljes algoritmus lépésszámát. Ehhez felhasználjuk majd a következő definíciókat és lemmát.

Legyen P utak egy halmaza egy hálózatban. Azt mondjuk, hogy P ***nem ismétlődő úthalmaz,*** ha bármely két útra a P halmazból igaz, hogy a metszetük üres vagy összefüggő, azaz ha találkoznak, akkor egy darabig együtt mennek, majd szétválás után nem találkoznak többé.

Két csomagot ***átfedőnek*** mondunk egy hálózatban, ha az általuk megtett utak legalább egy közös élt tartalmaznak.

4.5. lemma (sorban állási lemma). *Legyen P utak halmaza egy hálózatban, amelyeken csomagok haladnak. Ha P nem ismétlődő úthalmaz, akkor tetszőleges p csomag késleltetése legfeljebb akkora, mint a p-vel átfedő csomagok száma.*

Bizonyítás. Legyen p egy tetszőleges csomag. Ha egyetlen p-vel átfedő csomag sem

késlelteti p-t egynél több alkalommal, akkor az állítás teljesül. Tegyük fel, hogy valamely q p-vel átfedő csomag kétszer is késlelteti p-t. Ekkor q maga is késleltetve volt egy p-vel átfedő másik csomag által, amely így már nem fogja késleltetni p-t. ■

4.2.3. Az első fázis elemzése

Legyen π egy tetszőleges csomag, jelölje továbbá e_i az az élt amelyen π az i-edik szinten áthalad ($1 \leq i \leq d$). A sorbaállási lemma alapján π késleltetésének meghatározásához elegendő meghatározni a π-vel átfedő csomagok számát. Jelöljük n_i-vel azon csomagok számát, amelyek útvonalában szerepel e_i. Ekkor

$$D = \sum_{i=1}^{d} n_i \tag{4.5}$$

felső korlát a π-vel átfedő csomagok számára. Tekintsük most az e_i élet. Ezen az élen legfeljebb $2^i - 1$ csomag halad keresztül, mivel ennyi processzor van a nulladik szinten amelyek mohó útvonalában szerepelhet e_i. Minden ilyen csomag $\frac{1}{2^i}$ valószínűséggel halad át az e_i élen mivel minden, a nulladik szintről induló, csomag $\frac{1}{2}$ valószínűséggel választja vagy a közvetlen- vagy a keresztélt minden szinten, egymástól függetlenül, i szinten keresztül, míg áthalad e_i-n. Így az n_i értéke $B(2^{i-1}, \frac{1}{2^i})$ binomiális eloszlású, melynek várható értéke $\frac{1}{2}$. Ebből következik, hogy D várható értéke a várható értékek összege, azaz $\frac{d}{2}$. Most megmutatjuk, hogy a teljes késleltetés nagy valószínűséggel $O(d)$. A D változónak $B(d, \frac{1}{2})$ egy felső korlátja. A Csernov-egyenlőtlenséget felhasználva az alábbi egyenlőtlenségeket kapjuk:

$$\begin{aligned}
P[D > e\alpha d] &\leq \left(\frac{d/2}{e\alpha d}\right)^{e\alpha d} e^{e\alpha d - d/2} && (4.6)\\
&< \left(\frac{1}{2e\alpha}\right)^{e\alpha d} e^{e\alpha d} && (4.7)\\
&\leq \left(\frac{1}{2e\alpha}\right)^{e\alpha d} && (4.8)\\
&< 2^{-e\alpha d} && (4.9)\\
&< p^{-\alpha-1}\,, && (4.10)
\end{aligned}$$

ahol $\alpha \geq 1$ és kihasználtuk azt a tényt, hogy $d = \Theta(\lg p)$. Mivel a csomagok száma p-nél kisebb, így annak a valószínűsége, hogy legalább egyikük $2e\alpha d$-nél nagyobb késleltetést szenved, kisebb, mint $p^{-\alpha-1}p = p^{-\alpha}$. Ezzel bebizonyítottuk a következő tételt.

4.6. tétel (Véletlen-csomag-irányít lépésszáma). *A* Véletlen-csomag-irányít *algoritmus $\mathcal{B}_d$-n $\overline{O}(d)$ lépést tesz.*

Mivel bármely hálózatban a hálózat átmérője alsó korlát a csomagirányítási probléma maximális lépésszámára, így a fenti algoritmus nagy valószínűséggel aszimptotikusan optimális.

A sorméret elemzése
Az algoritmus várakozási sorok mérete szintén $\overline{O}(d)$. Legyen v_i egy i-edik szintű processzor. A v_i-n potenciálisan átmenő csomagok száma 2^i. Minden ilyen csomag $1/2^i$ valószínűséggel halad át v_i-n, így az áthaladó csomagok várható értéke 1. A Csernov-egyenlőtlenség felhasználásával megmutatható, hogy az áthaladó csomagok száma $\overline{O}(d)$.

4.3. Alapvető algoritmusok

Ebben a fejezetben alapvető problémák – üzenetszórás, prefixszámítás, adat koncentráció – megoldására mutatunk hiperkocka algoritmusokat. Mindezen algoritmusok lépésszáma $O(d)$. Mivel a hálózat átmérője minden nem triviális probléma megoldásánál alsó korlát a lépésszámra, így ezek aszimptotikusan optimális algoritmusok.

4.3.1. Üzenetszórás fában

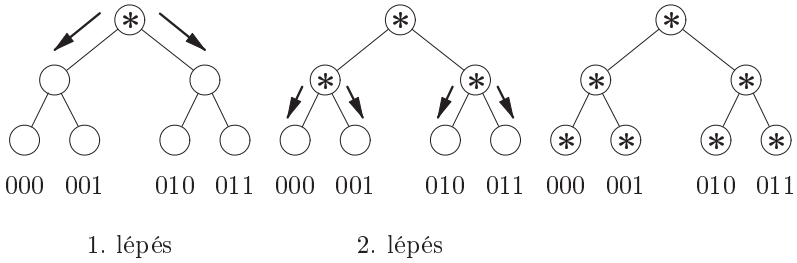
Az üzenetszórás problémája egy hálózatban azt a feladatot jelenti, amikor egy üzenetet valamely processzortól el kell juttatni a hálózathoz tartozó összes többi processzorhoz (vagy esetleg azok egy részhalmazához). Az üzenetszórást sokszor más algoritmusokba beépítve használjuk. A feladatot a FÁT-KOCKÁBA algoritmus segítségével oldjuk meg, amely a bináris fa architektúrának hiperkockába való beágyazásán alapul. Feltesszük, hogy az elküldendő üzenet a fa gyökerében található. Ez a processzor két másolatot készít a csomagról és elküldi a két leszármazottjának. Ha a fa egy belső processzora kap egy csomagot, akkor arról egy-egy másolatot továbbküld gyerekeinek. Ez folytatódik mindaddig, amíg minden levélen megjelenik az üzenet egy példánya.

4.7. tétel (üzenetszórás fában). *A fában való üzenetszórást a* FÁT-KOCKÁBA *algoritmus egy soros $\mathcal{H}_d$ hiperkockán $\Theta(d)$ lépésben oldja meg.*

Bizonyítás. A beágyazott bináris fa magassága d, így $O(d)$ lépés után minden levél processzor ismerni fogja az üzenetet. Az algoritmus végrehajtásakor minden időegységben a fának pontosan egy szintjén elhelyezkedő processzorok dolgoznak, más szóval az algoritmus normális, így egy lépése – amint azt a bináris fáknak a hiperkockába ágyazásakor megmutattuk – $\mathcal{H}_d$-nek pontosan egy lépésével szimulálható.

Másrészt legrosszabb esetben legalább egy levélhez el kell juttatni az üzenetet, ezért a fenti beágyazásra $W(d, \text{FÁT-KOCKÁBA}) = \Omega(d)$. ∎

4.1. példa. *Üzenetszórás 4 levelű fában.* A 4.11. ábra egy 3 szintes fát mutat, melyben a gyökérben lévő üzenetet juttatjuk el a többi csúcshoz. Az ábra első része azt mutatja, hogy az első lépésben az üzeneteket a gyökércsúcs gyerekei kapják meg. A középső részen az üzenetek a gyerekek gyerekeihez, azaz a levélcsúcsokhoz jutnak el. Végül az ábra harmadik része a végső állapotot mutatja, amelyben már minden csúcs megkapta az üzenetet. Mivel a fát úgy ágyaztuk be a hiperkockába, hogy a fa különböző szintjein lévő processzorok a



4.11. ábra. Üzenetszórás 4 levelű fában.

hiperkocka különböző processzorára képződjenek le, $\mathcal{H}_2$ két lépésben megoldja a feladatot.

4.3.2. Prefixszámítás fán

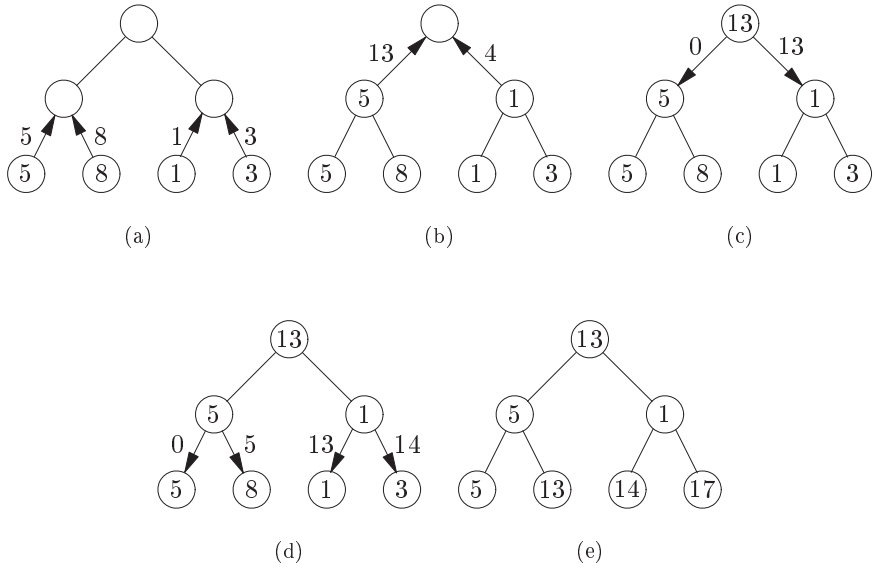
Ezen feladat megoldásához ismét a bináris fa beágyazását fogjuk felhasználni. Legyen x_i a bemenet a 2^d levelű bináris fa i-edik levelén. Az alább bemutatott PREFIXFÁBAN algoritmus két fázisban számítja ki az eredményt. Az első (felfelé haladó) fázisban az adatok felfelé áramlanak a bináris fában, a gyökér felé. A második fázisban (lefelé haladó) az adatok a gyökér felől haladnak a levelekhez. Mindkét fázis minden lépésében a bináris fának pontosan egy szintje vesz részt a számításban, azaz az algoritmus normális.

Felfelé haladó fázis. A levelek elküldik az x_i-ket a szülő csúcsoknak. A fa egy tetszőleges belső csúcsa, amennyiben két elemet kap (y-t a bal és z-t a jobb oldali leszármazottjától), akkor kiszámítja a $w = y + z$ értéket, tárolja w-t és y-t, majd elküldi w-t a szülőjének. Az első fázis végére minden processzor kiszámítja és tárolja a hozzá tartozó részfában levő bemeneti elemek összegét. Így a gyökér az összes elem összegét tartalmazza.

Lefelé haladó fázis. A gyökérben levő processzor elküld egy nullát a bal oldali gyerekének és y-t a jobb oldalinak. Egy tetszőleges belső processzor ha q értéket kapja a szülőjétől, akkor elküldi q-t a bal oldali és $q \oplus y$ értéket a jobb oldali gyerekének. Az i-edik levél processzor, ha a szülőjétől a q értéket kapja, akkor kiszámolja és végeredményként tárolja az $x_i + q$ értéket.

Az algoritmus felfelé haladó fázisában minden processzor a hozzá tartozó részfa leveleiben tárolt számok összegét számolja ki. Legyen P_v egy processzor és jelöljük $P_{v'}$-vel P_v részfájának legbaloldalibb levele. Ekkor a lefelé haladó fázis során a P_v által kapott q értéke, azaz q a $P_{v'}$-től balra eső elemek összege. Az észrevétel felhasználásával az algoritmus helyessége egyszerűen bizonyítható. Az algoritmus mindkét fázisa $d-1$ lépést igényel. Mivel minden időpillanatban a fának pontosan egy szintje aktív, így az algoritmus minden lépése szimulálható $\mathcal{H}_d$ egy lépésével.

4.2. példa. *Prefixszámítás fán.* A 4.12. ábrán egy 4 levelű fa leveleiben vannak az 5, 8,1 és 4 számok. A feladat a megfelelő prefixek számolása, ha az elvégzendő művelet az összeadás. Az ábra (a) része szerint minden levél elküldi a tárolt számot a szülőjének. A (b) ábrarész



4.12. ábra. Prefixszámítás bináris fán.

szerint a szülők tárolták a bal oldali gyereküktől kapott értéket (a kiszámított összeget is, de ezt az ábra nem mutatja), és szülőjüknek elküldték a kapott 2 szám összegét.

4.8. tétel (prefixszámítás fán és pillangón). *A* Prefix-fán *algoritmus a prefixszámítást* 2^d *levelű bináris fán, valamint* $\mathcal{H}_d$*-n is* $\Theta(d)$ *lépésben oldja meg.*

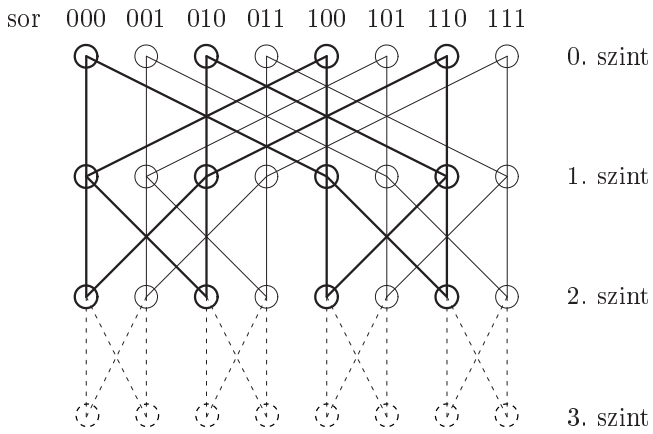
Összegzési feladatnak nevezzük az $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ kifejezés értékének kiszámítását adott x_i-khez. A fenti algoritmus felfelé haladó fázisának végére kiszámítja ezt az összeget, így az összegzési feladat megoldásához szükséges idő a fele az összes prefix kiszámításához szükséges időnek.

4.3.3. Adatkoncentráció

Legyen egy $\mathcal{H}_d$ hiperkocka $k (k < p)$ processzorán egy-egy elem elhelyezve. A feladat, hogy mozgassuk ezen elemeket a hiperkocka $P_0, P_1, \ldots, P_{k-1}$ processzoraira (processzoronként egy elemet elhelyezve).

Ha meg tudjuk határozni, hogy melyik elemet melyik processzorra kell juttatnunk, akkor a 4.2.2. pontban vizsgált véletlenített Három-fázisú csomagirányítási algoritmus segítségével $O(d)$ lépésben megtehetjük ezt. Ez az algoritmus párhuzamos (többportos) hiperkockát feltételez.

Van azonban egy jóval egyszerűbb, determinisztikus algoritmus is a feladat megoldására. Pontosabban a feladatra adunk egy normális megoldást pillangóhálózaton, majd felhasználjuk az ide vonatkozó beágyazási lemmát. Előtte azonban vizsgáljuk meg a pillangóhálózatok két tulajdonságát, amelyekre az algoritmus elemzésekor



4.13. ábra. A d-edik szinten lévő processzorok és élek eltávolítása.

hivatkozni fogunk.

1. tulajdonság. Ha egy pillangóhálózatból eltávolítjuk a d-edik szint összes processzorát és a hozzájuk kapcsolódó éleket (lásd 4.13. ábra), két $d-1$ szintes pillangóhálózatot kapunk eredményül. Az egyik hálózat az eredetinek a páros sorait tartalmazza, így ezt ***páros alhálózatnak*** nevezzük, a másik a páratlanokat, így az a ***páratlan alhálózat.***

2. tulajdonság. A nulladik szint bármely processzora a leszármazottaival együtt egy 2^d levelű bináris fát alkot, amelynek levelei pontosan a d-edik szintű processzorok.

Ezek után felvázolhatjuk az adatkoncentráció problémájának megoldó algoritmusát.

4.3.4. Kisszámú elem rendezése hiperkockán

A *ritka leszámláló rendezés* problémája már szerepelt a harmadik fejezetben. Legyen $X = k_0,\ k_1,\ \ldots k_{\sqrt{p}-1}$, ahol $p = 2^d$. Az általánosság megszorítása nélkül feltehető, hogy d páros. Tegyük fel, hogy a bemenő adatok a hiperkocka azon processzorain vannak (processzoronként egy kulcs), melyek címkéjét úgy kapjuk, hogy a címkék d bitje közül az első $\frac{d}{2}$-t nullának választjuk. A rendezett sorozatot ugyanezen processzorokon állítjuk elő kimenetként.

Legyen P_v a $\mathcal{H}_d$ hiperkocka egy processzora. P_v címkéje $\langle i, j\rangle$ párként is felfogható, ahol i az első $\frac{d}{2}$ bit, j pedig a második $\frac{j}{2}$ bit. Azok a processzorok, melyek címkéjének első fele i ($0 \leq i \leq 2^{d/2}-1$), egy $\mathcal{H}_d$ hiperkockát alkotnak. Ezt a hiperkockát $\mathcal{H}_d$ *i-edik sorának* fogjuk nevezni. Hasonlóképpen határozzuk meg a hiperkocka *j-edik oszlopát*. A rendezendő kulcsok kezdetben a nulladik sor processzorain vannak. Az egyértelműség kedvéért tegyük fel, hogy az r rangú kulcs kimenetként a $\langle 0, r-1\rangle$ ($0 \leq r \leq 2^{d/2}$) processzoron fog megjelenni.

A KEVESET-KOCKÁN algoritmus a következőképpen működik.

KEVESET-KOCKÁN(X, Y) *párhuzamos eljárás*
Számítási modell: hiperkocka
Bemenet: $X = k_0,\ k_1,\ \ldots, k_{\sqrt{p}-1}$ (rendezendő kulcsok)
Kimenet: $Y = l_0,\ l_1,\ \ldots, l_{\sqrt{p}-1}$ (rendezett kulcsok)

01 O_j **in parallel for** $j \leftarrow 1$ **to** $\sqrt{p}-1$
szórja k_j-t úgy, hogy minden sor
megkapja X egy másolatát
02 S_i **in parallel for** $i \leftarrow 1$ **to** $\sqrt{p}-1$
számítsa k_i rangját úgy, hogy a sor
minden processzorához
szétszórja k_i-t, majd a bemenet minden
elemével összehasonlítja, és összeget számol
03 S_i **in parallel for** $0 \le i \le \sqrt{p}-1$
szórja szét a sorban k_i rangját
04 S_i **in parallel for** $i \leftarrow 0$ **to** $\sqrt{p}-1$
if $r(k_i) = r_i$
then $\langle i, r_i - 1\rangle$ szórja k_i-t O_i
processzoraihoz úgy, hogy
végül $\langle 0, r_i - 1\rangle$ megkapja
azt a kulcsot, amelynek kiviteléért felelős

A lépészám a következő.

4.9. tétel (ritka leszámláló rendezés kockán). *A* KEVESET-KOCKÁN *algoritmus legfeljebb* $\sqrt{p}$ *kulcs ritka leszámláló rendezését egy* $\mathcal{H}_d$ *hiperkockán* $\Theta(d)$ *lépésben végzi el.*

Bizonyítás. Az algoritmus csak olyan műveleteket végez, amelyben egy sor vagy oszlop processzorai vesznek részt. Az elvégzett műveletek mindegyike – prefixszámítás, üzenetszórás, összehasonlítás – $O(d)$ lépésben elvégezhető, és legrosszabb esetben $\Omega(d)$ lépést igényel. ■

4.4. Kiválasztás

Az egyszerű kiválasztás célja, hogy adott n kulcs közül az i-edik $(1 \le i \le n)$ legkisebbet meghatározzuk. A korábbiakhoz hasonlóan két esetet vizsgálunk: az egyikben a processzorok p száma megegyezik a kulcsok n számával, a másikban pedig a processzorok száma kisebb, mint a kulcsoké.

Három algoritmust mutatunk be, mindegyik pillangó hálózaton fut. Az elsőnél $p = n$, a másik kettőnél $p < n$. Az első nagy valószínűséggel munkahatékony, a másik kettő viszont nem az.

4.4.1. Véletlenített algoritmus a $p = n$ esetre $(*)$

A PRAM-on futó munkahatékony véletlenített algoritmus alkalmazható hiperkockán is.

A KOCKÁN-KIVÁLASZT fokozatainak száma $\overline{O}(1)$. A PRAM-KIVÁLASZT első lépése hiperkockán $O(1)$ lépést vesz igénybe. A 2. és 5. lépésekben szükséges prefixszámítás összesen $O(d)$ lépésben elvégezhető. A 3. és 6. lépésekben a koncentrálás ugyancsak megoldható $O(d)$ lépésben. A 3. és 6. lépésekben szükséges ritka rendezés ugyanennyi lépést vesz igénybe. Mivel a 4. és 6. lépésben rendezett sorozatból kell kiválasztani, ezért az $O(1)$ lépésben megoldható. A 2., 4. és 5. lépésben lévő üzenetszórás lépésigénye $O(d)$. Innen adódik a következő tétel.

4.10. tétel (kiválasztás hiperkockán). *Ha $p = n$, akkor a* KOCKÁN-KIVÁLASZT *algoritmus a kiválasztási feladatot a $\mathcal{H}_d$ hiperkockán $\overline{O}(d)$ idő alatt megoldja.*

4.4.2. Véletlenített algoritmus a $p < n$ esetre $(*)$

Tegyük fel, hogy $\epsilon > 1$ és $n = p^\epsilon$. A PRAM modellen futó VÉLETLEN-KIVÁLASZT algoritmus módosított változata a rácsokhoz hasonlóan most is alkalmazható.

A KOCKÁN-VÉL-KIVÁLASZT algoritmus esetén minden processzor n/p kulccsal kezd. A **while** utasítás feltételét $(N > D)$-re változtatjuk (ahol D állandó). Az első lépésben minden processzor minden kulcsa $1/(N^{1-(1/3c)})$ valószínűséggel lesz a mintában.Ezért ez a lépés ebben az esetben n/p lépésig tart. A második lépés változatlan és $O(d)$ lépést vesz igénybe. A 3. lépésben a koncentrálás és a ritka leszámláló rendezés is végrehajtható $O(d)$ lépésben. A 4., 5. és 6. lépések szintén végrehajthatók $O(d)$ lépésben. Így minden fokozat $O(n/p+d)$ lépést vesz igénybe. Az algoritmusnak $\overline{O}(\lg\lg n)$ fokozatra van szüksége. A 6 lépésre külön kapott becslések összegét a fokozatszámra kapott becsléssel összeszorozva kapjuk a következő tételt.

4.11. tétel (kiválasztás hiperkockán). *Ha $c > 1$ és $n = p^c$, akkor a kiválasztási feladat a $\mathcal{H}_d$ hiperkockán $\overline{O}\left(((n/d) + d)\lg\lg p\right)$ idő alatt megoldható.*

4.4.3. Determinisztikus algoritmus a $p < n$ esetre

A négyzeten futó kiválasztó algoritmus adaptálható hiperkockára. Az így kapott KOCKÁN-DET-KIVÁLASZT algoritmus lépésszámának elemzése hosszú, csak az eredményét adjuk meg.

4.12. tétel (kiválasztás hiperkockán). *Ha $p < n$, akkor a* KOCKÁN-DET-KIVÁLASZT *algoritmus a kiválasztási feladat a $\mathcal{H}_d$ hiperkockán $O\left(\frac{n}{p}\lg\lg p + d^2\lg n\right)$ lépésben megoldja.*

4.3. példa. Legnagyobb elem kiválasztása hiperkockán Tegyük fel, hogy a $\mathcal{H}_3$ hiperkocka minden processzorának memóriájában 5-5 kulcsot helyezünk el – ahogyan azt a 4.14. ábra mutatja. A feladat a harminckettedik legkisebb elem kiválasztása.

Processzor	000	001	010	011	100	101	110	111
	5	20	(18)	35	63	21	62	11
	10	15	24	42	71	9	51	28
	(6)	7	12	3	1	36	(45)	17
	4	(16)	13	19	2	47	8	81
	27	23	32	(22)	(55)	(26)	30	(25)
					↓ A súlyozott médián 22.			
	(27)	(23)	(24)	(35)	(63)	(36)	62	(28)
			32	42	71	47	51	81
					55	26	(45)	25
							30	
					↓ A súlyozott médián 36.			
	-	-	-	42	63	47	62	81
					71		51	
					55		45	

4.14. ábra. Kiválasztás 8 elem közül

Először minden processzor meghatározza saját kulcsainak mediánját – ezeket az ábra felső részén bekarikáztuk. Ezen mediánok rendezett sorozata 6, 16, 18, 22, 25, 26, 45, 55. Mivel most minden processzornál 5-5 kulcs van, a súlyozott médián megegyezik a mediánnal, ami 22. Ezután meghatározzuk $M = 22$ rangját, ami 21. Mivel $i = 32 > 21$, a 22-nél kisebb vagy vele egyenlő kulcsokat elhagyjuk. i frissített értéke $32 - 21 = 11$ lesz. Ezzel a **while** ciklus egy menetét befejeztük.

A **while** ciklus következő menetében 19 elemmel kezdünk. Az ábra középső részén bekarikáztuk a helyi mediánokat. A mediánok rendezett sorozata 23, 24, 27, 28, 35, 36, 45, 63, a megfelelő súlysorozat pedig 1, 2, 1, 3, 2, 3, 4, 3, így a súlyozott médián 36. Elhagyjuk a kis kulcsokat és $i = 11 - 10 = 1$ lesz az új i érték. Ezzel vége a **while** ciklus következő menetének.

Így folytatva megkapjuk, hogy a kiválasztás eredménye a megmaradt 9 szám közül a legkisebb, a 42.

4.5. Összefésülés

Amint már láttuk az előző két fejezetben, az összefésülés célja, hogy két rendezett sorozatból a két sorozat minden elemét tartalmazó, rendezett sorozatot állítsunk elő.

Beláttuk, hogy ez a feladat hiperkockán – m hosszúságú bemenő sorozatokra – m^2 processzoron $O(\lg m)$ idő alatt megoldható. A számítási modell most is a hiperkocka lesz, de csak $2m$ processzort használunk (feltéve, hogy m 2 hatványa).

4.5.1. Páratlan-páros összefésülés

Legyen $X_1 = k_0,\ k_1,\ \ldots, k_{m-1}$ a két összefésülendő rendezett sorozat, ahol $2m = 2^d$.

Először szétválasztjuk X_1 és X_2 páros és páratlan részét. Legyenek ezek O_1, E_1, O_2 és E_2. Ekkor E_1-et rekurzívan összefésüljük O_2-vel, az eredmény $A = a_0, a_1, \ldots, a_{m-1}$. Ugyanígy kapjuk $B = b_0, b_1, \ldots, b_{m-1}$-et O_1-ből és E_2-ből. Ezután A és B összekeverésével kapjuk a $C = a_0, b_0, a_1, b_1, \ldots, a_{m-1}, b_{m-1}$ sorozatot, majd rendre összehasonlítjuk (és szükség esetén felcseréljük a_i-t és b_i-t $(0 \leq i \leq m-1)$. Az algoritmus helyessége belátható a 0-1-elv segítségével.

4.4. példa. 2×4 *elem összefésülése.* Legyen $X_1 = 7, 11, 24, 30$ és $X_2 = 2, 4, 27, 45$. Ebben az esetben $O_1 = 11, 30$ és $E_1 = 7, 24$, $O_2 = 4, 45$ és $E_2 = 2, 27$. E_1 és O_2 összefésülésével $A = 4, 7, 24, 45$ adódik. Hasonlóképpen $B = 2, 11, 27, 30$. A és B összekeverésének eredménye $C = 4, 2, 7, 11, 24, 27, 45, 30$. Ha a 4-et és 2-t, valamint a 45-öt és 30-at felcseréljük, akkor az eredmény 2, 4, 7, 24, 27, 30, 45.

A módosított algoritmust könnyű a $\mathcal{B}_\lceil$ pillangón megvalósítani. Például X_1 és X_2 páros és páratlan részre való felbontása a pillangó hálózaton egy lépésben elvégezhető.

A keverési művelet szintén könnyen elvégezhető. tegyük fel, hogy X_1 és X_2 is a $\mathcal{B}_d$ pillangó hálózat d-edik szintjének bemenő adatai. Legyen X_1 a bemenet az első m sorban és X_2 a második m sorban. Az algoritmus első lépése X_1 és X_2 szétválasztása páratlan és páros részre. Ezután rekurzívan összefésüljük E_1-et O_2-vel és O_1-et E_2-vel. Ennek érdekében az első m sorban lévő kulcsokat a közvetlen éleken, a többi kulcsot a keresztéleken mozgatjuk (lásd 4.15. ábra).

4.13. tétel. *Ha $m = 2^d$, akkor két m hosszúságú lista mind a $\mathcal{B}_d$ pillangó hálózaton, mind pedig a $\mathcal{H}_d$ hiperkocka hálózaton összefésülhető $O(d)$ idő alatt.*

4.5.2. Biton összefésülés

A $K = k_0,\ k_1,\ \ldots,\ k_{n-1}$ sorozatot *biton sorozatnak* nevezzük, ha rendelkezik a következő két tulajdonság valamelyikével:

a) van olyan j $(0 \leq j \leq n-1)$ index, amelyre $k_0 \leq k_1 \leq \ldots \leq k_j$ és $k_j \geq k_{j+1} \geq \ldots \geq k_{n-1}$;

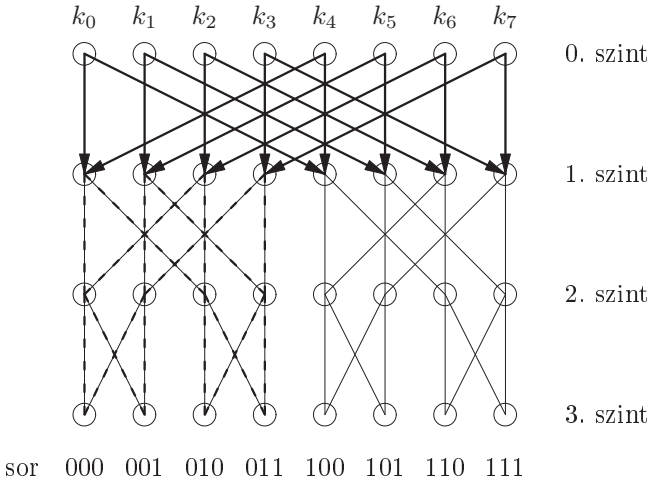
b) van olyan j $(1 \leq j \leq n-1)$ index, amelyre a K sorozat $K' = k_{j+1},\ k_{j+2},\ \ldots,\ k_{n-1},\ k_1, k_2,\ \ldots,\ k_{j-1}$ ciklikus eltoltja rendelkezik az előző tulajdonsággal.

Megmutatjuk, hogyan lehet rendezni egy biton sorozatot pillangó hálózaton.

Az első lépésben – ahogyan ezt a 4.16. ábra mutatja – a nulladik szint processzorai mind a közvetlen, mind a kereszt éleken elküldik a kulcsukat. Az első szint processzorai felülről 2-2 kulcsot kapnak.

Processzor	000	001	010	011	100	101	110	111
	5	20	(18)	35	63	21	62	11
	10	15	24	42	71	9	51	28
	(6)	7	12	3	1	36	(45)	17
	4	(16)	13	19	2	47	8	81
	27	23	32	(22)	(55)	(26)	30	(25)
				↓	A súlyozott médián 22.			
	(27)	(23)	(24)	(35)	(63)	(36)	62	(28)
			32	42	71	47	51	81
					55	26	(45)	25
							30	
				↓	A súlyozott médián 36.			
	-	-	-	42	63	47	62	81
					71		51	
					55		45	

4.15. ábra. Páratlan-páros összefésülés pillangón.



4.16. ábra. Bitonikus összefésülés pillangó hálózaton.

4.14. tétel. *Egy 2^d hosszúságú biton sorozat mind a $\mathcal{B}_d$ pillangó hálózaton, mind*

pedig a $\mathcal{H}_d$ hiperkocka hálózaton rendezhető $O(d)$ idő alatt.

4.6. Rendezés

Ebben az alfejezetben hiperkocka és pillangó hálózat lesz a számítási modell.

4.6.1. Páratlan-páros összefésülő rendezés

Első algoritmusunk a PÁROS-PÁRATLAN-FÉSÜL-RENDEZ egy megvalósítása.

Jelöljük $R(d)$-vel algoritmusunk lépésszámát $\mathcal{B}_d$-n. Ekkor

$$R(d) = R(d-1) + O(d). \tag{4.11}$$

Ennek a rekurzív egyenletnek a megoldása $S(d) = O(d^2)$.

4.15. tétel. *Ha $p = 2^d$, akkor p elem mind a $\mathcal{B}_d$ pillangó hálózaton, mind pedig a $\mathcal{H}_d$ soros hiperkockán rendezhető $O(d^2)$ idő alatt.*

4.6.2. Biton rendezés

Az összefésülő rendezés alapötlete a biton összefésülő algoritmussal kapcsolatban is alkalmazható – az eredmény a PILLANGÓN-BI TON-RENDEZ algoritmus.

Most ismét felírhatjuk a 4.11 rekurzív egyenletet, ahonnan a következő lépésszámot kapjuk.

4.16. tétel (biton sorozat rendezése pillangón és hiperkockán). *A* PILLANGÓN-BITON-RENDEZ *algoritmus $p = 2^d$ elemet $O(d^2)$ lépésben rendez a $\mathcal{B}_d$ pillangó hálózaton, és ugyancsak $O(d^2)$ lépésben hiperkockán.*

4.7. Gráfalgoritmusok

Ebben az alfejezetben minmátrix, tranzitív lezárt, összefüggő komponensek és minimális feszítőfa meghatározására szolgáló algoritmusokat mutatunk be.

4.7.1. Minmátrix meghatározása

Újra alkalmazzuk a második fejezetben bevezetett formalizmust.

Legyen q egy pozitív egész szám, $n = 2^q$ és legyen $M[0 : n-1, 0 : n-1]$ egy $n \times n$ méretű mátrix. Az M mátrix minmátrixának számítására a $\mathcal{H}_{3q}$ hiperkockát fogjuk használni, amelyben 2^{3q} processzor van, melyek címkéi $3q$ bit hosszúak. Ezek a címkék $\langle i, j, k\rangle$ alakban is felírhatók, ahol i a címke első q bitjét, j a címke második q bitjét és k a címke harmadik q bitjét jelentik.

Jelölje $(i, \star, \star)$ $(0 \le i \le 2^q - 1)$ a $\mathcal{H}_{3q}$ hiperkocka azon processzorait, melyek címkéjében az első q bit bináris számként i. Ezek a processzorok együtt egy $\mathcal{H}_{2q}$

hiperkockát alkotnak. Hasonlóképpen definiáljuk a $(\star, j, \star)$, $(\star, \star, k)$, $(i, j, \star)$, $(i, \star, k)$ és $(\star, j, k)$ jelöléseket is.

Ekkor a KOCKÁN-MINMÁTRIX algoritmus a következőképpen működik.

KOCKÁN-MINMATRIX(par) *párhuzamos eljárás*
Számítási modell: hiperkocka
Bemenet: M (egész elemeket tartalmazó mátrix)
Kimenet: M minmátrixa

```
01 frissítsük a q[i,j,k] elemeket
02 frissítsük az m[i,j] értékeket
```

A lépészám a következő.

4.17. tétel. *Ha M egy $n \times n$ méretű mátrix, akkor a* KOCKÁN-MIN- MATRIX *algoritmus M minmátrixát egy $\mathcal{H}_{3l}$ hiperkockán $O(\lg^2 n)$ lépésben meghatározza.*

Bizonyítás. Az algoritmus első szakaszában az üzenetszórás és a mátrixtranszponálás $O(\lg n)$ lépésben végrehajtható. A második szakaszban az $m[i, j]$ értékek frissítése prefixszámítással megoldható $O(lgn)$ lépésben. A négyzeten futó algoritmusban szereplő két üzenetszórás helyett itt a hiperkockában üzenetszórást végző algoritmust alkalmazzuk, ugyancsak $O(\lg n)$ lépésszámmal.

Tehát a **for** ciklus magjának minden végrehajtása $O(\lg n)$ lépést vesz igénybe. Mivel a ciklusmagot $\lg n$-szer kell végrehajtani, ebből a lépésszám felső korlátja már adódik.

Mivel például a második szakaszban végzett prefixszámítás lépésszáma $\Omega(n)$, ezért az elemzett algoritmus lépészámának nagyságrendje valóban $\lg^2 n$. ■

4.7.2. Tranzitív lezárt

Irányított gráf tranzitív lezártját a második fejezetben definiáltuk.

4.18. tétel. *Egy n-csúcsú irányított gráf tranzitív lezártja $O(\lg^2 n)$ lépésben meghatározható egy n^3 processzort tartalmazó hiperkockán.*

Bizonyítás. Állításunk a 4.17. tétel következménye. ■

4.7.3. Összefüggő komponensek

Gráfok összefüggő komponenseit a második fejezetben definiáltuk.

4.19. tétel. *Egy n csúcsú irányított gráf összefüggő komponensei egy n^3 processzort tartalmazó hiperkockán $O(\lg^2 n)$ lépésben meghatározhatók.*

Bizonyítás. Állításunk a 4.17. tétel következménye. ■

4.7.4. Legrövidebb utak

A HIPERKOCKÁN-UTAK algoritmus lépésszámára vonatkozik a következő tétel.

4.20. tétel. *A* HIPERKOCKÁN-UTAK *algoritmus egy n-csúcsú irányított gráf összes csúcspárjának távolságát meghatározza $O(\lg^2 n)$ lépésben egy $\frac{n^3}{\lg n}$ processzort tartalmazó hiperkockán.*

Bizonyítás. A PRAM modellek elemzése során megmutattuk, hogy egy gráf csúcsai közötti távolságokat $\lg n$ mátrixszorzás segítségével meghatározhatók.

Két $n \times n$ méretű mátrix egy $\frac{n^3}{\lg n}$ processzoros hiperkockán $O(\lg n)$ lépésben összeszorozható. ■

4.7.5. Konvex burok

A konvex burok számítására – hasonlóan a PRAM és négyzet modellek esetéhez – *oszd meg és uralkodj* elvű algoritmust javaslunk.

Ha a rekurzív algoritmusunk lépésszámát a d-dimenziós hiperkockán $K(d)$-vel jelöljük, akkor

$$K(d) = K(d-1) + O(d^2), \tag{4.12}$$

ahonnan $K(d) = O(l^3)$.

4.21. tétel. *Ha $n = 2^d$, akkor egy sík n pontjának konvex burka $O(\lg^3 n)$ lépésben meghatározható a $\mathcal{H}_d$ hiperkockán.*

Gyakorlatok

4.7-1. Állapítsuk meg, van-e Euler-kör és/vagy Hamilton-kör a vizsgált hálózatokban?

4.7-2. Mennyi idő alatt lehet egy üzenetet eljuttatni a fejezetben vizsgált hálózatok adott processzorától az összes többihez? Tervezzünk bejárási algoritmusokat és elemezzük lépésszámukat.

4.7-3. Mutassuk meg, hogy egy pillangó hálózat első szintjén lévő processzorból pontosan egy út vezet a d-edik szint bármelyik processzorához.

4.7-4. Foglaljuk táblázatba a vizsgált hálózatok jellemző tulajdonságait.

4.7-5. Hányféleképpen lehet beágyazni a 4.3. ábra bal oldalán látható G hálózatot az ábra jobb oldalán látható H hálózatba?

4.7-6. Adjuk meg a $\mathcal{G}_3$, $\mathcal{G}_4$ és $\mathcal{G}_5$ Gray-kódokat.

4.7-7. Tervezzünk $O(kd)$ lépésszámú algoritmust, amely 2^d k-bites kulcsnak a $\mathcal{B}_d$ pillangó hálózaton való rendezésére.

Feladatok

4-1. Gray-kód elemzése

Adjuk meg, összesen hány nullát és hány egyest tartalmaz $\mathcal{G}_k$ ($k \geq 1$). Hány olyan elrendezése van a k-jegyű bináris számoknak, melyekre jellemző, hogy az i-edik ($2 \leq i \leq 2^k$) szám pontosan egy helyen tér el az $(i-1)$-ediktől? Hány ilyen ciklikus elrendezés van?

4-2. Rács beágyazása hiperkockába

Tekintsük egy 4×8 méretű rácsnak $\mathcal{H}_5$-be ágyazását. Az első két bitet használjuk a rács sorainak, a további három bitet pedig a rács oszlopainak azonosítására: $\mathcal{G}_2$ feleljen meg a nulladik, első, második, illetve harmadik sornak. Ugyanígy $\mathcal{G}_3$ elemei rendre megfelelnek a 0., 1., $\ldots, (n-1)$. oszlopnak. Adjuk meg a beágyazás jellemző adatait (felfúvódás, késleltetés, torlódás).

4-3. Teljes bináris fa beágyazása de Bruijn-hálózatba

Mutassuk meg, hogy egy d szintes teljes bináris fa beágyazható egy d-dimenziós de Bruijn-hálózatba úgy, hogy a beágyazás késleltetése 1.

4-4. Mohó algoritmus korlátjának élessége

Lássuk be, hogy a mohó algoritmus lépésszámára és sorhosszúságára bizonyított felső korlát aszimptotikusan éles, azaz van csomagirányítási feladatoknak olyan sorozata amelyre a **??** egyenletben megadott nagyságrend a jellemző. *Útmutatás.* Vizsgáljuk meg azt az úgynevezett bitfordításos feladatot, ahol a $b_1 \ldots b_d$ sorbeli csomag címzettje a $b_d \ldots b_1$ sorbeli processzor. Vizsgáljuk meg ebben az esetben egy $(d/2)$-edik szintű él forgalmát.

4-5. Polinomok szorzása

Tervezzünk algoritmust, amely két 2^d-edfokú polinomot $O(d)$ idő alatt összeszoroz a $\mathcal{B}_d$ pillangó hálózaton.

4-6. Gyors Fourier-transzformáció

Tervezzünk algoritmust, amely a $\mathcal{B}_d$ pillangó hálózaton kiszámítja egy 2^d hosszúságú vektor gyors Fourier-transzformáltját.

Névmutató

Ez a névmutató

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, does**.

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.

Tartalomjegyzék