5. Szinkronizált hálózat

Ebben a fejezetben döntési feladatokat oldunk meg – speciális és általános hálózatokban.

5.1. Számítási modell

A H hálózatot a korábbiakhoz hasonlóan $H = (V, E)$ formában adjuk meg, ahol $V = \{P_1,\ P_2,\ \ldots,\ P_p\}$ a processzorok halmaza. Két processzor között

- vagy kétirányú adatátvitel lehetséges,
- vagy csak egyirányú adatátvitel van,
- vagy nincs kapcsolat.

Ennek megfelelően az élek E halmaza két részből áll: $E = (S, D)$, ahol S az egyirányú adatátviteli vonalakat leíró irányított élek halmaza, míg D a kétirányú adatátvitelt leíró irányítatlan élek halmaza.

A keverő-cserélő hálózatokban kétféle él van: kétirányú ***cserélő*** és egyirányú ***keverő*** él.

A d dimenziós ***teljes keverő-cserélő*** hálózatban $p = 2^d$ processzor van. A cserélő élek a P_{2i} processzorból a P_{2i+1} $(i = 0,\ 1,\ \ldots,\ 2^{d-1} - 1)$ processzorhoz vezetnek. Minden processzorból egy keverő él indul: a P_i $(i = 0,\ 1,\ \ldots\ 2^p - 1)$ processzorból induló keverő él a P_{2i} processzornál végződik, ahol az indexeket (mod $2^p - 1$) vesszük.

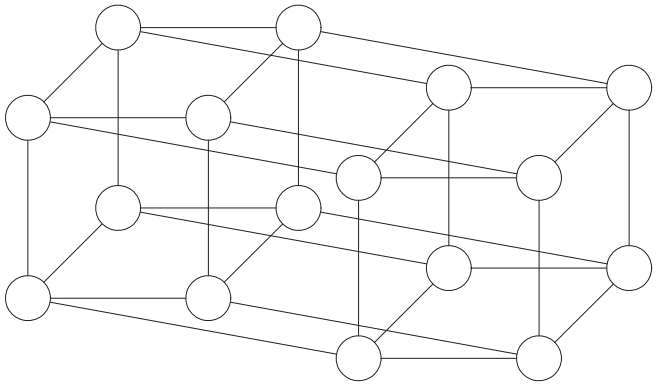
A 5.1. ábra egy 8-processzoros teljes keverő-cserélő hálózatot ábrázol.

A gyakorlatban használt hálózatok többségében csak kétirányú adatátviteli vonalak vannak. A de Bruijn-hálózat csak egyirányú adatátvitelt enged meg.

A P_i processzor ***szomszédait*** *szomszéd*$[i]$-vel jelöljük és a következőképpen definiáljuk:

$$szomszéd[i] = \{P_j \mid (i,j) \in S \vee (i,j) \in D \vee (i,j) \in D)\}qkoz. \tag{5.1}$$

A processzorokat automataként írjuk le, amelyek a szinkronizált lépésekben üzeneteket küldhetnek és kaphatnak, és adott kezdőállapotból kiindulva minden lépésben – a beérkező üzenetek és a korábbi állapot által meghatározott – új állapotba mennek át.



5.1. ábra. 8 processzoros keverő-cserélő hálózat.

A P_i processzor a KÜLD$_i$(*üzenet*) és a FOGAD$_i$(*üzenet*) függvénnyel küldenek, illetve fogadnak üzenetet. Az *üzenet* az $\ddot{U}\epsilon$ halmaz eleme, ahol $\ddot{U}$ a lehetséges üzenetek halmaza, ϵ pedig az üres üzenet. Egy üzenet lehet például a küldő folyamat azonosítója, és állhat több részből is.

Ebben a fejezetben a futási idő mellett az elküldött és fogadott üzenetek száma is gyakran használt hatékonysági jellemző.

5.2. Vezető választása

Ennek az alfejezetnek a témája az egyik legfontosabb döntési feladat, a ***vezetőválasztás.*** Tegyük fel, hogy kezdetben minden processzor azonos állapotban van. A cél olyan állapot elérése, amelyben pontosan egy processzor a *vezető* a többi processzor pedig a *nem_vezető* állapotban van. A későbbiekben (főleg az algoritmusok leírásában) használjuk a rövidebb *vez*, illetve *nem_vez* jelölést is.

Számos feladat megoldásához szükség van a processzorok szimmetriájának megtörésére, és egy ***vezető processzor***megválasztására. Ezt a feladatot LeLann fogalmazta meg 1977-ben.

Először megmutatjuk, hogy a vezetőválasztás bizonyos körülmények között ***megoldhatatlan feladat.***

Azután algoritmusokat mutatunk be és elemzünk, amelyek gyűrűben, fában és általános hálózatban megoldják a vezetőválasztást.

5.2.1. Vezetőválasztás megoldhatatlansága gyűrűben

Legyen H egy p processzoros gyűrű. Ha H-ban minden processzor azonos kezdeti állapotban van, akkor nincs mód ennek a kezdeti szimmetriának a megszüntetésére. Ezt az állítást formalizálja a következő tétel.

5.1. tétel (vezetőválasztás gyűrűben). *Ha G egy egyirányú vagy kétirányú gyűrű,*

melyben a processzorok kezdeti állapota, állapotátmeneti függvénye és üzenet-előállító függvénye is azonos, akkor ebben a gyűrűben a vezetőválasztás nem oldható meg.

Bizonyítás. Az állítást indirekt módon bizonyítjuk. Tegyük fel, hogy a P algoritmus megoldja a feladatot.

Feltehetjük, hogy a gyűrű minden processzorának csak egy kezdőállapota van (ha több van, közülük tetszőlegesen választva elérhetjük, hogy minden processzornak csak egy kezdőállapota legyen).

Az első lépésben minden processzor ugyanazt az üzenetet küldi szomszédjának (kétirányú gyűrűben mindkét szomszédjának), ezért a második lépésben a processzorok azonos új állapotba mennek át és azonos üzenet küldenek szomszédjuknak.

A lépések száma szerinti indukcióval adódik, hogy ha bármely processzor állapota *vezető*, akkor a többi processzor állapota is *vezető* lesz, ami nem biztosítja a vezetőválasztás egyértelműségét. ■

A gyakorlatban rendszerint nem azonos a processzorok kezdőállapota. A továbbiakban feltesszük, hogy minden processzornak egyedi ***azonosítója*** van, amely a többi processzortól megkülönbözteti.

5.2.2. Vezetőválasztás gyűrűben

Az első gyűrűs vezetőválasztó algoritmus LeLann nevéhez fűződik. A LeLann algoritmusra jellemző, hogy a szükséges üzenetek száma négyzetesen nő a processzorok számával. Chang és Roberts 1979-ben olyan javítást dolgoztak ki, amely legrosszabb esetben továbbra is négyzetes volt, de átlagos esetben már $O(n \lg n)$ lépésben megoldotta a vezetőválasztást. 1980-ban Hirschberg és Sinclair olyan megoldást talált, melyre a legrosszabb esetben is bizonyítani tudták az $O(n \lg n)$ felső korlátot. Igaz, míg a korábbi módszerek egyirányú gyűrűben is működtek, a Hirschberg-Sinclair algoritmusnak kétirányú adatátviteli vonalakra van szüksége.

Az alsó korlátokkal kapcsolatos eredmények szerint a vezetőválasztást aszimptotikusan optimálisan is meg tudjuk oldani.

LeLann algoritmusa

A LeLann algoritmus megengedi, hogy a vezető az úgynevezett *kezdő processzorok* közül kerüljön ki. A kezdő processzorok halmazát K-val jelöljük. A processzoroknak nincs szüksége arra, hogy ismerjék a hálózat méretét.

Az algoritmus pszeudokódja a következő.

LeLann(K, A) *párhuzamos eljárás*
Számítási modell: egyirányú gyűrű
Bemenet: K (a kezdő processzorok indexeinek halmaza), $A[1:n]$ (a processzorok azonosítóinak tömbje, amely különböző egész számokat tartalmaz)
Kimenet: i (a vezető processzor indexe)

```
01  P_i in parallel for i ← 1 to n
02        do if i ∈ K
```

```
03              then áll[i] ← jelölt
04                   J_i ← {i}
05              else áll[i] ← n_vez
06  P_i in parallel for i ← 1 to n
07        do if áll[i] = jelölt
08              then KÜLD_i(i)
09              while áll[i] ≠ vez
10                   FOGAD_i(a)
11                   J_i ← J_i ∪ {a}
12                   KÜLD_i(a)
13              if i = min{J_i}
14                 then áll[i] ← vez
15                 else áll[i] ← n_vez
16              while áll[i] ≠ vez
17                   FOGAD_i(a)
18                   KÜLD_i(a)
```

Az algoritmus szerint először az 1–6. sorokban beállítjuk a processzorok állapotát: *jelölt* lesz a kezdő processzorok állapota és *nem_jelölt* lesz a többi processzor kezdőállapota. A kezdő processzorok a jelöltek J_i halmazába beteszik saját azonosítójukat.

A 7–13. sorokban a kezdő processzorok addig fogadják és küldik az üzeneteket, amíg saját azonosítójukat – amely körbeért a gyűrűn – vissza nem kapják.

A 14–16. sorokban a legkisebb azonosítójú processzor *vez*-re, a többi kezdő processzor *n_vez*-re állítja a saját állapotát.

A nem kezdő processzorok szerepe az üzenetek továbbítása (17–21. sorok).

5.2. tétel. *A* LeLann *algoritmus egy egyirányú gyűrűn minden esetben* $\Theta(p)$ *lépésben és legrosszabb esetben* $O(p^2)$ *üzenetet küldve oldja meg a vezetőválasztást.*

Bizonyítás. Az 1–6. sorokban két lépésben beállítjuk $áll[i]$ és J_i értékét. A kezdő processzorok a p. lépésben visszakapják saját azonosítójukat (a többi kezdő processzor azonosítóját már korábban megkapták). Ekkor a legkisebb azonosítójú processzor állapota a 14–15. sor szerint *vezető* lesz, a többi kezdő processzor állapota pedig a 16. sorban *nem_vez* lesz.

Mivel legfeljebb p különböző azonosító van és mindegyik p lépést tesz, ezért az elküldött és fogadott üzenetek száma $O(p^2)$. Mivel legrosszabb esetben minden processzor kezdő, ezért az üzenetek száma $W(p, \text{LeLann}) = \Theta(p^2)$. Mivel a kezdő folyamatok azonosítója p lépés alatt ér körbe a gyűrűn, ezért az algoritmus lépésszáma minden esetben $p = \Theta(p)$. ∎

A LeLann algoritmus biztosítja, hogy a processzorok a megfelelő állapotba kerüljenek, de nem biztosítja azt, hogy a nem kezdő processzorok megálljanak. Ezt a megállást például úgy biztosíthatjuk, hogy a vezetőnek választott processzor körbeküld egy ***értesítő üzenetet.*** Az így kiegészített algoritmusra is érvényes a $W(p, \text{Értesít-LeLann}) = O(p)$ és $W_{ü(p,\text{Értesít-LeLann})} =$

Chang és Roberts algoritmusa

Chang és Roberts azzal javították az előző algoritmust, hogy csökkentették a feleslegesen továbbküldött azonosítók számát: a kezdő processzorok csak a saját azonosítójuknál kisebb azonosítókat küldik tovább.

CHANG-ROBERTS(U, i) *párhuzamos eljárás*
Számítási modell: egyirányú gyűrű
Bemenet: $U = u_1,\ u_2,\ \ldots,\ u_p$ (a processzorok azonosítói – különböző egész számok)
Kimenet: i (a vezető processzor indexe)

```
01  P_i in parallel for i ← 1 to n
02                      if i ∈ K then
03                                  áll[i] := jelölt
04                                  L_i := {i}
05                       else
06                                  áll[i] := nem_jelölt
07  P_i in parallel for 1 ≤ i ≤ n
08        if áll[i] = jelölt then
09                          KÜLD_i(i)
10                          while j ≠ i
11                                  FOGAD_i(a)
12                                  L := L ∪ {j}
13                                  KÜLD_i(a)
14                          if i = min{K} then
15                                              áll[i] := vez
16                                              else áll[i] := n_vez
17                          else
18                               while áll(i) ≠ vez
19                                              FOGAD_i(j)
20                                              KÜLD_i(j)
```

5.3. tétel. *A* CHANG-ROBERTS *algoritmus* $\Theta(p)$ *lépéssel és legrosszabb esetben* $\Theta(p^2)$ *üzenettel oldja meg egyirányú gyűrűben a vezetőválasztást. Az algoritmus átlagos üzenetszáma* $O(p \lg p)$.

Bizonyítás. A legrosszabb esetre vonatkozó bizonyítás hasonló a LELANN algoritmusra vonatkozó bizonyításhoz.

Az átlagos üzenetszámmal kapcsolatban legyen s a legkisebb a p azonosító közül. p különböző azonosítónak $(p-1)!$ különböző ciklikus permutációja van. Adott ciklikus permutációban legyen a_i az az azonosító, amely i lépéssel halad s előtt.

Mivel az s azonosító minden permutációban p lépést tett meg, ezért a $(p-1)!$ ciklikus permutációban összesen $p(p-1)!$ lépést tett meg. Az a_i azonosítót legfeljebb i-szer kellett továbbítani, mivel eldobjuk, amikor eléri az s azonosítójú processzort. Legyen $A_{i,k}$ azoknak a ciklikus permutációknak a száma, amelyekben az a_i azonosí-

tót pontosan k-szor kellett továbbítani. Ekkor az a_i azonosítót összesen

$$\sum_{k=1}^{i} kA_{i,k} \tag{5.2}$$

alkalommal kell továbbítani.

Ha a_i a legkisebb az $a_1,\ a_2,\ \ldots, a_i$ azonosítók között – ami $(p-1)!/i$ permutációban fordul elő – akkor az a_i azonosítót pontosan i-szer kell továbbítani, ezért

$$A_{i,i} = \frac{(p-1)!}{i}\ . \tag{5.3}$$

Ha az a_i azonosítót $k-1$ olyan azonosító követi, amelyek nagyobbak, mint a_i, akkor a_i-t legalább k-szor kell továbbítani (itt $k \le i$). Azoknak a ciklikus permutációknak a száma, amelyekben a_i a legkisebb az $a_{i-k+1},\ a_{i-k+2},\ \ldots,\ a_i$ azonosítók között, $(p-1)!/k$. Ezért ha $k < i$, akkor az a_i azonosítót

$$\frac{(p-1)!}{k} - \frac{(p-1)!}{k+1} \tag{5.4}$$

permutációban kell pontosan k-szor továbbítani, és így

$$A_{i,k} = \frac{(p-1)!}{k(k+1)} \quad (\text{ha } k < i). \tag{5.5}$$

Ezért az a_i azonosítót az összes ciklikus permutációban összesen

$$\sum_{k=1}^{i-1} k\left(\frac{(p-1)!}{k(k+1)}\right) + i\frac{1}{i}(p-1)! = (p-1)!\sum_{k=1}^{i}\frac{1}{k} \tag{5.6}$$

alkalommal kell továbbítani.

Ismert, hogy az egyenlőség jobboldalán lévő szumma a H_i *harmonikus szám*, amelyre

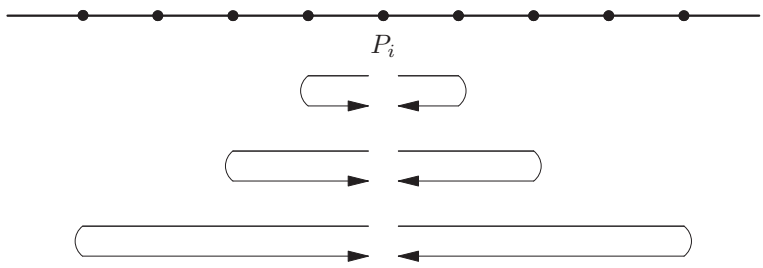
$$\sum_{i=1}^{m} H_i = (m+1)H_m - m\ . \tag{5.7}$$

Most összegezzük az s-től különböző i azonosítók által megtett lépések számát:

$$\sum_{i=1}^{p-1}\lfloor (p-1)!H_i \rfloor = (pH_{p-1} - (p-1))(p-1)!\ . \tag{5.8}$$

Mivel ez a lépésszám az összes ciklikus permutációhoz tartozik, ezért az átlag pH_p. Mivel $H_p = \ln p + O(1)$, azt kaptuk, hogy az átlag valóban $O(p \lg p)$. ■

Ha az azonosítók kezdeti permutációja kedvező – például $a_1 > a_2 > \ldots > a_p$ – akkor az a_j $(1 \le j \le p-1)$ azonosító csak egy lépést tesz meg, ezért $B_{\ddot{u}}(p, \textsc{Chang-Roberts}) = O(p)$.



5.2. ábra. A Hirschberg-Sinclair algoritmus szemléltetése.

Hirschberg és Sinclair algoritmusa

Az eddig tárgyalt vezetőválasztó algoritmusok kevés lépést tesznek, de sok üzenetre van szükségük. Most egy olyan algoritmust mutatunk be, amelynek a korábbinál lényegesen kevesebb üzenetet igényel.

Hirschberg és Sinclair algoritmusa is a legnagyobb azonosítóval rendelkező folyamatot választja vezetőnek. Itt azonban az azonosítók nem körbejárják a gyűrűt, hanem bizonyos (egyre nagyobb) lépés megtétele után visszafordulnak. Ezt szemlélteti a 5.2. ábra.

5.4. tétel. *A* Hirschberg-Sinclair *algoritmus üzenetszáma kétirányú gyűrűben* $W(p, \text{Hirschberg-Sinclair}) = O(p \lg p)$

Idő-szelet algoritmus

Az eddigi vezetőválasztó algoritmusok az azonosítók összehasonlításával jutottak információhoz.

A következő Idő-szelet algoritmus nagyon kevés üzenetet használ. Az algoritmus szakaszokban működik, és minden szakasz p lépésből áll. A j-edik szakaszban csak j azonosítót lehet üzenetként elküldeni. Ha a P_i processzor azonosítója a_i, akkor ez a processzor az $1., 2., \ldots, (a_i - 1)$. szakaszban nem küld üzenetet. Ha a P_i processzor az első $a_i - 1$ szakaszban nem kap üzenetet, akkor az a_i-edik szakasz első lépésében elküldi szomszédjának a saját azonosítóját, és ez az azonosító körbemegy az egyirányú gyűrűn.

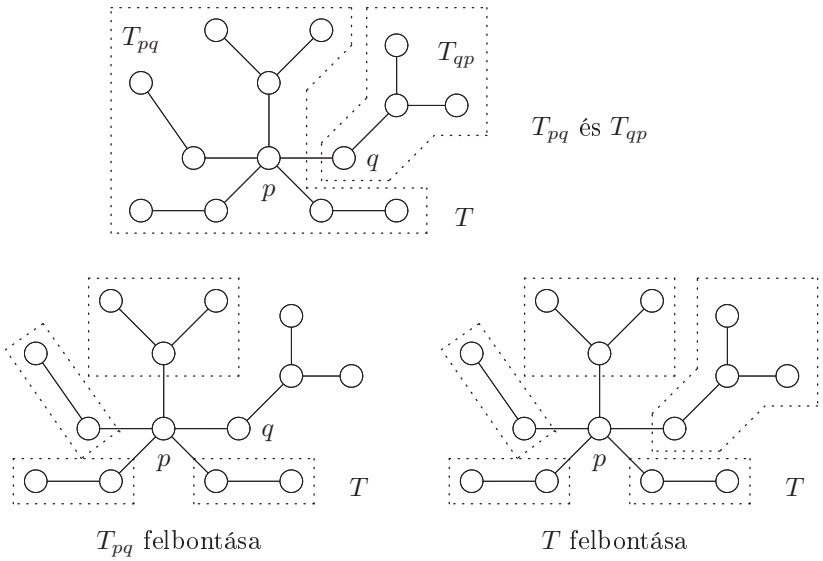
5.5. tétel. *Az* Idő-szelet *algoritmus egy p-processzoros egyirányú gyűrűben p üzenettel $\Theta(pa_{min})$ lépésben oldja meg a vezetőválasztást.*

Ennek a tételnek közvetlen következménye, hogy az Idő-szelet algoritmus üzenetszámát tekintve aszimptotikusan optimális.

Alsó korlát az üzenetszámra

Az összehasonlítás alapú vezetőválasztó algoritmusok üzenetszámára érvényes a következő alsó korlát.

5.6. tétel. *Ha a $\mathcal{P}$ algoritmus bármely p-processzoros gyűrűben vezetőt tud választani, akkor megadható p darab különböző azonosító olyan permutációja, amelyre az $\mathcal{A}$ algoritmus $N_{ü}(p, P) = \Omega(p \lg p)$ üzenetet küld.*



5.3. ábra. T_{pq} részfái.

5.2.3. Vezetőválasztás fában

Az alábbi algoritmus fában megoldja a vezetőválasztást.

FÁBAN-VEZETŐ(A, i) *párhuzamos eljárás*
Számítási modell: gyűrű
Bemenet: $A[1:p]$ (a processzorok azonosítói, különböző egészek)
Kimenet: i (a vezető processzor indexe)

```
01  P_i in parallel for i ← 1 to n
02        if i ∈ K
03           then áll[i] ← jelölt
04           else áll[i] ← n_vez
```

Az algoritmus menetét illusztrálja az 5.3. ábra. Az ábra felső része a T fának a T_{pq} és T_{qp} részfákra bontását mutatja. Az ábra bal alsó része a T_{pq} fa felbontását mutatja.

5.7. tétel. *A* FÁBAN-VEZETŐ *algoritmus egy p-processzoros fában $O(p)$ üzenettel és $O(tmr)$ lépésben megoldja a vezetőválasztást.*

5.2.4. Vezetőválasztás általános hálózatban

Általános hálózatban először egy egyszerű üzenetterjesztő algoritmust, majd annak javított változatát mutatjuk be.

Max-terjed algoritmus

A Max-terjed algoritmus alapötlete, hogy a processzorok minden menetben elküldik szomszédaiknak az addig hozzájuk eljutott legnagyobb azonosítót.

5.8. tétel. *Ha egy tetszőleges H hálózat átmérője $átm(H)$, akkor a* Max-terjed *algoritmus ebben a hálózatban legfeljebb $átm(H)$ menetben a legnagyobb azonosítójú folyamatot vezetővé választja. Az elküldött üzenetek száma pedig $O(|E|tm)$.*

Opt-max-terjed algoritmus

Általános hálózatban is alkalmazható az a javítás, amit már a gyűrű esetében láttunk: a processzorok csak akkor küldenek tovább azonosítót, ha az új információt tartalmaz.

Ezzel ugyan a legrosszabb esetben szükséges üzenetek számának nagyságrendje változatlan marad, az üzenetek átlagos száma azonban lényegesen csökken.

5.9. tétel. *Ha egy tetszőleges H hálózat átmérője $átm(H)$, akkor az* Opt-max-terjed *algoritmus ebben a hálózatban legfeljebb $átm(H)$ menetben vezetővé választja a legnagyobb azonosítójú folyamatot.*

5.2.5. Alsó korlát az üzenetek számára

Az általános hálózatokban szükséges üzenetek számára vonatkozik a következő tétel.

5.10. tétel. *Ha H egy p processzort tartalmazó hálózat, akkor a vezetőválasztás ebben a hálózatban*

$$N(p) \geq pH_p \tag{5.9}$$

üzenetet igényel.

Ebből a tételből adódik a következő állítás.

5.11. következmény. *A* Chang-Roberts *algoritmus üzenetszáma aszimptotikusan optimális.*

Bizonyítás. A 5.3. tétel szerint az algoritmus üzenetszámára

$$W(p) = O(p \lg p). \tag{5.10}$$

Mivel $H_p = \Theta(\lg p)$, így $\Theta(W(p)) = N(p)$. ∎

5.3. Megegyezés

A következő döntési feladat a megegyezés. Tegyük fel, hogy kezdetben minden P_i processzor rendelkezik egy b_i bemeneti értékkel, és az a cél, hogy a processzorok azonos k kimenő értékre jussanak.

Ezt a problémát mind az üzenetek egy részének elvesztését, mind a processzorok

hibáját megengedve is szokták vizsgálni.

A ***k-megegyezés*** problémája az egyszerű megegyezési probléma természetes általánosítása: a processzorok feladata az, hogy a bemenő értékek egy k-elemű részhalmazából válasszanak közösen elfogadott értéket.

5.3.1. Megegyezés vonalhibák esetében

A probléma lényegét jól tükrözi az ***összehangolt támadási feladat*** Eszerint tábornokok összehangolt támadást terveznek közös célpont ellen. A tábornokok hírnökök segítségével válthatnak üzenetet.

Feltesszük, hogy a tábornokok egy G irányítatlan (nem teljes) gráf csúcsaiban vannak, és az élek mentén küldhetnek üzenetet. Megbízható élekkel $G_{átm}$ lépésben minden tábornok teljes információval rendelkezik a többiek véleményéről, és a katonai akadémián tanultak alapján ugyanarra a döntésre juthatnak.

Ha azonban az élek meghibásodhatnak, ez az egyszerű gondolatmenet nem alkalmazható, a probléma nem oldható meg (ennek belátását meghagyjuk gyakorlatnak).

Hibás élek esetén csak az a reális célkitűzés, hogy megadott valószínűséggel jussanak a tábornokok közös véleményre. A problémának ez a változata már determinisztikus és véletlenített algoritmussal is kezelhető.

5.3.2. Megegyezés processzorhibák esetében

A processzorok működése során különböző ***hibák*** fordulhatnak elő. Az egyik a ***megállási hiba,*** melyben a processzor bármely lépésben beszüntetheti működését. A másik a ***bizánci hiba,*** melyben a processzorok a számukra megadott korlátokon (elvégezhető műveletek, felhasználható üzenetábécé) belül tetszőlegesen működhetnek.

Ennek a problémának egy egyszerű megoldását biztosítja a HALMAZ-TERJED algoritmus. Ennek lényege, hogy a processzorok türelmesen terjesztik a tudomásukra jutott összes információt – és ha bizonyos ideig nem kapnak új információt, akkor az addig kapott üzenetek alapján döntenek.

Ha a processzorok értékelik is a beérkezett információt és csak a lényeges részt adják tovább, akkor az elküldendő üzenetek száma csökkenthető. Így jutunk az OPT-HALMAZ-TERJED algoritmushoz.

5.3.3. k-megegyezés

A k-megegyezési feladatnál a processzoroknak a bemeneti értékek k-elemű részhalmazából kell közösen elfogadott értéket választaniuk.

Ezt a feladatot például a MIN-TERJED algoritmussal lehet megoldani. Ennek lényege, hogy a processzorok karbantartják és terjesztik az addig kapott legkisebb értéket. Erről az algoritmusról belátható, hogy ha legfeljebb h processzor hibásodhat meg, akkor $\lfloor h/k+1 \rfloor$ lépésben megoldja a feladatot.

A következő alsó korlát ismert.

5.12. tétel. *Ha $p \geq h+k+1$, akkor minden algoritmusnak legalább $\lfloor h/k+1 \rfloor$ lépésre van szüksége, hogy h hibás processzor esetén megoldja a k-megegyezési feladatot.*

5.3.4. Közelítő megegyezés

A ***közelítő megegyezési feladatban*** minden processzornak van egy valós kezdeti értéke és a processzorok valós értékeket küldenek egymásnak és egymástól kevéssé eltérő értékekben kell megegyezniük.

Megengedjük, hogy a processzorok bizáci hibákat kövessenek el.

3 feltételnek kell teljesednie.

A ***befejeződési feltétel*** szerint minden hibátlanul működő processzornak végül döntést kell hoznia.

Az ***érvényességi feltétel szerint*** a hibátlanul működő processzoroknak a hibátlan processzorok kezdeti értékeit tartalmazó (lehető legrövidebb) intervallumból vett értékkel kell megállniuk.

A ***megegyezési feltétel*** szerint akármely két hibátlanul működő processzor kimenő értéke legfeljebb egy előre adott ϵ értékkel térhet el egymástól.

Ismertek olyan algoritmusok, amelyek teljes hálózatban biztosítják a közelítő megegyezést, ha a hibás processzorok száma kisebb, mint az összes processzorok számának egy harmada.

Gyakorlatok

5.3-1. Elemezzük a LeLann és a Chang-Roberts algoritmusokat.

a. Adjuk meg az azonosítók olyan permutációját, amelyre az elküldött üzenetek száma $\Omega(n^2)$.

b. Adjuk meg az azonosítók olyan permutációját, amelyre az elküldött üzenetek száma $O(n)$.

5.3-2. Módosítsuk az Chang-Roberts algoritmust úgy, hogy az összes nem-vezető folyamat a *em nem_vezető* kimenetet eredményezze, vagyis az összes folyamat végül is álljon meg. Adjuk meg a módosított algoritmus pszeudokódját.

5.3-3. Mutassuk meg, hogy a Chang-Roberts algoritmus különböző induló időpontok mellett is helyesen működik. (Ehhez módosítsuk a kódot.)

5.3-4. Bizonyítsuk be a LeLann és a Chang-Roberts algoritmusok helyességét.

5.3-5. Mutassuk meg, hogy a Hirschberg-Sinclair algoritmus különböző induló időpontok mellett is helyesen működik. (Ehhez egy kicsit módosítsuk a pszeudokódot.)

5.3-6. Tegyük fel, hogy a Hirschberg-Sinclair algoritmust úgy módosítjuk, hogy kettő-hatványok helyett egymás utáni k-hatványokat használunk az utak hosszára ($k > 2$). Elemezzük a módosított algoritmus lépésszámát és kommunikációs bonyolultságát úgy, mint az eredeti Hirschberg-Siclair algoritmusnál. Hasonlítsuk össze az eredményeket.

5.3-7. Tekintsük a Hirschberg-Sinclair algoritmus olyan módosított változatát, ahol a processzorok mindkét irány helyett csak az egyik irányba küldhetnek üzeneteket.

a. Mutassuk meg, hogy a könyvben megadott algoritmus legkézenfekvőbb módosítása nem eredményez $O(n \log n)$ üzenetszámot. Adjunk felső korlátot az üzenetszámra.

b. Módosítsuk úgy az algoritmust, hogy az üzenetszáma $O(n \log n)$ legyen.

5.3-8. Tervezzünk egyirányú gyűrűben olyan vezetőválasztó algoritmust, amely nem ismeri a gyűrű méretét és legrosszabb esetben is csak $O(n \log n)$ számú üzenetet használ. Az algoritmus az azonosítókra kizárólag az összehasonlítás műveletet használhatja.

5.3-9. Adjunk a menetek számára vonatkozó minél jobb *alsó* korlátot valamely n méretű gyűrű vezető folyamat kiválasztásos algoritmusának legrosszabb esetére. A feltevéseket körültekintően fogalmazzuk meg.

5.3-10. Adjuk meg az $n = 16$ csúcsú bitfordító gyűrű pontos leírását.

5.3-11. Bizonyítsuk be, hogy az $n = 2^k$ méretű bitfordító gyűrű minden $k \in \mathbb{N}$ esetén 1/2-szimmetrikus.

5.3-12. Tervezzünk c-szimmetrikus gyűrűt nem kettő-hatvány számú csúcs esetén valamilyen $c > 0$ értékre.

5.3-13. Valamely szinkron gyűrű esetén tekintsük a vezető folyamat kiválasztásának problémáját, ahol minden folyamat ismeri a gyűrű n méretét és a processzoroknak nincs azonosítójuk. Adjunk a probléma megoldására *véletlenített algoritmust,* vagyis olyat, ahol a processzorok kódjuk determinisztikus végrehajtásán kívül véletlen választással is élhetnek. A helyes működést kielégítő tulajdonságokat óvatosan fogalmazzuk meg. Például az egyedi vezető folyamat kiválasztása biztosan garantált-e vagy valamilyen kis valószínűséggel elképzelhető, hogy ez nem történik meg? Mennyi lesz az algoritmus lépésszáma és üzenetszáma?

5.3-14. Tekintsünk valamilyen ismeretlen n méretű kétirányú gyűrűt, ahol a processzoroknak van egyedi azonosítójuk. Adjunk az üzenetek számára vonatkozó alsó és felső korlátot olyan összehasonlítás alapú algoritmus esetén, ahol minden processzor mod 2 számolja ki n-et.

5.3-15. A MAX-TERJED algoritmusban használt üzenetek *átm*$|E|$ száma $O(n^3)$. Adjunk meg olyan irányított gráfokat, amelyekre az *átm*$|E|$ szorzat $\Omega(n^3)$, vagy mutassuk meg, hogy nincs ilyen irányított gráf.

5.3-16. Az OPT-MAX-TERJED algoritmus által elküldött üzenetek számára adjunk a $O(n^3)$-nál kisebb felső korlátot vagy mutassuk meg, hogy a korlát aszimptotikusan éles.

5.3-17. Elemezzük a vezetőválasztás lépésszámát és üzenetszámát, feltéve, hogy néhány szomszédos csúcs között kétirányú kommunikációt is megengedünk.

5.3-18. Tervezzünk egy vezetőválasztó algoritmust egy olyan erősen összefüggő irányított hálózatban, amelyben a processzoroknak van egyedi azonosítójuk.

a. Tegyük fel, hogy a kommunikáció a szomszédos csúcsok között kétirányú.

b. Ne alkalmazzuk az előző feltevést.

5.3-19. Adjunk algoritmust a csúcsok számának megállapítására egy olyan erősen összefüggő irányított hálózatban, amelyben a processzoroknak van egyedi azonosítójuk.

a. Tegyük fel, hogy a kommunikáció a szomszédos csúcsok között kétirányú.

b. Ne alkalmazzuk az előző feltevést.

5.3-20. Adjunk algoritmust az élek számának megállapítására egy olyan erősen összefüggő irányított hálózatban, amelyben a processzoroknak van egyedi azonosítójuk.

a. Tegyük fel, hogy a kommunikáció a szomszédos csúcs közt kétirányú.

b. Ne alkalmazzuk az előző feltevést.

5.3-21. Tegyük fel, hogy egy láncban minden P_i processzor meg tudja különböztetni a bal oldalát a jobb oldalától, és ismeri azt is, hogy ő maga végpont-e vagy sem. Tegyük fel, hogy minden processzor kezdetben egy nagyon nagy a_i egész értékkel rendelkezik, és azt, hogy az ilyen értékekből egy adott időpillanatban csak adott számút tarthatunk nyilván a memóriában. Tervezzük meg azt az ezen értékeket sorba rendező algoritmust, amelyben az egyes P_i processzorok által előállított o_i kimeneti értékek összeszorzott halmaza megegyezik az a_i bemeneti értékek összeszorzott halmazával, és $o_1 \leq \ldots \leq o_n$. Próbáljuk meg előállítani mind az üzenetek, mind a menetek száma tekintetében a leghatékonyabb algoritmust.

5.3-22. Mutassuk meg, hogy az összehangolt támadási probléma (determinisztikus változatának) megoldása bármely nem triviális, összefüggő gráf esetében magában foglalja a probléma megoldását arra az egyszerű, két pontból álló gráfra, mely egy éllel van összekötve. (Ebből következik, hogy a probléma megoldhatatlan tetszőleges, nem triviális gráf esetében.)

5.3-23. Tekintsük a (determinisztikus) összehangolt támadási probléma következő változatát. Tegyük fel, hogy a hálózat $n > 2$ résztvevőből álló teljes gráf. A befejezési és érvényességi feltételek az 5.3. alfejezetben leírtakkal azonosak. A megegyezési feltételt azonban gyengítjük: „Ha van olyan a folyamatok között, amelyik döntése 1, akkor legalább kettőnek 1-est kell döntenie.” (Azaz szeretnénk kizárni azt az esetet, amikor egy tábornok magányosan támad, de megengedjük azt, hogy két vagy több tábornok együtt támadjon.) Vajon ez a probléma megoldható, vagy nem?

5.3-24. Tekintsük az összehangolt támadási problémát vonalhibák esetében arra az egyszerű esetre, amikor két folyamat egy éllel van összekötve. Tegyük fel, hogy a processzorok működése determinisztikus, de az üzenetrendszer véletlenített abban az értelemben, hogy mindegyik üzenetnek van egy független q valószínűségi értéke ($0 < q < 1$), ami annak a valószínűségét adja meg, hogy az üzenet sikeresen megérkezik. (Ahogy általában, most is megengedjük, hogy a folyamatok menetenként csak egy üzenetet küldjenek.) Tervezzünk ezekkel a beállításokkal olyan algoritmust, mely rögzített r számú meneten belül befejeződik, a megegyezés hiányának valószínűsége legfeljebb ϵ, és ehhez hasonlóan az érvényességi feltétel megsértésének valószínűsége is legfeljebb ϵ. A lehető legkisebb ϵ érték elérésére törekedjünk.

5.3-25. Az előző gyakorlat kikötései szerinti modellben adjunk alsó korlátot az ϵ értékére, bizonyítsuk be, hogy ez az elérhető legalacsonyabb érték.

5.3-26. Általánosítsuk az összehangolt támadási probléma véletlenített változatát úgy, hogy megengedjük ϵ valószínűséggel mind az érvényességi, mind a megegyezési szabályok megsértését. Írjuk át a VÉLETLENÍTETT-TÁMADÁS algoritmust úgy, hogy a módosított feltételek mellett elérje a lehető legkisebb ϵ értéket. Végezzünk elemzést.

5.3-27. Általánosítsuk a VÉLETLENÍTETT-TÁMADÁS algoritmust és az elemzését az általános irányítatlan gráfokra.

5.3-28. Mi történne a fejezetben tárgyalt, véletlenített környezettel kapcsolatos eredményekkel, ha az ellenfél kommunikációs mintája nem lenne előre rögzítve, mint ahogy eddig feltettük, hanem az ellenfél közvetlen irányítással határozhatná meg azt. Pontosabban szólva, tegyük fel, hogy az ellenfél képes arra, hogy megvizsgálja a végrehajtási sorozatot bármely k-adik menettől visszafelé a kezdetig, mielőtt döntene,

hogy a k-adik menetbeli üzenetek közül melyek legyenek kézbesítve.

a. Milyen ϵ korlát garantálható a VÉLETLENÍTETT-TÁMADÁS algoritmus esetében a megegyezés hiányára, ilyen közvetlen irányításra képes ellenfelek esetében?

b. Adhatunk-e valamilyen érdekes alsó korlátot az elérhető ϵ értékekre?

5.3-29. Bizonyítsuk be, hogy tetszőleges olyan algoritmus, mely megoldja a bizánci megegyezés problémát, megoldja a megegyezési problémát megállási hibák esetében is, ha a megállási hiba modellben úgy módosítjuk az érvényességi feltételt, hogy csak a hibamentes folyamatok megegyezését követeljük meg.

5.3-30. Bizonyítsuk be, hogy tetszőleges olyan algoritmus, mely megoldja a bizánci megegyezés problémát és amelyben a hibátlan folyamatok mindig egyszerre, ugyanazon menetben hoznak döntést, megoldja a megegyezési problémát megállási hiba modellben is.

5.3-31. Kövessük nyomon a HALMAZ-TERJED algoritmus végrehajtását négy folyamattal és két hibával, melyben a folyamatok kezdőértékei rendre az 1, 0, 0, 0 értékek. Tegyük fel, hogy P_1 és P_2 folyamatok hibásak, P_1 az első menetben lesz hibás, miután egyedül a P_2 folyamatnak elküldte az üzenetet, P_2 pedig a második menetben lesz hibás, P_1-nek és P_3-nak küld üzenetet, viszont P_4-nek nem.

5.3-32. Tekintsük a HALMAZ-TERJED algoritmust f hibára. Tegyük fel, hogy az algoritmus $f+1$ menet helyett csak f menetben fut, ugyanazzal a döntési értékkel. Találjunk egy olyan végrehajtási sorozatot, mely megsérti a helyességi feltételeket.

5.3-33. Legfeljebb mennyi lehet a hibamentes folyamatok által hozott, egymástól különböző döntési értékek darabszáma, ha a HALMAZ-TERJED algoritmus $f+1$ menet helyett csak f menetben fut.

5.3-34. *a.* Találjunk egy másik lehetséges, helyesen működő döntési szabályt a HALMAZ-TERJED algoritmusban, amelyik eltér szövegben megadottól.

b. Adjunk pontos jellemzést azon döntési szabályok halmazáról, amelyek helyesen működnek.

5.3-35. Terjesszük ki a HALMAZ-TERJED algoritmust, a helyesség bizonyítását, és az elemzést, tetszőleges összefüggő gráfokra.

5.3-36. Készítsük el az OPT-HALMAZ-TERJED algoritmus kódját.

5.3-37. Tekintsük a következő egyszerű algoritmust a megállási hibák mellett történő megegyezésre, egy adott V értékhalmaz esetében. Legyen mindegyik processzornak egy *min-érték* változója, melyet induláskor a saját kezdeti értékére állít be. Az $f+1$ menet mindegyikében a processzorok közreadják *min_érték* változójuk értékét, majd újra beállítják úgy, hogy a minimuma legyen a *min_érték* változó eredeti értékének, valamint az üzenetekben kapott értékeknek. Végül a processzor döntési értéke *min_érték* lesz. Készítsük el ennek az algoritmusnak a kódját és bizonyítsuk be (vagy direkt módon, vagy szimulációval), hogy helyesen működik.

Feladatok

5-1. Vezetőválasztás négyzeten

Bizonyítsuk be, hogy négyzeten $O(n \log n)$ idő alatt megoldható a vezető választása.

5-2. Vezetőválasztás tóruszon
Bizonyítsuk be, hogy tóruszban $O(n \log n)$ idő alatt megoldható a vezetőválasztás.

5-3. Vezetőválasztás hiperkockán
Bizonyítsuk be, hogy hiperkockán $O(n \log n)$ idő alatt megoldható a vezetőválasztás.

5-4. Nem összehasonlítás alapú vezetőválasztás
Az anyagban csak összehasonlítás alapú vezetőválasztó algoritmusokat tárgyaltunk. Vizsgáljunk meg néhány más típusú algoritmust is.

a. Írjuk meg az IDŐ-SZELET algoritmus pszeudokódját.

b. Módosítsuk úgy az IDŐ-SZELET algoritmust, hogy hozzávett üzenetek árán fázisonként egyetlen azonosító helyett k darab azonosító továbbküldésének engedélyezésével csökkenjen a lépésszám. Bizonyítsuk be az algoritmus helyességét és elemezzük bonyolultságát.

c. Adjuk meg a VÁLTOZÓ-SEBESSÉGEK algoritmus pszeudokódját.

d. Mutassuk meg, hogy ha a processzorok különböző időpontokban ébredhetnek fel, a VÁLTOZÓ-SEBESSÉGEK algoritmus üzenetszáma nem szükségszerűen $O(n)$.

5-5. Harmonikus szám
Bizonyítsuk be a harmonikus számok alábbi tulajdonságait:

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n \quad (\text{ha } n \geq 1) \tag{5.11}$$

és

$$\ln(n+1) < H_n < 1 + \ln(n+1) \quad (\text{ha } n \geq 1)\ . \tag{5.12}$$

5-6. CHANG-ROBERTS algoritmus

a. Ha minden processzor kezdő processzor, legjobb esetben hány üzenetet továbbít a CHANG-ROBERTS algoritmus?

b. Ha pontosan s kezdő processzor van, amelyek egyforma valószínűséggel lesznek vezetők, akkor mennyi lesz az algoritmus átlagos *kommunikációs bonyolultsága* (üzeneteinek száma)?

5-7. Vezetőválasztás síkhálózatokban
Mutassuk meg, hogy ha egy hálózat síkba rajzolható, akkor tervezhető rá olyan vezetőválasztó algoritmus, amelynek $O(p \lg p)$ üzenetre van szüksége.

5-8. Véglegesítés
Tervezzünk egy algoritmust, amely a véglegesítési problémát az erős befejezési feltétellel megoldja. Elérhető-e egyidejűleg, hogy a menetek száma legrosszabb esetben

$n + k$ legyen (ahol k konstans), a hibamentes esetben a döntéshez és megálláshoz szükséges menetek száma egy kis konstans legyen és a hibamentes esetben alacsony legyen az üzenetszám?

5-9. Bizánci megegyezés

Tervezzünk a bizánci megegyezés megoldására *egyszerű* $f + 1$ menetes algoritmust, melyhez csak $3f + 1$ processzor szükséges, és üzenetszáma polinomiális.

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