Tartalomjegyzék

3. fejezet

Vezető folyamat kiválasztása szinkron gyűrűben

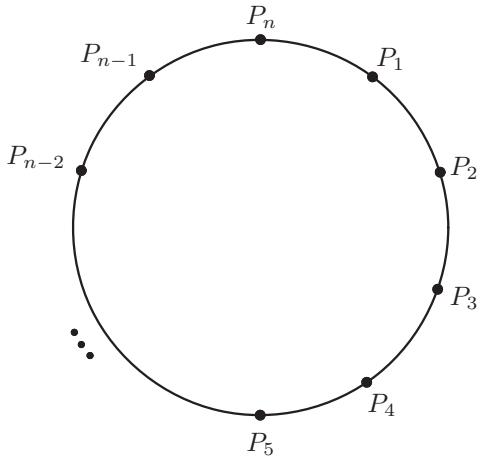
Ebben a fejezetben első megoldandó problémaként a második fejezet szinkron modelljét használó alábbi feladatot mutatjuk be: egy hálózat folyamatai közül egy *egyértelműen meghatározott vezető folyamat kiválasztását*. Először tekintsük azt az egyszerű esetet, amikor a hálózat topológiája gyűrű.

A probléma eredetileg a *vezérlőjeles gyűrű* (token ring) alapú helyi hálózatok vizsgálatainál vetődött fel. Egy ilyen hálózatban egyetlen ún. „vezérlőjel" (token) kering körbe-körbe, az éppen aktuális tulajdonosának biztosítva kizárólagos jogot kapcsolat kezdeményezésére. Ha a hálózat két csomópontja egyidőben kísérelne meg kommunikálni, az üzenetek zavarnák egymást. Néha azonban a vezérlőjel elveszhet. Ilyen esetekben a folymatoknak egy olyan algoritmust kell végrehajtaniuk, ami újra képezi az elveszett vezérlőjelet. Ezt a regeneráló eljárást hívjuk a vezető folyamat kiválasztásának.

3.1.. A feladat

Tételezzük fel, hogy a G hálózatunk egy n csúcsból álló gyűrű, az óramutató járásával egyező irányban számozva 1-től n-ig (lásd a 3.1. ábrát). Számításainkat általában modulo n végezzük, megengedve ezzel azt, hogy 0-val azonosítsuk az n-edik folyamatot, $(n+1)$-gyel az elsőt, és így tovább. A G gráf csúcsaihoz rendelt folyamatok nem ismerik a saját azonosítójukat, sem pedig szomszédaikét. Feltételezzük továbbá, hogy az üzeneteket létrehozó és továbbító függvények lokális kifejezésekkel adottak, relatív nevekkel utalva a szomszédokra. Azt is feltételezhetjük, hogy minden folyamat képes az óramutató járásával megegyező irányba eső szomszédját megkülönböztetni a másiktól. A követelmény az, hogy végül pontosan egy folyamat döntsön úgy, hogy ő a vezető folyamat, és ekkor rendeljen *vezető* értéket állapotának egy speciális *státus* összetevőjéhez. A problémának különböző változatai vannak.

1. Az is megkövetelhető lenne, hogy a vezető folyamaton kívüli összes folyamat előbb-utóbb azt a kimenetet eredményezze, hogy ő nem vezető, mondjuk



3.1.. ábra. Folyamatok egy gyűrűje.

nem_ vezető értékre változtatva *státus* összetevőjét.

2. A gyűrűt leíró gráf lehet *csak az egyik irányban irányított* vagy *mindkét irányban irányított*. Ha csak az egyik irányban irányított, akkor ez az irány az általánosság megszorítása nélkül lehet az óramutató járásával megegyező irány, vagyis üzenetek csak ebben az irányban küldhetők a szomszéd csúcs felé.

3. A gyűrűt alkotó csúcsok n számát a folyamatok vagy ismerik, vagy nem. Ha ez ismert, a folyamatoknak csak az n méretű gyűrűben kell pontosan működniük, és ekkor n értékét a folyamatok használhatják programjaikban. Ha n nem ismert, feltételezhető, hogy a folyamatok különböző méretű gyűrűkben futnak. Ekkor a gyűrű méretéről semmilyen információ nem használható fel.

4. A folyamatok lehetnek azonosak, vagy lehetnek különbözők. Utóbbi esetben kezdődhetnek egy *egyedi azonosítóval* (*UID*), amelyek egy megfelelően nagy teljesen rendezett halmazból, például a természetes számok $\mathbb{N}$ halmazából választhatók. Feltételezhetjük, hogy a gyűrűben a folyamatazonosítók különböznek egymástól, de nincs megszorítás arra, hogy milyen UID-ok szerepelhetnek ténylegesen a gyűrűben. (Például nem szükséges, hogy egymás utáni egészek legyenek.) Megadható továbbá, hogy az azonosítókat kizárólagosan mely műveletek használhatják (például összehasonlítások), vagy nem teszünk ilyen jellegű megszorításokat.

3.2.. Megoldhatatlansági eredmény azonos folyamatokra

Először is könnyű észrevenni, hogy ha az összes folyamat azonos, akkor a problémának az adott modellben nem létezik megoldása. Még akkor is ez a helyzet, ha a gyűrű mindkét irányban irányított és a folyamatok ismerik a gyűrű méretét.

3.1. tétel . *Legyen A egy n folyamatból ($n > 1$) álló rendszer, egy mindkét irányban irányított gyűrűben elrendezve. Ha A összes folyamata azonos, akkor A nem oldja meg a vezető folyamat kiválasztásának problémáját.*

Bizonyítás. Indirekt módon, tegyük fel, hogy létezik egy olyan A rendszer, ami megoldja a vezető folyamat kiválasztásának problémáját. Az általánosság megszorítása nélkül feltehető, hogy A minden egyes folyamatának pontosan egy kezdő állapota van. Ez azért igaz, mert ha minden folyamatnak több kezdő állapota is létezik, tetszőlegesen kiválaszthatunk közülük egyet, így egy olyan új megoldást kapunk, ahol minden folyamatnak már csak egy kezdő állapota van. Ezzel a feltételezéssel élve A-nak pontosan egy végrehajtási sorozata létezik.

Tekintsük tehát A-nak az egyetlen végrehajtási sorozatát. A végrehajtott menetek r száma szerinti indukcióval nem nehéz igazolni, hogy az r-edik menet után minden folyamat ugyanabban az állapotban van. Vagyis, ha valamely folyamat valaha is eléri azt az állapotot, amikor a *státus* állapota *vezető*, akkor A összes folyamata ugyanebben az időben szintén ugyanebben az állapotban lesz. Ez ellentmond az egyértelműségre vonatkozó követelménynek. □

A 3.1. tételből következik, hogy a vezető folyamat kiválasztásának problémáját csakis úgy lehet megoldani, ha valahogy megtörjük a szimmetriát. A gyakorlatban rendszerint azzal az ésszerű feltételezéssel élnek, hogy a folyamatok azonosak, kivéve az UID-et. A fejezet hátralévő részében mi is ezt a feltételt fogjuk használni.

3.3.. Egy alapvető algoritmus

Az első bemutatandó megoldás nagyon egyszerű. Le Lann, Chang és Roberts tiszteletére, akiknek a munkáiból az algoritmust vettük, LCR algoritmusnak fogjuk nevezni. Az algoritmus csak egyirányú kapcsolatot használ és nem igényli a gyűrű méretének ismeretét. Ezenkívül csak a vezető folyamatnak lesz kimenete. Az algoritmus az UID-ek között csak az összehasonlítás műveletét használja. A alábbiakban az LCR algoritmus vázlatos leírását adjuk meg.

LCR algoritmus (vázlatosan)

Minden folyamat körbeküldi az azonosítóját a gyűrűben. Amikor valamely folyamathoz egy azonosító érkezik, összehasonlítja azt a sajátjával. Ha az érkezett azonosító nagyobb, mint a sajátja, a folyamat továbbítja, ha kisebb, eldobja. Amennyiben az érkezett azonosító megegyezik a sajátjával, a folyamat önmagát nevezi meg vezetőnek.

Ebben az algoritmusban a legnagyobb UID-del rendelkező folyamat az egyetlen, amelyik a *vezető* kimenetet adja. Intuíciónkat világosabbá téve a második fejezet modelljének segítségével most algoritmusunknak egy sokkal pontosabb leírását adjuk meg.

LCR algoritmus (formálisan)
Az üzenetek M abc-je pontosan az UID-ek halmaza.
Az *állapotok*$_i$-ben lévő állapotok minden P_i folyamatra az alábbi összetevőket tartalmazzák:

> *u*, egy UID, kezdetben a P_i folyamat UID-je,
> *küld*, egy UID vagy *null*, kezdetben a P_i folyamat UID-je,
> *státus*, az {*ismeretlen, vezető*} értékek valamelyike, kezdetben *ismeretlen*.

A kezdeti állapotok *kezdő*$_i$ halmaza az imént meghatározott kezdeti értékekkel egyetlen állapotból áll.
Az *üzenetek*$_i$ üzenetgeneráló függvényt minden P_i folyamatra az alábbi módon adjuk meg:

> **send** *küld* aktuális értékét a P_{i+1} folyamatnak

Látjuk, hogy a P_i folyamatban relatív hivatkozás történik a P_{i+1} folyamatra. Például az „óramutató járásával megegyező irányba eső szomszéd" helyett egyszerűen P_{i+1}-et írunk. Emlékezzünk vissza, hogy a második fejezetben a *null* értéket használtuk az üzenet hiányának jelölésére. Vagyis ha a *küld* összetevő értéke *null*, a megfelelő *üzenetek*$_i$ függvény ténylegesen nem eredményez semmilyen üzenetet.
Az *átmenetek*$_i$ állapotátmeneti függvényt minden P_i folyamatra az alábbi pszeudokóddal adjuk meg:

```
küld := null
if az érkező üzenet v (egy UID) then
    case
        v > u: küld := v
        v = u: státus := vezető
        v < u: do semmi
    endcase
```

Az állapotátmeneti függvény definíciójának első sora törli a korábban érkezett üzenet hatását (ha volt ilyen), *null*-ra állítva a *küld* állapot-összetevőt. A kód többi része az igazán érdekes – ez tartalmazza a döntést arról, hogy a függvény továbbítsa vagy eldobja a beérkező UID-et, esetleg elfogadja azt, hogy önmagát kiálthassa ki vezető folyamatnak.

Az iménti leírás egy könnyen olvasható programozási nyelven íródott, de azért jegyezzük meg, hogy szükség esetén közvetlenül lefordítható lenne a második

fejezetben látott állapotokkal rendelkező gépmodellre. Ebben az értelmezésben az összes folyamat minden változójának van értéke, és az átmenetek a változók értékeinek megváltozását jelentik. Jegyezzük meg, hogy az *átmenetek*$_i$ függvény kódjának teljes blokkja egyetlen menetben, oszthatatlanul hajtódik végre.

Hogyan adjunk formális bizonyítást algoritmusunk helyességére? A helyesség most azt jelenti, hogy pontosan egy folyamat fogja előbb-utóbb a *vezető* kimenetet adni. Jelölje i_{max} a maximális UID-ű folyamat indexét és jelölje u_{max} ennek UID-jét. Elég lenne azt bebizonyítani, hogy (1) az n-edik menet végén az i_{max} folyamat adja a *vezető* kimenetet, és (2) egyetlen más folyamat sem ad ilyen kimenetet. A 3.2. és a 3.3. lemmákban ezen tulajdonságokat fogjuk bebizonyítani.

Itt és a könyv sok más részén is az i alsó indexet írjuk azon állapot-összetevők nevéhez, amelyeknél jelezni akarjuk, hogy a megfelelő állapot-összetevő példánya a P_i folyamathoz tartozik. Például az u_i jelölést használjuk a P_i folyamat u állapot-összetevője értékének jelölésére. A folyamat kódjának írásakor azonban általában nem használjuk az alsó indexes jelölést.

3.2. lemma . *Az n-edik menet végén az i_{max} folyamat a „vezető” kimenetet adja.*

Bizonyítás. Vegyük észre, hogy a kezdeti értékadás miatt u_{max} az $u_{i_{max}}$ változó kezdeti értéke, vagyis az i_{max} folyamat u változójáé. Azt is észrevehetjük, hogy az u változók értékei sohasem változnak meg (a kód szerint), mind különbözőek (a feltétel miatt), valamint hogy i_{max}-nak van a legnagyobb u értéke (i_{max} definíciója miatt). A kód szerint elegendő megmutatni, hogy az alábbi invariáns állítás teljesül:

3.3.1. állítás. *Az n-edik menet után státus$_i$ = „vezető”.*

Az invariáns bizonyításának legkézenfekvőbb módja az, ha a menetek száma szerinti indukciót használunk. Hogy ezt megtehessük, szükségünk lesz egy előzetes invariánsra, ami alacsonyabb menetszámok esetére mond valamit. Ezért az alábbi állítással egészítjük ki a korábbikat.

3.3.2. állítás. *Az r-edik menet után minden $0 \leq r \leq n-1$ esetén küld$_{i_{max}+r} = u_{max}$.*

(Ne felejtsük el, hogy az összeadás modulo n értendő.) Az állítás azt mondja ki, hogy a *küld* összetevőben lévő maximális érték pozíciója a gyűrűben r távolságra van i_{max}-tól.

Elég nyilvánvaló, hogy a 3.3.2. állítást r szerinti indukcióval bizonyítjuk. Az $r = 0$ esetben a kezdeti értékek miatt a nulladik menet után küld$_{i_{max}} = u_{max}$, ami pontosan az, amire szükségünk van. Az indukciós lépés azon a tényen alapul, hogy az i_{max}-tól különböző minden más csúcs felveszi a maximum értéket és ez el is tárolódik a *küld* összetevőben, mivel u_{max} nagyobb minden más értéknél.

A következő feladat a 3.3.1. állítás bizonyítása. A 3.3.2. állítás eredményét fogjuk felhasználni az $r = n-1$ speciális esetben, továbbá még egy gondolatot arról, hogy mi is történik egyetlen menet alatt. A kulcsészrevétel az, hogy az i_{max} folyamat elfogadja az u_{max} jelet, ami *vezető*-re állítja *státus* állapot-összetevőjét. □

3.3. lemma . *Az i_{max}-on kívül egyetlen más folyamat sem eredményezi a „vezető" kimenetet.*

Bizonyítás. Elég azt megmutatni, hogy minden más folyamatra a *státus = ismeretlen* teljesül. Ahogy korábban, egy erősebb invariáns segíthet. Legyen P_i és P_j a gyűrű két tetszőleges folyamata, $i \neq j$, és jelölje $[i, j)$ indexek egy $\{i, i+1, \ldots, j-1\}$ halmazát, ahol az összeadás modulo n értendő. Vagyis a gyűrűben az óramutató járásával megegyező irányba haladva $[i, j)$ a P_i-vel kezdődő folyamattól ellentétes körüljárással a P_j szomszédjáig tartó folyamatokból álló halmazt jelöli. Az alábbi invariáns azt állítja, hogy semmilyen v UID nem adódik értékül egyetlen i_{max} és v eredeti i pozíciója közötti folyamat *küld* változójának sem.

> **3.3.3. állítás.** *Az r-edik menet után, ha $i \neq i_{max}$ és $j \in [i_{max}, i)$, akkor küld$_j \neq u_i$ tetszőleges r, valamint i, j esetén.*

Ismét indukciót használunk. A bizonyítás kulcsa most az, hogy egy nemmaximális UID-ű folyamathoz sohasem érkezhet ugyanaz az UID-érték. Ez azért van így, mert i_{max} összehasonlítja a bejövő értéket u_{max}-szal és u_{max} nagyobb, mint az összes többi UID.

Végezetül a 3.3.3. állítás használható annak bizonyítására, hogy csak az i_{max} folyamat kaphatja vissza a saját UID-jét és így csak i_{max} tudja a *vezető* kimenetet előállítani. □

A 3.2. és 3.3. lemmákból az alábbi tétel következik.

3.4. tétel . *Az* LCR *algoritmus megoldja a vezető folyamat kiválasztásának problémáját.*

Megállás és *nem_vezető* kimenetek. Amint már említettük, az LCR algoritmus sohasem ér véget abban az értelemben, hogy az összes folyamat elérné a megállás állapotát. Természetesen minden folyamatot kibővíthetnénk egy megállás állapottal, ahogy azt a 2.1. alfejezetben leírtuk. Ekkor algoritmusunkat úgy módosíthatnánk, hogy a kiválasztott vezető folyamat egy speciális *értesítő* üzenetet küldjön körbe a gyűrűn. Minden folyamat, amelyik megkapta az *értesítő* üzenetet, álljon meg, miután továbbította azt. Ez a stratégia nemcsak hogy engedélyezi a folyamatoknak a megállást, de arra is használható, hogy a nemvezető folyamatok a *nem_vezető* kimenetet adják. Továbbá hozzácsatolva az *értesítő* üzenethez a vezető indexét, a stratégia az összes résztvevő folyamatnak lehetőséget ad a vezető azonosítójának közzétételére. Jegyezzük meg, hogy minden nemvezető folyamat eredményezheti a *nem_vezető* kimenetet közvetlenül azután is, amint a beérkező UID-ből látja, hogy az nagyobb, mint a sajátja. De ez nem mondaná meg a nemvezető folyamatoknak azt, hogy mikor álljanak le.

Bonyolultságelemzés. Az alap LCR algoritmus időbonyolultsága a kiválaszott elem bejelentkezéséig n menet. A legrosszabb eset kommunikációs bonyolultsága $O(n^2)$ üzenet. Az algoritmus megállásos változatában az időbonyolultság $2n$, a kommunikációs bonyolultság továbbra is $O(n^2)$. A megálláshoz és a *nem-vezető* bejelentéshez szükséges többletidő csak n menet és a többlet kommunikáció is csak n üzenet.

Transzformáció. Az előző két bekezdés egy általános transzformációt ír le és elemez, kezdve egy tetszőleges vezető folyamatot kiválasztó algoritmustól, ahol csak a vezető ad kimenetet és a többi folyamat sohasem áll le, egészen addig, amikor a vezető és nem-vezető folyamatok mindegyike ad kimenetet és minden folyamat megáll. A kimenet többletköltsége a megállással együtt is csak n menet és n üzenet volt. Ez a transzformáció egyéb feltevéseink tetszőleges kombinációjában működik.

Különböző kezdési időpontok. Jegyezzük meg, hogy az LCR algoritmus a szinkron modellben különböző kezdési időpontok esetén is módosítás nélkül működik. A modell ezen változatának leírása a 2.1. alfejezetben található.

A szimmetria megtörése. A vezető folyamat kiválasztásának gyűrűbeli problémájában az igazi nehézség a szimmetria megtörése. A szimmetria megtörése más osztott rendszerben megoldandó problémáknak is fontos részét képezik, ilyenek például az *erőforrás-hozzárendelő* és *megegyezés* problémák. (Lásd a 5.-7., 10.-12., 20.-21. és 25. fejezeteket).

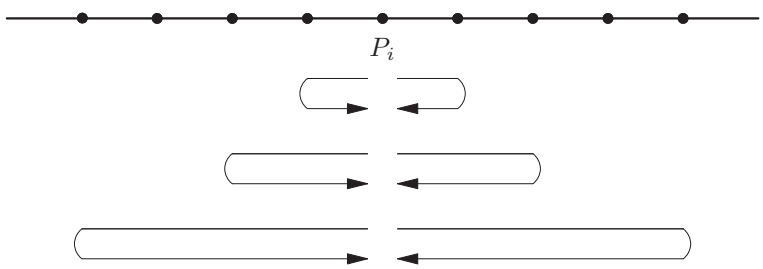
3.4.. Egy $O(n \log n)$ kommunikációs bonyolultságú algoritmus

Ámbár az LCR algoritmus időbonyolultsága alacsony, az algoritmusban előforduló üzenetek száma meglehetősen magas, $O(n^2)$. Ez egyáltalán nem tűnik jelentősnek, ha figyelembe vesszük, hogy egy csatornán egy időben soha sincs több egy üzenetnél. Azonban a második fejezetben azt is megmutattuk, hogy az üzenetek számának minimalizálása miért is olyan érdekes: sok egyidőben futó osztott algoritmus teljes kommunikációs terhelése a hálózat forgalmi torlódását eredményezheti. Ebben az alfejezetben egy olyan algoritmust mutatunk be, amely a kommunikációs bonyolultságot $O(n \log n)$-re csökkenti. Az első olyan algoritmust, amelynél a legrosszabb eset bonyolultsága $O(n \log n)$, Hirschberg és Sinclair publikálta. Tiszteletükre az algoritmust HS algoritmusnak fogjuk nevezni. Ahogy korábban, ismét feltételezzük, hogy csak a vezetőnek szükséges kimenetet eredményeznie, bár a 3.3. alfejezet végén leírt transzformációból következik, hogy ez a megszorítás nem lényeges. Ahogy korábban, most is feltételezzük, hogy a gyűrű mérete nem ismert, de most mindkét irányú kommunikációt megengedünk.

Ahogy az LCR algoritmus tette, a HS algoritmus is a maximális UID-dal rendelkező folyamatot választja ki. Most azonban az LCR algoritmustól eltérően, ahol minden folyamat az UID-jét elejétől végig körbeküldi a gyűrűn, a HS algoritmusban minden egyes folyamat UID-je bizonyos távolság megtétele után megfordul és visszatér az eredeti folyamathoz. Ezután ez ismétlődik egyre nagyobb távolságokkal. A HS algoritmus az alábbi módon jár el.

HS algoritmus (vázlatosan)

Minden P_i folyamat a $0, 1, 2, \ldots$ szakaszokban (fázisokban) működik. Minden l szakaszban a P_i folyamat az u_i UID-jét tartalmazó jeleket küld mindkét szomszédja felé. Ezek az üzenetek 2^l távolságot hivatottak megtenni,



3.2.. ábra. A HS algoritmus P_i folyamatából kiinduló egymás utáni üzenetek pályája.

miután visszatérnek a kiindulási i indexű folyamathoz (lásd a 3.2. ábrát). Ha mindkét jelsorozat biztonságban visszaérkezik, a P_i folyamat a következő szakasszal folytatódik. Azonban nem biztos, hogy a jelek biztonságban vissza is érnek. Amíg az u_i jelsorozat P_i-től távolodik, minden, az u_i útjába eső P_j folyamat összehasonlítja u_i-t a saját UID-jével, u_j-vel. Ha $u_i < u_j$, akkor P_j egyszerűen eldobja az üzenetet, míg ha $u_i > u_j$, akkor P_j továbbítja u_i-t. Ha pedig $u_i = u_j$, akkor ez azt jelenti, hogy a P_j folyamathoz még visszafordulás előtt a saját UID-je érkezett, vagyis a P_j folyamat önmagát választja vezetőnek.

Ha a P_i folyamat által küldött üzenet közeledik, azt minden folyamat mindig továbbítja.

Most az algoritmust formálisan is megadjuk. A formalizálás némi könyvelést is igényel, hiszen biztosítani kell, hogy a jelsorozatok a megfelelő irányokba tartsanak. Például a jelsorozatoknak tartalmazniuk kell azt az információt, hogy távolodnak vagy közelednek a kiindulási helyüktől. Hasonlóképpen, a jelsorozatoknak tartalmazniuk kell a közben érintett folyamatok számát is, hogy távolodáskor a távolságot számon lehessen tartani. Ez lehetőséget ad a folyamatoknak arra, hogy kitalálják, mikor kell az üzenetet visszafordítani. Az algoritmus formalizálása után az LCR algoritmusnál látott módon indokoljuk annak helyességét.

HS algoritmus (formálisan)

Az üzenetek M abc-je olyan hármasokból álló halmaz, ahol az elemek összetevői az UID-ek, az üzenetek mozgási irányát jelző $\{$*megy, jön*$\}$ értékek valamelyike és egy pozitív egész szám (*ugrás_ számláló*), ami a közben érintett folyamatok számát jelöli.

Az *állapotok*$_i$-ben lévő állapotok minden P_i folyamatra az alábbi összetevőket tartalmazzák:

u, egy UID, kezdetben a P_i folyamat UID-je,
küld+, ami vagy az M egy elemét tartalmazza, vagy *null*,
kezdetben a hármas tartalma a P_i folyamat UID-je, *megy* és 1,
küld−, ami vagy az M egy elemét tartalmazza, vagy *null*,
kezdetben a hármas tartalma a P_i folyamat UID-ja, *megy* és 1,
státus, az {*ismeretlen, vezető*} értékek valamelyike, kezdetben *ismeretlen*,
fázis, nemnegatív egész, kezdetben 0.

A kezdeti állapotok *kezdő*$_i$ halmaza az imént definiált kezdeti értékekkel egyetlen állapotból áll.
Az *üzenetek*$_i$ üzenetgeneráló függvényt minden P_i folyamatra az alábbi módon adjuk meg:

a *küld+* aktuális értékét továbbítsd a P_{i+1} folyamatnak,
a *küld−* aktuális értékét továbbítsd a P_{i-1} folyamatnak.

A *átmenetek*$_i$ állapotátmeneti függvény minden P_i folyamatra az alábbi pszeudokóddal adott:

küld+ := *null*
küld− := *null*
if a P_{i-1}-ből érkező üzenet (*v, megy, h*) **then**
  case
    $v > u$ és $h > 1$: *küld+* := (*v, megy,* $h-1$)
    $v > u$ és $h = 1$: *küld−* := (*v, jön,* 1)
    $v = u$: *státus* := *vezető*
  endcase
if a P_{i+1}-ből érkező üzenet (*v, megy, h*) **then**
  case
    $v > u$ és $h > 1$: *küld−* := (*v, megy,* $h-1$)
    $v > u$ és $h = 1$: *küld+* := (*v, jön,* 1)
    $v = u$: *státus* := *vezető*
  endcase
if a P_{i-1}-ből érkező üzenet (*v, jön,* 1) és $v \neq u$ **then**
  küld+ := (*v, jön,* 1)
if a P_{i+1}-ből érkező üzenet (*v, jön,* 1) és $v \neq u$ **then**
  küld− := (*v, jön,* 1)
if a P_{i-1}-ből és a P_{i+1}-ből érkező üzenetek mindegyike (*u, jön,* 1) **then**
  fázis := *fázis*+1
  küld+ := (*u, megy,* $2^{fázis}$)
  küld− := (*u, megy,* $2^{fázis}$)

Ahogy a korábbi esetben, az első két sor inicializálja a folyamatot. A kód következő két részlete a távolodó üzenetek kezelését írja le: azok a jelek, amelyek UID-része nagyobb, mint u_i, vagy továbbítódnak, vagy visszafordítódnak, a közben érintett folyamatok számától függően. Amennyiben éppen az u_i érkezett, a P_i folyamat önmagát választja vezetőnek. A kód következő két részlete a közeledő üzenetek kezelésére ad választ: ezek egyszerűen továbbítódnak. (A közeledő üzenetek esetén az *ugrás_számláló* értéke 1.) Ha a P_i folyamat mindkét saját üzenetét visszakapta, a folyamat a következő szakaszba lép.

Bonyolultságelemzés. Először a kommunikációs bonyolultságot elemezzük. A 0 szakaszban minden folyamat elküld egy üzenetet. Ez összesen $4n$, mindkét szomszéd felé egy elküldött, egy érkezett üzenet. Minden $l > 0$ esetén az l szakaszban egy folyamat pontosan akkor küld jeleket, ha az $l-1$ szakaszban mindkét üzenet visszaérkezett. Márpedig ez a helyzet akkor, ha semmilyen más, a gyűrű bármely irányában 2^{l-1} távolságon belüli folyamat nem hiúsítja meg a továbbhaladást. Ebből az következik, hogy bármely $2^{l-1}+1$ darab egymás melletti folyamat közül legfeljebb egy olyan lesz, amelyik az l szakaszban üzenetet küld. Ezen gondolat segítségével megmutatható, hogy összesen legfeljebb

$$\left\lfloor \frac{n}{2^{l-1}+1} \right\rfloor$$

folyamat küld üzenetet az l szakaszban. Az l szakaszban elküldött összes üzenetek számára így az alábbi felső becslést kapjuk:

$$4\left(2^l \cdot \left\lfloor \frac{n}{2^{l-1}+1} \right\rfloor\right) \leq 8n.$$

Azért ennyi, mert az l szakaszban a jelsorozatok 2^l távolságot tesznek meg. Mint korábban, a 4-es szorzó abból adódik, hogy az üzenetek kétirányban távolodnak és minden távolodó üzenet valamikor megfordul és visszaérkezik.

A vezető folyamat kiválasztása és az összes kommunikáció befejezése előtt végrehajtott szakaszok száma (a 0 szakasszal együtt) legfeljebb $1 + \lceil \log n \rceil$, így az üzenetek teljes száma legfeljebb $8n(1 + \lceil \log n \rceil)$, ami $O(n \log n)$, a konstans szorzó pedig közelítőleg 8.

Az algoritmus időbonyolultsága $O(n)$. Ezt könnyű belátni, hiszen minden l szakasz $2 \cdot 2^l = 2^{l+1}$ ideig tart (az üzenetek először távolodnak, aztán közelednek). Az utolsó szakasz n ideig tart – ez nem egy teljes szakasz, mert az üzenetek csak távolodnak. Az utolső előtti szakasz az $l = \lceil \log n \rceil - 1$ szakasz, ennek időigénye legalább annyi, mint az összes ezt megelőző szakaszoké összesen. Vagyis az utolsó szakasz kivételével a teljes időigény legfeljebb

$$2 \cdot 2^{\lceil \log n \rceil}.$$

Összegezve, ha n kettőhatvány, akkor a teljes időigény legfeljebb $3n$, egyébként legfeljebb $5n$. A hiányzó részletek belátását az Olvasóra bízzuk.

Különböző kezdési időpontok. A HS algoritmus a szinkron modellben különböző kezdési időpontok esetén is módosítás nélkül működik.

3.5.. Nem összehasonlításon alapuló algoritmusok

A következőkben azt a kérdést feszegetjük, hogy lehetséges-e a vezető folyamat kiválasztása kevesebb, mint $\Omega(n \log n)$ üzenettel. A válasz erre a kérdésre nemleges, a megoldhatatlansági eredményt meg is mutatjuk. Ez az eredmény azonban csak azokra az algoritmusokra igaz, amelyek kizárólag az UID-ek összehasonlításán alapulnak. (Az *összehasonlításon alapuló algoritmusokat* a 3.6. alfejezetben definiáljuk.)

Ebben a fejezetben feltételezzük, hogy az UID-ek pozitív egészek, amelyeken általános aritmetikai műveleteket hajthatunk végre. Két algoritmust is adunk erre az esetre, az egyik az IdőSzelet algoritmus, a másik a VáltozóSebességek algoritmus, mindkettő $O(n)$ kommunikációs bonyolultsággal. Ezen algoritmusok létezéséből az is következik, hogy az általános esetben $\Omega(n \log n)$-nél jobb alsó korlát nem adható meg.

3.5.1.. Az IdőSzelet algoritmus

A most bemutatandó algoritmus azt a feltevést használja, hogy a gyűrű n méretét minden folyamat ismeri és az üzenettovábbítás egyirányú. Ezekkel a feltevésekkel az alábbi egyszerű algoritmus – amit IdőSzelet algoritmusnak fogunk nevezni – megfelelően működik. Az algoritmus a minimális UID-ű folyamatot választja ki.

Jegyezzük meg, hogy ez az algoritmus a szinkronitást mélyebb értelemben használja, mint az LCR algoritmus vagy a HS algoritmus. Az IdőSzelet algoritmus bizonyos menetekben az üzenetek nemlétét is felhasználja (egészen pontosan az érkező *null-üzenet*eket), mint információt.

IdőSzelet algoritmus (vázlatosan)

A számítások az $1, 2, \ldots$ szakaszokban (fázisokban) hajtódnak végre, ahol minden szakasz n egymás utáni menetből áll. Minden szakasz olyan lehetséges forgalmat jelent, amihez bizonyos UID-ű üzenetek tartoznak és ezek körbemennek a gyűrűn. Egészen pontosan a v szakaszban, ami a $(v-1)n+1, \ldots, vn$ menetekből áll, csak a v UID-ű üzenetek forgalma engedélyezett.

Ha valamely időpillanatban a P_i folyamat a v UID-del rendelkezik és a $(v-1)n+1$ menet anélkül ért véget, hogy a P_i folyamat előzőekben egyetlen *nem-null* üzenetet kapott volna, akkor a P_i folyamat önmagát választja vezetőnek és a saját UID-jét körbeküldi a gyűrűn. Amikor ez az üzenet megy körbe a gyűrűn, az összes érintett folyamat feljegyzi annak megérkeztét. Ez a feljegyzés a későbbiekben megakadályozza őket abban, hogy önmagukat választhassák vezető folyamatnak vagy bármely későbbi fázisban üzenetküldést kezdeményezzenek.

Az algoritmusban a minimális UID (u_{min}) előbb-utóbb körbemegy a gyűrűn, aminek következtében annak eredeti tulajdonosa önmagát választja kitüntetett folyamatnak. Az $(u_{min}-1)n+1$ menet előtt nincs üzenetküldés, ahogy az $u_{min} \cdot n$ menet után sincs. Az elküldött üzenetek száma így n. Ha a minimális UID-ű folyamat helyett mi a maximális UID-ű folyamatot szeretnénk meghatározni, akkor a minimális UID-ű folyamat megkeresése után az egyszerűen körbeküld egy speciális üzenetet, aminek a segítségével u_{max} könnyen meghatározható. A kommunikációs bonyolultság így is $O(n)$.

Az IdőSzelet algoritmus jó tulajdonsága, hogy az üzenetek száma n. Sajnos az időbonyolultság körülbelül $n \cdot u_{min}$, ami még rögzített méretű gyűrűben is tetszőlegesen nagy lehet. Ezért az időbonyolultság erősen korlátozza az algoritmus gyakorlati használhatóságát: csak kis elemszámú gyűrű hálózatban és kis pozitív egész UID-ű folyamatok esetén praktikus.

3.5.2.. A VáltozóSebességek algoritmus

Az IdőSzelet algoritmus szerint abban az esetben, ha a folyamatok ismerik a gyűrű n méretét, $O(n)$ üzenet elegendő a vezető folyamat kiválasztásához. De mi a helyzet akkor, ha n ismeretlen? Megmutatjuk, hogy ebben az esetben is létezik $O(n)$ kommunikációs bonyolultságú algoritmus. Az általunk bemutatott algoritmust VáltozóSebességek algoritmusnak fogjuk nevezni. Az elnevezés rövid időn belül világos lesz. A VáltozóSebességek algoritmus is csak egyirányú kommunikációt használ.

Sajnos a VáltozóSebességek algoritmus időbonyolultsága még az IdőSzelet algoritmusénál is rosszabb, $O(n \cdot 2^{u_{min}})$. Világos, hogy senkinek sem jutna eszébe az algoritmust a gyakorlatban alkalmazni. A VáltozóSebességek algoritmus az, amit mi *ellenpélda algoritmusnak* hívunk. Az *ellenpélda algoritmus* egy olyan algoritmus, amelynek a fő célja annak megmutatása, hogy a megsejtett megoldhatatlansági eredmény hamis. Az ilyen algoritmus önmagában sem gyakorlati, sem matematikai szempontból nem érdekes és nem is különösebben elegáns. Azonban annak megmutatására kiválóan alkalmas, hogy a megoldhatatlansági eredményt nem lehet bebizonyítani. Íme az algoritmus.

VáltozóSebességek algoritmus (vázlatosan)

Minden P_i folyamat egy olyan jelsorozatot küld körbe a gyűrűn, ami a kiindulási folyamat u_i UID-jét tartalmazza. A különböző üzenetek különböző sebességgel haladnak. Egészen pontosan a v UID-ot tartalmazó üzenet minden 2^v menetben egyetlen lépést halad előre. Vagyis minden folyamat, miután üzenetet kapott, vár 2^v menetet, és csak azután küld valamit tovább.

Időközben minden folyamat figyelemmel követi az általa addig látott legkisebb UID-ot és egyszerűen nem küldi tovább azokat az üzeneteket, amelyekben az UID nagyobb ennél az értéknél.

Amikor egy üzenet visszaérkezik a feladójához, a folyamat önmagát választja vezetőnek.

Ahogy az IdőSzelet algoritmus, a VáltozóSebességek algoritmus is garantálja a minimális UID-ű folyamat kiválasztását.

Bonyolultságelemzés. A VáltozóSebességek algoritmus garantálja, hogy a legkisebb UID-et tartalmazó üzenet (u_{min}) teljesen körbemegy a gyűrűn, a második legkisebb azonosítót tartalmazó üzenet legfeljebb a teljes út felét, a harmadik legkisebb UID legfeljebb a teljes út negyedét, általában pedig a k-adik legkisebb azonosítót tartalmazó üzenet legfeljebb a teljes út $1/(2^{k-1})$ részét teheti meg. Ezért a vezető kiválasztásának pillanatáig az u_{min} azonosítót tartalmazó üzenetek száma az összes többi üzenet együttes számánál is nagyobb. Mivel az u_{min} körbeérése pontosan n üzenetet jelent, a vezető folyamat kiválasztásáig az összes üzenetek száma kevesebb, mint $2n$.

Vegyük észre, hogy amialatt az u_{min} azonosítót tartalmazó üzenet körbemegy a gyűrűn, az összes érintett folyamat tudomást szerez erről az értékről, vagyis ők a későbbiekben már semmilyen üzenetet nem fognak küldeni. Ebből az következik, hogy $2n$ a *valaha elküldött üzenetek* számának egy felső becslése (magába foglalva a *vezető* kimenet megjelenése utáni időt is).

Ahogy korábban említettük, az algoritmus időbonyolultsága $n \cdot 2^{u_{min}}$, mert minden folyamat $2^{u_{min}}$ időegységgel késlelteti az u_{min} UID-et tartalmazó üzenet továbbküldését.

Különböző kezdési időpontok. Az LCR algoritmustól és a HS algoritmustól eltérően az IdőSzelet algoritmus nem használható minden további nélkül a szinkron modell válzozatban különböző kezdési időpontok mellett. Az algoritmus egy módosított változata azonban már igen.

> **MódVáltozóSebességek algoritmus (vázlatosan)**
>
> Egy folyamatot nevezzünk *induló* folyamatnak, ha pontosan egy menettel korábban kapott egy *ébresztő* üzenetet, vagyis bármilyen közönséges (nemnull) üzenetet.
>
> Minden P_i *induló* folyamat a saját u_i UID-jét tartalmazó üzenetet küld körbe a gyűrűn. A nem *induló* folyamatok sohasem küldenek jelet. Kezdetben ez a jel „gyorsan" halad, egy átvitel/menet sebességgel. Az üzenetet fogadó „nem-induló" folyamat „felébred", ha pedig már *induló* folyamat volt (lehetett egy másik *induló* folyamat, vagy P_i önmaga), akkor az üzenet lelassulva folytatja útját, minden 2^{u_i} menetben egy átvitel sebességgel.
>
> Időközben minden folyamat feljegyzi az *induló* folyamatok közül a minimális UID-űt és eldob minden olyan üzenetet, amiben az UID nagyobb, mint ez a legkisebb érték. Amikor egy üzenet visszaérkezik a feladójához, a folyamat önmagát választja vezetőnek.

A MódVáltozóSebességek algoritmus biztosítja a minimális UID értékkel rendelkező *induló* folyamat vezetővé választását. Jelölje ezen folyamat indexét i_{minind}.

Bonyolultságelemzés. Az üzeneteket három csoportba oszthatjuk.

1. A kezdeti gyorsan továbbításra kerülő üzenetek. Ezekből pontosan n darab van.
2. A lassan továbbítandó üzenetek addig a pillanatig, amíg az i_{minind} üzenet először ér el valamely *induló* folyamatot. Ez az első felébredő folyamat idejétől kezdve legfeljebb n menet. Ezen idő alatt a v UID-del rendelkező üzenetek legfeljebb $n/2^v$-szer fordulhatnak elő, az üzenetek teljes összege így legfeljebb $\sum_{v=1}^{n} n/2^v < n$.
3. A lassan továbbítandó üzenetek attól a pillanattól kezdve, amikor az i_{minind} üzenet először ér el valamely *induló* folyamatot. Innentől az elemzésünk hasonló a VÁLTOZÓSEBESSÉGEK algoritmusnál látottakhoz. Amíg a győztes jelsorozat végighalad a gyűrűn, a k-adik legkisebb *induló* folyamat azonosítóját tartalmazó üzenet legfeljebb a teljes út $1/(2^{k-1})$ részét teheti meg. Ezért az üzenetek számának teljes összege a vezető folyamat kiválasztásának pillanatáig kevesebb, mint $2n$. De amíg a győztes jelsorozat végighalad a gyűrűn, minden folyamat megismervén annak minimális UID értékét, a továbbiakban semmilyen üzenetet nem küld. Vagyis ebben az esetben az üzenetek számának egy felső korlátja $2n$.

Azt kaptuk, hogy a teljes kommunikációs bonyolultság legfeljebb $4n$.

Az időbonyolultság $n + n \cdot 2^{u_{minind}}$.

3.6.. Alsó korlát az összehasonlításos algoritmusokra

Az eddigiekben különféle algoritmusokat láttunk vezető folyamat kiválasztására szinkron gyűrűben. Az LCR algoritmus és a HS algoritmus összehasonlításon alapuló algoritmusok voltak, az utóbbi $O(n \log n)$ kommunikációs bonyolultsággal és $O(n)$ időbonyolultsággal. Ezzel szemben az IDŐSZELET algoritmus és a VÁLTOZÓSEBESSÉGEK algoritmusok nem összehasonlításos algoritmusok voltak $O(n)$ kommunikációs és nagyon nagy időbonyolultsággal. Ebben az alfejezetben az üzenetek számának egy $\Omega(n \log n)$ alsó korlátját bizonyítjuk az összehasonlításos algoritmusokra. Ez az alsó korlát akkor is érvényben marad, ha a gyűrűben kétirányú kommunikációt engedélyezünk, és ha a folyamatok ismerik a gyűrű n méretét. A következő alfejezetben hasonló alsó korlátot mutatunk a nem összehasonlításon alapuló algoritmusokra abban az esetben, ha az időbonyolultság korlátos.

Ezen alfejezet eredménye a *szimmetria megtörésének* nehézségén alapszik. Emlékezzünk vissza 3.1. tétel megoldhatatlansági eredményére: a szimmetria, a megkülönböztetett információ (mint például az UID-ek) hiánya miatt nem lehetséges a vezető folyamat kiválasztása. Az alábbi érvelés legfontosabb eleme az, hogy bizonyos mennyiségű szimmetria még az (egyedi) UID-ek jelenléte mellett is előfordulhat. Ebben az esetben az UID-ek lehetővé teszik a szimmetria megtörését, de ez hatalmas mennyiségű kommunikációt vehet igénybe.

Idézzük fel az egész fejezetre vonatkozólag tett korábbi megjegyzésünket, miszerint a gyűrű folyamatai azonosak, kivéve az UID-eket. Vagyis a folyamatok kezdőállapotai azonosak, kivéve azon összetevőket, amelyek az UID-eket tartal-

mazzák. Általában nem tettünk semmilyen megszorítást arra nézve, hogy az üzenetgeneráló és állapotátmeneti függvények az UID-eket hogyan használhatják.

A fejezet hátralévő részében (ezen alfejezet és a következő) feltételezzük, hogy csak egyetlen kezdő állapot létezik, ami minden egyes UID-et tartalmaz. (Ahogy a 3.1. tétel bizonyításánál, a feltevés most sem csorbítja az általánosságot.) A feltételezés előnye, hogy következményeként (az UID-ek rögzített értékei mellett) a rendszernek pontosan egy végrehajtása létezik.

Az összehasonlítás alapú algoritmusok megengednek bizonyos megszorításokat, amelyek az alábbi, kissé vázlatos definíció segítségével is kifejezhetők. Egy UID alapú gyűrű algoritmus *összehasonlítás alapú*, ha az UID-ek kezelésének kizárólagos módjai a másolás, az üzenetekben való részvétel (küldés, fogadás) és az összehasonlítás $(<, >, =)$ lehetnek.

A definíció megengedi a folyamatoknak az algoritmus végrehajtása során korábban előfordult UID-ek bármelyikének tárolását. Megengedi továbbá, hogy esetleg más információval együtt a folyamatok továbbküldjék őket. A folyamatok összehasonlíthatják a tárolt UID-eket és az összehasonlítások eredményeit felhasználhatják az üzenetgeneráló és állapotátmeneti függvények választásainál. Ezek a választások magukban foglalhatnak olyanokat is, mint például küldjön vagy ne küldjön üzenetet a szomszédainak, válassza vagy ne önmagát vezetőnek, tárolja vagy ne tárolja tovább az UID-ek valamelyikét, stb. A lényeg az, hogy a folyamatok bármely tevékenysége csak az előforduló UID-ek relatív rangsorától függ, nem pedig a pontos értékeitől.

Az alábbi formális fogalom az esetleg fennálló, UID-eket is tartalmazó szimmetriát írja le. Legyen $U = (u_1, u_2, \ldots, u_k)$ és $V = (v_1, v_2, \ldots, v_k)$ UID-ek két k hosszúságú sorozata. Azt mondjuk, hogy az U *sorrendekvivalens* a V-vel, ha minden i, j esetén $(1 \leq i, j \leq k)$ $u_i \leq u_j$ akkor és csak akkor teljesül, ha $v_i \leq v_j$.

3.6.1. példa. Sorrendekvivalencia.

Az $(5, 3, 7, 0), (4, 2, 6, 1)$ és az $(5, 3, 6, 1)$ sorozatok sorrendekvivalensek, amennyiben az UID-ek halmaza a nemnegatív egész számok halmaza a szokásos rendezéssel.

Vegyük észre, hogy UID-ek két sorozata akkor és csak akkor sorrendekvivalens, ha a sorozatokban az UID-ek relatív sorrendje megegyezik. Most még két definíció következik. Azt mondjuk, hogy egy kiértékelés egy menete *aktív*, ha benne legalább egy (nem-null) üzenet elküldésre kerül. Az n méretű R gyűrű valamely P_i folyamatának *k-szomszédsága* alatt a $P_{i-k}, \ldots, P_{i+k}$ ($2k + 1$ darab) folyamatokat értjük $(0 \leq k < \lfloor n/2 \rfloor)$. Ezek pontosan azok a folyamatok, amelyek a P_i folyamattól legfeljebb k távolságra vannak és magát a P_i folyamatot is beleértjük.

Végezetül szükségünk lesz egy olyan definícióra, ami a folyamatok állapotainak azonosságát írja le (megengedve a folyamatok UID-jeinek különbözőségét). Azt mondjuk, hogy az s és t folyamatállapotok az $U = (u_1, u_2, \ldots, u_k)$ és $V = (v_1, v_2, \ldots, v_k)$ sorozatora vonatkozóan *összhangban vannak*, ha az alábbiak teljesülnek: s minden UID-je az U sorozatból való, t minden UID-je a V sorozatból való, valamint s és t megegyeznek, kivéve, hogy az s-beli u_i minden

előfordulása helyett a t-beli v_i kerül minden $1 \leq i \leq k$-ra. Az *összhangban lévő üzenetek* analóg módon definiálhatók.

Most már készen állunk az alsó korlát bizonyítására. Az alábbi lemma azt állítja, hogy a sorrendekvivalens k-szomszédságban lévő folyamatok lényegében ugyanúgy viselkednek, és ezt a viselkedésbeli azonosságot csak valamilyen, a k-szomszédságon kívülről érkező üzenet képes megtörni.

3.5. lemma . *Legyen A egy n méretű R gyűrűben működő összehasonlításon alapuló algoritmus és legyen k egy egész szám, $0 \leq k < \lfloor n/2 \rfloor$. Legyen továbbá P_i és P_j két olyan folyamat A-ban, amelyekre a k-szomszédságaik UID-sorozatai sorrendekvivalensek. Ekkor legfeljebb k aktív menet utáni tetszőleges pillanatban a P_i és P_j folyamatok állapotai k-szomszédságaik UID-sorozataira vonatkozóan összhangban vannak.*

3.6.2. példa. Összhangban lévő állapotok

Legyen a P_i folyamat 3-szomszédságú UID-sorozata $(1, 6, 3, 8, 4, 10, 7)$ (ahol a P_i folyamat UID-ja 8), a P_j folyamat 3-szomszédságú UID-sorozata pedig $(4, 10, 7, 12, 9, 13, 11)$ (ahol a P_j folyamat UID-je 12). Mivel a két sorozat sorrendekvivalens, az iménti lemmából következik, hogy a P_i és P_j folyamatok állapotai legfeljebb három aktív meneten keresztül 3-szomszédságaik UID sorozataira vonatkozóan biztosan összhangban maradnak. Durván fogalmazva az indok az, hogy amennyiben csak három aktív menet történik, a sorrendekvivalens 3-szomszédságon kívüli üzeneteknek nincs lehetőségük P_i és P_j elérésére.

Bizonyítás. (3.5. lemma). Az általánosság megszorítása nélkül feltehető, hogy $i \neq j$. A bizonyítás a menetek r száma szerinti indukcióval történik. Minden egyes r esetére a lemmát az összes lehetséges k-ra bebizonyítjuk.

Induló eset. $r = 0$. Az összehasonlítás-alapú algoritmusok definíciójának megfelelően P_i és P_j kezdő állapotai az UID-jeiktől eltekintve megegyeznek, ezért kezdő állapotaik k-szomszédságaikra vonatkozóan minden k-ra összhangban vannak.

Indukciós feltevés. Tegyük fel, hogy a lemma állítása minden $r' < r$ menetre teljesül. Rögzítsük k-t oly módon, hogy P_i és P_j k-szomszédságai sorrendekvivalensek legyenek és tegyük fel, hogy az első r menet legfeljebb k aktív menetet tartalmaz.

Ha az r menetben sem P_i, sem P_j nem kap üzenetet, akkor az indukciós feltevés miatt P_i és P_j állapotai k-szomszédságaikra vonatkozóan összhangban lesznek, hiszen az $r-1$ menetben is abban voltak, és új üzenet híján az állapotátmenetek összhangban maradnak.

Vagyis feltehető, hogy az r menetben vagy P_i vagy P_j új üzenetet kap. Ekkor az r menet aktív, tehát az első $r-1$ menet legfeljebb $k-1$ aktív menetet tartalmazott. Figyeljük meg, hogy P_i és P_j állapotainak $(k-1)$-szomszédságai sorrendekvivalensek, ami P_{i-1} és P_{j-1} valamint P_{i+1} és P_{j+1} állapotaira is igaz.

Ezért az $r-1$ menet után az indukciós feltevés miatt ($r-1$-re és $k-1$-re) P_i és P_j állapotai k-szomszédságaikra vonatkozóan szintén összhangban lesznek. Hasonló állítás igazolható P_{i-1} és P_{j-1} valamint P_{i+1} és P_{j+1} állapotaira.

A bizonyítást az alábbi esetek vizsgálataival folytatjuk.

1. Az r menetben sem P_{i-1}, sem P_{i+1} nem küld P_i-nek üzenetet.
   Ekkor, mivel P_{i-1} és P_{j-1} állapotai az $r-1$ menet után összhangban vannak és ugyanez igaz P_{i+1} és P_{j+1} állapotaira, azt kapjuk, hogy az r menetben sem P_{j-1} sem P_{j+1} nem küld P_j-nek üzenetet. De ez ellentmond azon feltételezésünknek, hogy vagy P_i vagy P_j az r menetben új üzenetet kap.
2. Az r menetben P_{i-1} üzenetet küld P_i-nek, de P_{i+1} nem.
   Ekkor, mivel P_{i-1} és P_{j-1} állapotai az $r-1$ menet után összhangban voltak, az r menetben P_{j-1} szintén küld üzenetet P_j-nek, és ez az üzenet P_{i-1} és P_{j-1} állapotai $(k-1)$-szomszédságaira vonatkozóan, ezért P_i és P_j állapotai k-szomszédságaira vonatkozóan is összhangban lesz azzal az üzenettel, amit P_{i-1} küld P_i-nek. Hasonló indoklással az r menetben P_{j+1} sem küld üzenetet P_j-nek. Mivel P_i és P_j állapotai az $r-1$ menet után összhangban voltak és összhangban lévő üzeneteket kaptak, ezért állapotaik k-szomszédságaikra vonatkozóan is összhangban maradnak.
3. Az r menetben P_{i+1} üzenetet küld P_i-nek, de P_{i-1} nem.
   Ez az eset az előzőhöz hasonló módon látható be.
4. Az r menetben P_{i-1} is és P_{i+1} is küld üzenetet P_i-nek.
   Az indoklás itt is a korábbihoz hasonló.

□

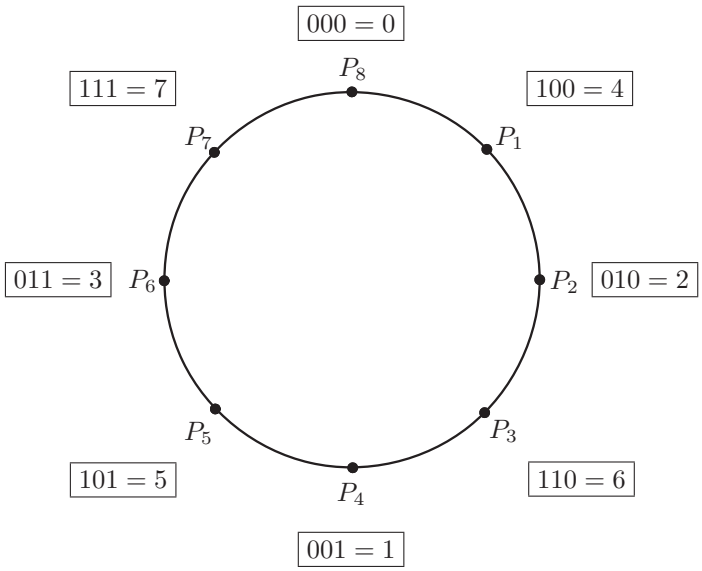
A 3.5. lemma azt mutatja, hogy elég sok aktív menet szükséges a szimmetria megtöréséhez, ha nagy sorrendekvivalens szomszédságok léteznek. Most egy olyan különleges tulajdonságokkal rendelkező gyűrűt definiálunk, amiben sok, különböző méretű sorrendekvivalens szomszédság található. Legyen c egy adott konstans ($0 \leq c \leq 1$) és legyen az R gyűrű mérete n. Azt mondjuk, hogy az R gyűrű *c-szimmetrikus*, ha minden l-re ($\sqrt{n} \leq l \leq n$) és R minden l hosszú S szegmenséhez létezik legalább $\lfloor cn/l \rfloor$ olyan szegmens R-ben, ami sorrendekvivalens S-sel (önmagát S-et is beleszámolva). Megjegyezzük, hogy az alsó korlátnál a négyzetgyökös feltétel elhagyható.

Ha n kettőhatvány, akkor $\frac{1}{2}$-szimmetrikus gyűrűt könnyű konstruálni. A következőkben az n méretű *bitfordító gyűrűt* definiáljuk. Legyen $n = 2^k$. Ekkor minden P_i folyamathoz azt a $[0, n-1]$ intervallumból való egészet rendeljük hozzá, amelynek a k-bites bináris ábrázolása a fordítottja az i indexű folyamatban az i érték k-bites bináris ábrázolásának (ahol 0^k jelöli a 0 folyamat k-bites bináris ábrázolását).

3.6.3. példa. Bitfordító gyűrű.

$n = 8$ esetén $k = 3$ és a megfeleltetések a 3.3. ábrán láthatók.

3.6. lemma . *Minden bitfordító gyűrű $\frac{1}{2}$-szimmetrikus.*



3.3.. ábra. Egy 8 méretű bitfordító gyűrű.

Bizonyítás. Azzal a megjegyzéssel, hogy a bitfordító gyűrűk esetén nincs szükség a négyzetgyököt tartalmazó alsó korlátra, a bizonyítást az olvasóra bízzuk. □

A nem kettőhatvány n értékekre is léteznek c-szimmetrikus gyűrűk, de az általános eset kisebb c konstansot igényel.

3.7. tétel . *Létezik olyan c konstans, hogy minden $n \in \mathbb{N}$ esetben megadható egy n méretű c-szimmetrikus gyűrű.*

A 3.7. tétel bizonyítása egy meglehetősen bonyolult rekurzív konstrukción alapul (itt használnánk ki maradéktalanul a négyzetgyökös alsó korlátot). A keresett gyűrű előállítása nem könnyű feladat. Nem mondhatjuk egyszerűen azt, hogy az n-nél kisebb legnagyobb kettőhatvány méretű bitfordító gyűrűhöz hozzáveszünk néhány folyamatot. A hozzávett folyamatok ugyanis elrontják a szimmetriát.

Vagyis feltételezzük, hogy minden n-re létezik egy n méretű c-szimmetrikus R gyűrű. A következő lemma azt állítja, hogy ha egy ilyen gyűrű kiválasztja a vezető folyamatot, akkor az eljárásban sok aktív menet hajtódik végre.

3.8. lemma . *Legyen A egy öszehasonlításon alapú algoritmus valamely n méretű c-szimmetrikus gyűrűben és tegyük fel, hogy A kiválasztja a vezető folyamatot. Tegyük fel továbbá, hogy k egy olyan egész, amelyre $\sqrt{n} \leq 2k+1$ és $\lfloor cn/(2k+1) \rfloor \geq 2$. Ekkor A több mint k aktív menetet hajt végre.*

Bizonyítás. Indirekt módon okoskodunk. Tegyük fel, hogy A legfeljebb k aktív menetben választja ki a vezető P_i folyamatot. Legyen S a P_i folyamat k-szomszédsága. Ekkor S egy $2k+1$ hosszúságú szegmens. Mivel a gyűrű c-szimetrikus, a gyűrűben léteznie kell legalább $\lfloor cn/(2k+1) \rfloor \geq 2$ darab S-sel

sorrendekvivalens szegmensnek S-et önmagát is beleértve. Vagyis létezik legalább még egy S-sel sorrendekvivalens szegmens. Legyen ennek a szegmensnek a közepe P_j. De ekkor a 3.5. lemma miatt a végrehajtás alatt P_i és P_j a vezető folyamat kiválasztásának pillanatáig ekvivalens állapotokban maradnak. Azt kapjuk, hogy a P_j folyamat szintén vezető folyamat lesz. Ellentmondásra jutottunk. □

Most bebizonyítjuk az alsó korlátot.

3.9. tétel . *Legyen A egy olyan öszehasonlításon alapú algoritmus, amely bármely n méretű gyűrűben kiválasztja a vezető folyamatot. Ekkor A-nak van olyan végrehajtása, amelyben a vezető folyamat kiválasztásának pillanatáig $\Omega(n \log n)$ üzenetetküldés történik.*

Megjegyezzük, hogy az $\Omega(n \log n)$ kifejezésben egy n-től független konstans szorzó is szerepel.

Bizonyítás. Legyen c egy olyan rögzített konstans, amelynek létezését a 3.7. tétel garantálja és tekintsük az ennek megfelelő n méretű c-szimmetrikus R gyűrűt. Most az A algoritmus R gyűrűbeli végrehajtásait fogjuk vizsgálni.

Legyen $k = \lfloor (cn-2)/4 \rfloor$. Ekkor $\sqrt{n} \leq 2k+1$ (feltéve, hogy n elég nagy) és $\lfloor cn/(2k+1) \rfloor \geq 2$. A 3.8. lemmából következik, hogy az algoritmus több, mint k (vagyis legalább $k+1$) aktív menetet hajt végre.

Vizsgáljuk meg az r-edik aktív menetet, ahol $\sqrt{n}+1 \leq r \leq k+1$. Mivel a futam aktív, létezik olyan P_i folyamat, ami az r-edik menetben üzenetet küld. Legyen S a P_i folyamat $(r-1)$-szomszédsága. Az R gyűrű c-szimmetrikus volta miatt létezik legalább $\lfloor cn/(2r-1) \rfloor$ darab S-sel ekvivalens szegmens R-ben. De a 3.5. lemma miatt az r-edik aktív menet előtti pillanatig ezen szegmensek középpontjai állapotai összhangban voltak, vagyis mindegyikük küldött üzenetet.

Legyen most $r_1 = \lceil \sqrt{n} \rceil + 1$ és $r_2 = \lfloor (cn-2)/4 \rfloor + 1$. Az iménti gondolatmenetből következik, hogy az üzenetek összes száma legalább

$$\sum_{r=r_1}^{r_2} \left\lfloor \frac{cn}{2r-1} \right\rfloor \geq \sum_{r=r_1}^{r_2} \frac{cn}{2r-1} - r_2.$$

Az összeg második tagja $O(n)$ nagyságrendű, vagyis elegendő azt belátni, hogy az első tag nagyságrendje $\Omega(n \log n)$. A kifejezést átalakítva kapjuk, hogy

$$\begin{aligned}
\sum_{r=r_1}^{r_2} \frac{cn}{2r-1} &= \Omega\left(n \sum_{r=r_1}^{r_2} \frac{1}{r} \right) \\
&= \Omega(n(\ln r_2 - \ln r_1)) \\
&= \Omega\left(n \left(\ln \left(\left\lfloor \frac{cn-2}{4} \right\rfloor + 1 \right) - \ln(\lceil \sqrt{n} \rceil + 1) \right) \right) \\
&= \Omega(n \log n).
\end{aligned}$$

A második sorban az összeg integrálközelítését használtuk. A bizonyítást befejeztük. □

3.7.. Alsó korlát a nem összehasonlításos algoritmusok üzeneteinek számára*

Vajon a nem összehasonlítás alapú algoritmusok esetében az üzenetek számára vonatkozóan milyen alsó korlát adható? Korábban láttuk, hogy $\Omega(n \log n)$-nél létezik jobb alsó korlát. Megmutatható, hogy ez a jobb korlát csak nagy időbonyolultság árán érhető el. Tegyük fel, hogy a vezető folyamat kiválasztásáig eltelt idő t-vel felülről korlátos. Ha az azonosítók állapotterében az UID-ek összes száma elég nagy – mondjuk nagyobb, mint valamilyen gyorsan növő $f(n,t)$ függvény –, akkor az azonosítóknak létezik egy olyan U részhalmaza, amelyen meg lehet mutatni, hogy az algoritmus legalább t meneten keresztül „összehasonlításos algoritmusként" viselkedik. Ebből az következik, hogy az összehasonlítások alsó korlátja átvihető az U-beli azonosítókat használó időkorlátos algoritmusokra.

Gondolatunkat ennél részletesebben is kifejtjük, de leírásunk így is nagyon vázlatos marad. A *Ramsey-tétel* segítségével – ami a skatulya elv egyfajta általánosítása – definiálni fogjuk az imént említett gyorsan növő $f(n,t)$ függvényt. A tétel állításában n-részhalmazon egy n elemű részhalmazt értünk, színezésen pedig minden halmazhoz valamilyen szín hozzárendelését.

3.10. tétel . (Ramsey-tétel) *Minden m, n és c egészhez található egy $g(n,m,c)$ egész az alábbi tulajdonsággal. Minden legalább $g(n,m,c)$ elemű S halmazhoz és az S halmaz n-részhalmazainak bármilyen legfeljebb c színt tartalmazó színezéséhez létezik S-nek olyan m elemű C részhalmaza, aminek az összes n-részhalmaza ugyanazzal a színnel színezett.*

Először az algoritmusokat olyan *normálformára* hozzuk, amelyben minden állapot a LISP nyelv S-kifejezéseinek formátumában rögzíti a kezdeti UID-eket, az összes valaha érkezett üzenetet és minden *nem-null* üzenet tartalmazza küldőjének teljes állapotleírását. Ezen S-kifejezések közül bizonyosak *kiválasztott* állapotokat jelölnek, amelyekben a folyamat vezető folyamatként azonosítható. Ha az eredeti algoritmus egy megfelelő vezető folyamat kiválasztó algoritmus volt, akkor az új (a módosított kimenetre vonatkozó megállapodásokkal) úgyszintén és a kommunikációs bonyolultságaik megegyeznek.

Az alsó korlátra vonatkozó tételünk az alábbit állítja.

3.11. tétel . *Minden n és t egészhez létezik egy $f(n,t)$ egész az alábbi tulajdonsággal. Legyen A egy tetszőleges (nem szükségképpen összehasonlításon alapuló) algoritmus, ami az n méretű gyűrűben t időn belül kiválasztja a vezető folyamatot úgy, hogy az UID-ek tere legalább $f(n,t)$ méretű. Ekkor A-nak létezik egy olyan végrehajtása, amelyben a vezető folyamat kiválasztásáig $\Omega(n \log n)$ üzenetküldés történik.*

Bizonyításvázlat. Legyen n és t rögzített. Az általánosság megszorítása nélkül feltehető, hogy az algoritmusok normálformában vannak. Mivel az algoritmusok n darab folyamaton futnak és t menetidőt használnak, minden előforduló S-kifejezés legfeljebb n különböző argumentumot és legfeljebb t mélységű egymásba ágyazást tartalmaz.

Az UID-ek n-halmazain (n elemű halmazokon) minden A algoritmusra definiálunk egy $\equiv_A$ ekvivalenciarelációt. Két n-halmaz akkor lesz ekvivalens, ha az A algoritmus ugyanúgy viselkedik rajtuk. Formálisan, legyen V és V' UID-ek két n-halmaza. Azt mondjuk, hogy $V \equiv_A V'$, ha az A algoritmus minden V feletti legfeljebb t mélységű egymásba ágyazást tartalmazó S-kifejezés és a neki megfelelő V' feletti S-kifejezés esetén (amit úgy kapunk, hogy V minden elemét a sorrendben ugyanannyiadik V'-belire cserélünk) ugyanarra a döntésre jut: küldjön vagy ne küldjön üzenetet, a folyamat válassza vagy ne önmagát vezetőnek.

Mivel az ekvivalenciareláció definíciójában az S-kifejezéseknek legfeljebb n argumentumuk van és legfeljebb t mélységű egymásba ágyazást tartalmaznak, csak véges sok ekvivalenciaosztály létezik. Valójában az osztályok számára létezik egy, az A algoritmustól független, csak n-től és t-től függő felső korlát. Jelöljük ezt a korlátot $c(n,t)$-vel.

Rögzítsük le az A algoritmust. Mivel alkalmazni szeretnénk a Ramsey-tételt, először egy módszert kell adnunk az UID-ek n-halmazainak színezésére. Feleltessünk meg egy színt az n-halmazok minden $\equiv_A$ ekvivalenciaosztályának és egy osztályon belül minden n-halmazt színezzünk ugyanazzal a színnel.

Legyen $f(n,t) = g(n, 2n, c(n,t))$, ahol g a 3.10. tételben szereplő függvény és tekintsük az UID-ek egy tetszőleges, legalább $f(n,t)$ azonosítót tartalmazó terét. Ekkor a 3.10. tétel miatt létezik az UID-ek terének egy olyan, legalább $2n$ elemet tartalmazó C részhalmaza, aminek minden n-részhalmaza ugyanazzal a színnel színezett. A C halmaz n legkisebb elemeiből álló halmazt jelöljük U-val.

Most azt állítjuk, hogy ha az UID-eket az U halmazból választjuk, az algoritmus t meneten keresztül ugyanúgy viselkedik, mint egy összehasonlításos algoritmus. Vagyis bármely folyamat bármilyen döntést is hoz arról, hogy küldjön-e üzenetet valamely irányba vagy a folyamat vezetőnek titulálja-e önmagát, a döntés csak az aktuális állapot argumentumainak relatív sorrendjétől függ. Ennek belátásához rögzítsük le U-nak bármely két, mondjuk m elemű W és W' részhalmazát. Tegyük fel, hogy S egy legfeljebb t egymásba ágyazást és W-beli UID-eket tartalmazó S-kifejezés és S' a neki megfelelő W' feletti S-kifejezés (amit úgy kapunk, hogy W minden elemét a sorrendben ugyanannyiadik W'-belire cserélünk). Ekkor W és W' kibővíthető az n elemű V és V' halmazokká úgy, hogy mindegyikükhöz hozzávesszük a C halmaz legnagyobb $n-m$ elemét. Mivel V és V' színezése megegyezik, a két S-kifejezés ugyanolyan döntést eredményez arról, hogy küldjenek-e üzenetet a szomszédaik felé vagy a folyamatok vezetőnek választják-e magukat.

Mivel az algoritmus t meneten keresztül pontosan úgy viselkedik, mint egy összehasonlítás alapú algoritmus (amennyiben az UID-eket az U halmazból választjuk), a 3.9. tételben bizonyított alsó korlát érvényben marad. □

3.8.. Megjegyzések a fejezethez

A 3.2. fejezet megoldhatatlansági eredménye a terület vizsgálatának legelejére nyúlik vissza. Az eredmény egy más modellre alkalmazott verziója Angluin [13] cikkében található meg. Az LCR algoritmus Le Lann [191] egy fejlesztéséből

származik, amit Chang és Roberts optimalizált [71]. A HS algoritmus Hirschberg és Sinclair munkája [156].

Az $O(n \log n)$ felső korlátjának konstans értékét folyamatosan javították, a jelenlegi érték $1.271n \log n + O(n)$ Higham és Przytycka [155] eredménye. A korlát egyirányú gyűrűkre is igaz. Peterson [239], valamint Dolev, Klawe és Rodeh [97] $O(n \log n)$-es algoritmust adtak az egyirányú esetre.

Az IdőSzelet algoritmus szintén ősi darab, amelynél a vezető folyamat kiválasztásának stratégiája hasonlít az MIT vezetőlőjeles gyűrű hálózatban használatoshoz. A VáltozóSebességek algoritmus Frederickson és Lynch [127], valamint tőlük függetlenül Vitányi [282] fejlesztése.

Az összehasonlítás alapú és a nem összehasonlítás alapú algoritmusokra vonatkozó alsó korlátok Frederickson és Lynch [127] eredményei. Attiya, Snir és Warmuth [27] a c-szimmetrikus gyűrűk másféle előállítását adták. A Ramsey-tétel a kombinatorika jól ismert eredménye, megtalálható például Berge [47] gráfelmélet könyvében.

Attiya, Snir és Warmuth [27] cikke más eredményeket is tartalmaz a szinkron gyűrűkben való számítások korlátairól. Az általuk használt bizonyítási technikák a 3.6. fejezetben látottakhoz hasonlatosak.

3.9.. Gyakorlatok

3-1. Finomítsuk az LCR algoritmus helyességét bizonyító induktív bizonyítás részleteit.

3-2. Tekintsük az LCR algoritmust.

(a) Adjunk olyan UID felsorolást, ahol az elküldött üzenetek száma $\Omega(n^2)$.

(b) Adjunk olyan UID felsorolást, ahol az elküldött üzenetek száma $O(n)$.

(c) Mutassuk meg, hogy az elküldött üzenetek átlagos száma $O(n \log n)$, ahol az átlagot a gyűrű folyamatainak összes lehetséges egyformán valószínű elrendezésén való működésből számoljuk.

3-3. Módosítsuk az LCR algoritmust úgy, hogy az összes nem-vezető folyamat a *nem_vezető* kimenetet eredményezze vagyis az összes folyamat végül is álljon meg. Adjuk meg a módosított algoritmust az LCR algoritmusnál látott kódolási stílus segítségével.

3-4. Mutassuk meg, hogy az LCR algoritmus a szinkron modell verzióban különböző induló időpontok mellett is helyesen működik. (Ehhez egy kicsit módosítsuk a kódot.)

3-5. Az LCR algoritmusnál látott invariáns állítások módszerével bizonyítsuk be a HS algoritmus helyességét.

3-6. Mutassuk meg, hogy a HS algoritmus a szinkron modell verzióban különböző induló időpontok mellett is helyesen működik. (Ehhez egy kicsit módosítsuk a kódot.)

3-7. Tegyük fel, hogy a HS algoritmust úgy módosítjuk, hogy kettőhatványok helyett egymás utáni k-hatványokat használunk az utak hosszára ($k > 2$). Elemezzük a módosított algoritmus idő és kommunikációs bonyolultságát úgy, ahogy tettük ezt az eredeti HS algoritmusnál. Hasonlítsuk össze az eredményeket.

3-8. Tekintsük a HS algoritmus olyan módosított változatát, ahol a folyamatok mindkét irány helyett csak az egyik irányba küldhetnek üzeneteket.

(a) Mutassuk meg, hogy a könyvben megadott algoritmus legkézenfekvőbb módosítása nem eredményez $O(n \log n)$ kommunikációs bonyolultságot. Mi lesz a kommunikációs bonyolultság egy felső korlátja?

(b) Egy kis ravaszkodással módosítsuk úgy az algoritmust, hogy kommunikációs bonyolultsága ismét $O(n \log n)$ legyen.

3-9. Tervezzünk egyirányú gyűrűben olyan vezető folyamat kiválasztásos algoritmust, ami nem ismeri a gyűrű méretét és legrosszabb esetben is csak $O(n \log n)$ számú üzenetet használ. Az algoritmus az UID-ekre kizárólag az összehasonlítás műveletet használhatja.

3-10. Kódoljuk le az IdőSzelet algoritmust az állapotgép segítségével.

3-11. Módosítsuk úgy az IdőSzelet algoritmust, hogy hozzávett üzenetek árán fázisonként egyetlen UID helyett k darab UID továbbküldésének engedélyezésével csökkenjen a futási idő. Bizonyítsuk be az algoritmus helyességét és elemezzük bonyolultságát.

3-12. Kódoljuk le a VáltozóSebességek algoritmust az állapotgép segítségével.

3-13. Mutassuk meg, hogy ha a folyamatok különböző időpontokban ébredhetnek fel, a VáltozóSebességek algoritmus kommunikációs bonyolultsága nem szükségszerűen $O(n)$.

3-14. Adjunk a menetek számára vonatkozó minél jobb *alsó* korlátot valamely n méretű gyűrű vezető folyamat kiválasztásos algoritmusának legrosszabb esetére. A feltevéseket körültekintően fogalmazzuk meg.

3-15. Adjuk meg az $n = 16$ csomópontú bitfordító gyűrű pontos leírását.

3-16. Bizonyítsuk be, hogy az $n = 2^k$ méretű bitfordító gyűrű minden $k \in \mathbb{N}$ esetén $\frac{1}{2}$-szimmetrikus.

3-17. Tervezzünk c-szimmetrikus gyűrűt nem kettőhatvány számú csomópont esetén valamilyen $c > 0$ értékre.

3-18. Valamely szinkron gyűrű esetén tekintsük a vezető folyamat kiválasztásának problémáját, ahol minden folyamat ismeri a gyűrű n méretét és a folyamatoknak nincs UID-jük. Adjunk a probléma megoldására *véletlenített algoritmust*, vagyis olyat, ahol a folyamatok 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 időbonyolultsága és kommunikációs bonyolultsága?

3-19. Tekintsünk valamilyen ismeretlen n méretű szinkron, mindkét irányban irányított gyűrűt, ahol a folyamatoknak van UID-jük. Adjunk az üzenetek számára vonatkozó alsó és felső korlátot olyan összehasonlítás alapú algoritmus esetén, ahol minden folyamat n mod 2 számol.

Tárgymutató