Tartalomjegyzék

6. fejezet

Egyetértés processzorhibák esetében

Ebben a fejezetben folytatjuk a szinkron modellre vonatkozó egyetértési/megegyezési feladat tárgyalását, melyet az 5. fejezetben kezdtünk el. Most azzal az esettel foglalkozunk, amelyben a folyamatok, és nem a vonalak hibázhatnak. Természetesen értelmesebb lenne fizikai „processzor" hibákról beszélni, mint logikai „folyamat" hibákról, de hogy következetesek maradjunk a könyv többi részében használt szaknyelvhez, a *folyamatok* kifejezést fogjuk használni. Kétfajta hibamodellt vizsgálunk meg: a *megállási hiba* modellt, melyben a folyamat minden előzetes értesítés nélkül megállhat, és a *bizánci hiba* modellt, melyben a folyamat teljesen tetszőlegesen viselkedhet. A megállási hibák a megjósolhatatlan processzor összeomlásokat hivatottak modellezni. A bizánci hibák a processzor egyfajta önkényes, hibás viselkedését modellezik, ide sorolható például, ha a processzoron belül valamelyik önálló egység meghibásodik.

A *bizánci* kifejezést az ilyenfajta hibákra először Lamport, Pease és Shostak használják egy irányt mutató cikkükben, melyben az egyetértési problémát *bizánci tábornokok* példáján keresztül vezetik be. Az 5. fejezet összehangolt támadási problémájához hasonlóan, a bizánci tábornokok megpróbálnak megegyezni, hogy támadjanak-e. Most viszont a tábornokoknak nincs félni valójuk amiatt, hogy a hírnökök elesnek, hanem a többi áruló tábornok viselkedésétől kell tartaniuk. A *bizánci* jelző egy szójáték eredménye – a történet az ókori Bizáncban játszódik, így lett a néhány áruló módon viselkedő tábornok jelzője „bizánci".

Azokban a feladatokban – egyszerűen *megegyezési feladatnak* nevezzük őket –, melyeket ebben a fejezetben tárgyalunk, a folyamatok egyedi kezdő értéküket egy V halmazból veszik. Az összes hibamentesen működő folyamat kimeneti értéke ugyanezen V halmazba tartozik a megegyezési és érvényességi feltételek értelmében. (Az érvényesség feltétele, hogy amikor minden folyamat ugyanazzal a v értékkel kezd, az egyetlen megengedett döntési érték v lehet.)

A megegyezési probléma egy egyszerűsített változata annak a problémának, mely eredetileg a repülőgépek fedélzeti ellenőrző rendszerével kapcsolatban merült fel. Ott processzorok egy csoportjának kell megegyeznie a repülési magasság tekintetében, úgy, hogy mindegyikük egy különálló magasságmérővel áll kapcso-

latban, amely magasságmérők között lehet hibás is. Bizánci megegyezési algoritmusokat alkalmaznak a hibatűrő többprocesszoros hardverek esetében is, amelyekben a processzorok egy kis csoportja ugyanazt a számítást végzi el, és lépésenként megegyezésre kell jutniuk. E redundanciának köszönhetően a processzorok eltűrik az egyik processzor bizánci hibáját. A bizánci algoritmusok hasznosak a hibadiagnózisban is, lehetővé teszik, hogy a processzorok megegyezzenek, melyik hibás közülük (ezért azt vagy helyettesíteni kell, vagy figyelmen kívül kell hagyni).

Mindkét hiba modellünk esetében egy határt kell szabnunk a processzorhibák előfordulásának gyakoriságára. Hogyan írhatók le ezek a korlátok? Egyéb olyan munkákban, ahol processzorhibákat tartalmazó rendszerek elemzésével foglalkoznak, a korlátok valószínűségi eloszlásokon alapulnak, mely eloszlások a hibák előfordulásait határozzák meg. Itt, valószínűség használata helyett, egyszerűen azt feltételezzük, hogy a hibák számát egy előre megadott, rögzített f érték korlátozza. Ez egy egyszerűen kezelhető megkötés, elkerülhetjük vele azt a bonyolultságot, melyet a hiba előfordulások valószínűségi leírása okozna. A gyakorlatban ez a megközelítés élethű lehet olyan értelemben, hogy nem tartjuk valószínűnek azt, hogy több mint f hiba lép fel. Mindamellett ne feledkezzünk meg róla, hogy e feltevésünk egy kissé vitatható: a legtöbb valóságos helyzetben, ha már nagy számú hiba lépett fel, valószínű, hogy még több hiba fog előfordulni. Feltevésünkből, hogy a hibák száma korlátos, az következik, hogy a hibák *korrelációja negatív*, ugyanakkor a valóságban a hibák vagy függetlenek, vagy korrelációjuk pozitív.

A megegyezési probléma definiálása után bemutatunk egy sor algoritmust a megállási hiba és a bizánci hiba kezelésére. Majd bebizonyítunk két alsó korlátot, egyet a processzorok számára, melyek képesek a bizánci hiba problémájának megoldására, egy másikat pedig azon menetek számára, melyek – mindkét hibatípus esetében – szükségesek a megoldáshoz.

6.1.. A feladat

Feltesszük, hogy a hálózat egy n-csúcsú összefüggő, irányítatlan gráf, $P_1, \ldots, P_n$ folyamatokkal, amelyben minden egyes folyamat ismeri a teljes gráfot. Feltesszük, hogy minden egyes folyamat egy adott rendszer-összetevőben tárolt, és egy rögzített V halmazba tartozó bemeneti értékkel kezd; feltesszük továbbá azt, hogy a folyamatok egy adott bemeneti értékéhez egy kezdőállapot tartozik. A cél, hogy a folyamatok – a *döntési* állapot összetevőjüket beállítva – olyan döntést hozzanak, mely a V halmazba tartozó érték. Itt is a 3–5. fejezetben tárgyalt szinkron modellt használjuk, csak most megengedjük, hogy korlátos számú (legfeljebb f) folyamat hibázhat. Ebben a fejezetben feltesszük, hogy a kapcsolatok teljesen megbízhatók – az összes elküldött üzenet megérkezik. Kétfajta folyamathibával foglalkozunk: a megállási hibával és a bizánci hibával.

A megállási hiba modellben a folyamat az algoritmus végrehajtásának bármely pontján egyszerűen leállhat és a továbbiakban már nem tart lépést a többiekkel. Az is előfordulhat, hogy a folyamat az *üzenetküldő lépés közepén* áll le; így abban a menetben, amikor a folyamat leállt, azoknak az üzeneteknek csak

egy *részhalmaza* indul el ténylegesen, amelyeket a folyamat eredetileg el akart küldeni. Előfordulhat processzor-megállás abban a pillanatban is, amikor a folyamat egy adott menetben az üzeneteit már elküldte, de épp az adott menetnek megfelelő átmenet előtt áll.

A megállási hiba modellben a megegyezési probléma helyességi feltételei az alábbiak.

Megegyezés. Nincs két olyan folyamat, mely eltérő döntést hoz.

Érvényesség. Ha mindegyik folyamat ugyanazzal a $v \in V$ kezdeti értékkel kezd, akkor egyedül v a lehetséges döntési érték.

Befejezés. Végül minden hibamentes folyamat döntést hoz.

A bizánci hiba modellben a folyamat hibájának nemcsak a megállás lehet az oka, hanem az önkényes viselkedés is. Ez azt jelenti, hogy a folyamat induláskor tetszőleges állapotban lehet, nemcsak a kezdőállapotainak egyikében; tetszőleges üzenetet küldhet, nemcsak a saját *üzenet* függvényének megfelelőt, és tetszőleges *állapot átmenetet* hajthat végre, nemcsak az *átmenet* függvényében meghatározott átmeneteket. (Gyakorlatilag – kényelmi okokból – azt az esetet is megengedjük, hogy egy bizánci folyamat teljesen hibátlanul működik.) Az egyetlen megkötés a hibás folyamat viselkedésére, hogy csak azon rendszeralkotókra lehet hatással, amelyek felett irányítási joga van, nevezetesen a saját kimenő üzeneteire és a saját állapotaira. Nem ronthatja el például a többi folyamat állapotát, nem módosíthatja vagy helyettesítheti mással azok üzeneteit.

A bizánci hiba modell megegyezési és érvényességi feltételei kissé eltérnek a megállási hiba modellnél megadottaktól.

Megegyezés. Nincs két olyan hibamentes folyamat, mely eltérő döntést hoz.

Érvényesség. Ha mindegyik hibamentes folyamat ugyanazzal a $v \in V$ kezdeti értékkel kezd, akkor egyedül v a lehetséges döntési érték a hibamentes folyamatoknál.

Befejezés. A befejezés feltétele ugyanaz.

A módosított feltételek azt a tényt tükrözik, hogy a bizánci modellben nem adhatunk korlátot a hibás folyamat kezdeti beállításaira és végső döntésére. A bizánci modell szerinti megegyezési problémára *bizánci megegyezési probléma* néven fogunk hivatkozni.

A megállási és a bizánci megegyezési probléma közötti összefüggés. Az nem igaz, hogy egy olyan algoritmus, amelyik megoldja a bizánci megegyezési problémát, automatikusan megoldja a megegyezési problémát megállási hibák esetében; az a különbség, hogy a megállási hiba modellben megkívánjuk, hogy az összes döntésre jutott folyamat megegyezzen, *még azok is, amelyek utólag meghibásodtak*. Ha kicseréljük a megállási hiba megegyezési feltételét a bizánci hibánál megadottal, akkor már helytálló a fenti következtetés. Egy másik lehetőség az, ha a bizánci algoritmusban az összes hibamentes folyamat mindig egyazon menetben hoz döntést, akkor megállási hibák esetére is működik az algoritmus. A bizonyításokat a 6-1. és 6-2. gyakorlatra hagyjuk.

Erősebb érvényességi feltétel megállási hibára. Egy másik érvényességi feltétel, melyet a megállási hiba modellben használnak, így szól.

Érvényesség. Tetszőleges folyamat tetszőleges döntési értékére igaz, hogy az kezdeti értéke volt valamelyik folyamatnak.

Könnyen belátható, hogy ez a feltétel magában foglalja az általunk korábban megadott érvényességi feltételt. Ezt az erősebb érvényességi feltételt fogjuk használni a megegyezési probléma általánosításánál, a 7. fejezetben tárgyalt k-megegyezési problémánál. Ebben a fejezetben a korábban megadott gyengébb feltételt használjuk; ez kissé gyengíti az algoritmussal kapcsolatos állításainkat, és kissé erősíti a megoldhatatlansági eredményeinket. E fejezet algoritmusainál külön kitérünk arra, hogy kielégítik-e ezt a szigorúbb érvényességi feltételt.

Bonyolultsági mértékek. Az időbonyolultság mértékéhez megszámoljuk, hány menet szükséges ahhoz, hogy az összes hibamentes folyamat döntésre jusson. A kommunikációs bonyolultság mértékéhez az elküldött üzenetek számát és az üzenetek bitszámát határozzuk meg; a megállási hiba modellben számításba vesszük az összes folyamat által küldött üzenetet, míg a bizánci hiba modellben csak a hibamentes folyamatokra alapozzuk a számítást. Ennek az az oka, hogy a bizánci modellben a hibás folyamatok kommunikációjára nem tudunk egyszerű korlátot adni.

6.2.. Algoritmusok megállási hibák kezelésére

Ebben az alfejezetben a megállási hiba modell megegyezési problémájára mutatunk be algoritmusokat, speciálisan az n-csúcsú teljes gráfok esetére. Egy alap algoritmussal kezdjük, melyben a folyamatok egyszerűen újra meg újra közzéteszik az összes általuk valaha megkapott értéket. Folytatjuk az alap algoritmus néhány csökkentett bonyolultságú változatával, és végül bemutatunk olyan algoritmusokat, melyek az *exponenciális információgyűjtés* ($EIGY$) néven ismert stratégiát használják. Bár az exponenciális információgyűjtő algoritmusok költségesebbek és némileg bonyolultabbak, a kevésbé jól viselkedő hibamodellekre is kiterjeszthetők.

Megállapodások. Ebben és a következő részben a v_0 jelölés egy előre meghatározott alapértéket jelöl, mely eleme a V bemeneti halmaznak. Úgyszintén használjuk a b jelölést azon bitek számának felső korlátjára, mely egy V-beli érték ábrázolásához szükséges.

6.2.1.. Egy alapvető algoritmus

A megállási hiba modell megegyezési problémájára van egy igen egyszerű algoritmus, melynek neve HalmazTerjed. A folyamatok egyre csak terjesztik az összes olyan V-beli értéket, melyről valaha tudomást szereztek, és végül egy egyszerű döntési szabály szerint döntenek.

HalmazTerjed algoritmus (vázlatosan)

Mindegyik folyamat egy W változót használ, melynek értéke a V halmaz egy részhalmaza. Kezdetben a P_i folyamat W változója csak P_i kezdeti értékét tartalmazza. Minden egyes menetben az $(f+1)$-edik menetig bezárólag, a folyamatok elküldik W-t, majd hozzáadják W-hez a kapott üzenet összes elemét.

$f+1$ menet után P_i a következő döntési szabályt alkalmazza. Ha W egy egyelemű halmaz, akkor P_i döntése az az érték, ami a W halmaz eleme; különben P_i a v_0 alapérték mellett dönt.

Az algoritmus kódja a következő.

HalmazTerjed algoritmus (formálisan)

Az üzenet ábécé V halmaz részhalmazaiból áll.

állapotok$_i$:
menetek $\in \mathbb{N}$, kezdetben 0
döntés $\in V \cup \{ismeretlen\}$, kezdetben *ismeretlen*
$W \subseteq V$, kezdetben egyelemű halmaz, amely kezdetben P_i kezdeti értékét tartalmazza

üzenetek$_i$:
if *menetek* $\leq f$ **then** küldd el W-t az összes többi folyamatnak

átmenet$_i$:
menetek := *menetek* + 1
legyen X_j a P_j-től érkezett üzenet minden olyan j esetében,
  amelyre P_j-től érkezett üzenet
$W := W \cup \bigcup_j X_j$
if *menetek* $= f+1$ **then**
  if $|W| = 1$ **then** *döntés* := v, ahol $W = \{v\}$
    else *döntés* := v_0

A HalmazTerjed algoritmus helyességének tárgyalása során $W_i(r)$ a P_i folyamat W változójának r menet utáni értékét fogja jelölni. Ahogy általában, az i

index a P_i folyamathoz tartozó rendszeralkotó jelölésére szolgál. Azt mondjuk, hogy a folyamat r menet után *aktív*, ha az r-edik menet végéig nem hibázik.

Az első egyszerű lemma szerint, ha létezik olyan menet, amelyben egyik folyamat sem hibázik, akkor a menet végén az összes aktív folyamat W halmaza megegyezik.

6.1. lemma . *Ha egy folyamat sem hibázik egy adott r-edik ($1 \leq r \leq f+1$) menetben, akkor $W_i(r) = W_j(r)$ minden olyan P_i és P_j folyamat esetében, amely az r-edik menet után aktív.*

Bizonyítás. Tegyük fel, hogy az r-edik menetben egy folyamat sem hibázik, és legyen I a folyamatoknak azon halmaza, amelyek aktívak az r-edik menet után (vagy ennek megfelelően, az $(r-1)$-edik menet után). Azt követően, hogy az összes I-beli folyamat elküldi a saját W halmazát az összes többi folyamatnak, az r-edik menet végén mindegyik I-beli folyamat W halmaza pontosan azoknak az értékeknek a halmaza, melyek az r-edik menet előtti valamely I-beli folyamat W halmazába tartoztak. □

A következő állításban belátjuk, hogy ha adott r menet után az aktív folyamatok W halmaza megegyezik, akkor ez igaz a rákövetkező menetekben is.

6.2. lemma . *Tegyük fel, hogy $W_i(r) = W_j(r)$ minden olyan P_i és P_j folyamat esetében, amely az r-edik menet után aktív. Akkor ez igaz lesz bármely r'-edik ($r \leq r' \leq f+1$) menetben is, azaz $W_i(r') = W_j(r')$ minden olyan P_i és P_j folyamat esetében, amely az r'-edik menet után aktív.*

Bizonyítás. A bizonyítást a 6-3. gyakorlatra hagyjuk. □

A következő lemma döntő fontosságú a megegyezési követelmény szempontjából.

6.3. lemma . *Ha P_i és P_j folyamatok mindketten aktívak $f+1$ menet után, akkor az $(f+1)$-edik menet végén $W_i = W_j$.*

Bizonyítás. Minthogy legfeljebb f hibás folyamat van, biztosan létezik egy olyan r-edik ($1 \leq r \leq f+1$) menet, melyben egyetlen folyamat sem hibázott. A 6.1. lemmából következik, hogy $W_i(r) = W_j(r)$ minden olyan P_i és P_j folyamat esetében, amely az r-edik menet után aktív. Ebből a 6.2. lemma szerint következik, hogy $W_i(f+1) = W_j(f+1)$ minden olyan P_i és P_j folyamat esetében, amely az $(f+1)$-edik menet után aktív. □

6.4. tétel . *A* HalmazTerjed *algoritmus megoldja a megegyezési problémát megállási hibák esetében.*

Bizonyítás. A befejezés nyilvánvalóan adódik a döntési szabályból. Az érvényesség bizonyításához tételezzük fel, hogy az összes kezdeti érték egyenlő v-vel. Ekkor a folyamatok egyedül a v értéket küldözgetik. A $W_i(f+1)$ halmazok egyike sem üres, minthogy P_i kezdeti értékét tartalmazzák. Ezekből következik, hogy

$W_i(f+1)$ egyedül a $\{v\}$ halmazzal lehet egyenlő, amiből a döntési szabály szerint adódik, hogy az egyetlen lehetséges döntési érték a v.

A megegyezést illetően, legyen P_i és P_j két tetszőleges folyamat, mely döntést hozott. Minthogy a döntések csak az $(f+1)$-edik menet végén keletkeznek, ez azt jelenti, hogy P_i és P_j aktívak az $(f+1)$-edik menet után. Ebből a 6.3. lemma szerint következik, hogy $W_i(f+1) = W_j(f+1)$. Ekkor a döntési szabály szerint P_i és P_j azonos döntést hoz. □

Bonyolultságelemzés. A HALMAZTERJED algoritmusnak pontosan $f+1$ menetre van szüksége, hogy az összes folyamat hibátlan folyamat döntést hozzon. Az üzenetek száma összesen $O((f+1)n^2)$. Minden egyes üzenet legfeljebb egy n elemű halmazt tartalmaz (minthogy a halmaz minden elemére igaz, hogy valamely folyamat kezdő értéke), így az üzenetenkénti bitszám $O(nb)$, ahol b az egy érték ábrázolásához szükséges bitek száma. Ebből adódik, hogy a kommunikáció bitszáma összesen $O((f+1)n^3b)$.

Másfajta döntési szabály. A HALMAZTERJED algoritmusnál megadott döntési szabály kicsit önkényes. Mivel a HALMAZTERJED algoritmus garantálja, hogy az összes hibamentes folyamat ugyanazzal a W halmazzal rendelkezik $f+1$ menet után, több döntési szabályt használva is helyesen fog működni az algoritmus mindaddig, míg az összes folyamat ugyanazon szabály alapján dönt. Például, ha a V halmaz elemein értelmezve van egy teljes rendezés, akkor mindegyik folyamat választhatná a W halmaz legkisebb elemét. Ennek a szabálynak megvan az az előnye, hogy kielégíti a 6.1. alfejezet vége felé közölt erősebb érvényességi feltételt. A HALMAZTERJED algoritmusnál megadott döntési szabály nem feltétlenül biztosítja ezen erősebb feltétel szerinti érvényességet, mivel előfordulhat, hogy a döntés alapértékeként használt v_0 egyetlen folyamatnak sem kezdeti értéke.

Összevetés a kommunikációs hibák esetével. A HALMAZTERJED algoritmus igazolja, hogy a megegyezési probléma megállási hibák esetében megoldható. Vessük össze ezt a pozitív eredményt az összehangolt támadási probléma megoldhatatlanságával kommunikációs hibákat tartalmazó környezetben. (Lásd 5.1. tétel és 5-1. gyakorlat)

6.2.2.. A kommunikációs bonyolultság csökkentése

Az információ mennyisége csökkenthető a HALMAZTERJED algoritmusban szükséges $O((f+1)n^2)$ üzenethez és $O((f+1)n^3b)$ bithez képest. Például, a szükséges üzenetek száma lecsökkenthető $2n^2$ értékre, a kommunikációhoz szükséges bitek száma pedig $O(n^2b)$ értékre, ha a következő egyszerű gondolatmenetet követjük. Vegyük észre, hogy a befejezésnél elegendő, ha a W_i halmazok elemeit csak akkor ismerik pontosan a folyamatok, ha $|W_i| = 1$, különben elegendő, ha azt a tényt tudják, hogy $|W_i| \geq 2$. Így kézenfekvő, hogy a folyamatoknak csak az általuk megismert *első két értéket* kell közvetíteniük, nem pedig az összes értéket. Ezen a gondolaton alapszik a következő algoritmus.

Az OPTHALMAZTERJED algoritmus (vázlatosan)

A folyamatok ugyanazt teszik, mint a HALMAZTERJED algoritmusban, kivéve, hogy minden egyes P_i folyamat legfeljebb két értéket közvetít. Az első közreadás az első menetben történik, amikor P_i a saját kezdeti értékét közvetíti. A második közreadás az első olyan r-edik ($2 \leq r \leq f+1$) menetben zajlik, amelyben igaz, hogy P_i már a menet elején ismer egy saját kezdeti értékétől eltérő v értéket (ha egyáltalán létezik ilyen menet). Ekkor P_i közreadja ezt az új v értéket. (Ha két vagy több új érték van ebben a menetben, akkor közülük bármelyiket közvetítheti.)

Ugyanúgy, mint a HALMAZTERJED algoritmusban, P_i döntése v lesz, ha a végső W_i halmaz az egyelemű $\{v\}$ halmazzal egyenlő, és v_0 különben.

Bonyolultságelemzés. Az OPTHALMAZTERJED algoritmusban a szükséges menetek száma éppúgy $f+1$, mint a HALMAZTERJED-ben. Az üzenetek száma legfeljebb $2n^2$, minthogy minden egyes folyamat legfeljebb két nem *null* üzenetet küld a többi folyamatnak. A kommunikáció bitszáma $O(n^2b)$.

Az OPTHALMAZTERJED algoritmus helyességét a a HALMAZTERJED algoritmusra vonatkozó *szimulációs relációra* támaszkodva fogjuk bizonyítani, (ehhez hasonló stratégiát használtunk a 4.1.3. szakaszban az OPTMAXTERJED algoritmus helyesség-bizonyításánál, felhasználva a MAXTERJED algoritmusra vonatkozó szimulációs kapcsolatát). Ehhez csakúgy, mint HALMAZTERJED-nél, meg kell adnunk az OPTHALMAZTERJED algoritmus leírásához szükséges részleteket, beleértve a következőket: *menetek*, *döntés*, W változó. A $W_i(r)$ jelölést használjuk a HALMAZTERJED algoritmusban r menet utáni W_i értékekre, míg az $OW_i(r)$ jelölést az OPTHALMAZTERJED algoritmus hasonló értékeire. A következő lemma az üzenetek terjedését írja le a HALMAZTERJED algoritmusban.

6.5. lemma . *Tegyük fel, hogy a* HALMAZTERJED *algoritmusban P_i folyamat küld egy üzenetet az $(r+1)$-edik menetben P_j folyamatnak, amelyet az megkap és feldolgoz. Ekkor $W_i(r) \subseteq W_j(r+1)$*

Bizonyítás. A bizonyítást a 6-9. gyakorlatra hagyjuk. □

A következő lemma fogalmazza meg az OPTHALMAZTERJED algoritmus kulcsfontosságú ritkító tulajdonságát.

6.6. lemma . *Tegyük fel, hogy az* OPTHALMAZTERJED *algoritmusban P_i folyamat küld egy üzenetet az $(r+1)$-edik menetben P_j folyamatnak, amelyet az megkap és feldolgoz. Ekkor*

1. *Ha $|OW_i(r)| = 1$, akkor $OW_i(r) \subseteq OW_j(r+1)$.*
2. *Ha $|OW_i(r)| \geq 2$, akkor $|OW_j(r+1)| \geq 2$.*

Sőt, e két állítás akkor is igaz, ha P_i folyamat hibátlanul működik az első r menetben és nem küld üzentet az $(r+1)$-edik menetben P_j-nek, minthogy az OPTHALMAZTERJED *algoritmusban nincs arra kikötés, hogy a folyamatnak ilyen üzenetet kellene küldenie.*

Bizonyítás. A bizonyítást a 6-9. gyakorlatra hagyjuk. □

Most futtassuk OptHalmazTerjed és HalmazTerjed algoritmusokat azonos bemenettel és azonos hibázási mintával. Ez azt jelenti, hogy mindkét végrehajtásban ugyanazok a folyamatok ugyanazokban a menetekben hibáznak. Azonkívül, ha P_i az r-edik menetbeli üzeneteinek csak egy részét küldi el az egyik algoritmusban, akkor ugyanazon folyamatoknak küld üzenetet a másik algoritmusban; még pontosabban, nincs olyan P_j, melynek P_i az egyik algoritmusban küld üzenetet az r-edik menetben, de a másikban nem. Invariáns állításokat adunk a két algoritmus állapotaira vonatkozólag.

6.7. lemma . *Tetszőleges r $(0 \leq r \leq f+1)$ menet után fennáll*

1. $OW_i(r) \subseteq W_i(r)$.
2. *Ha* $|W_i(r)| = 1$, *akkor* $OW_i(r) = W_i(r)$.

Bizonyítás. A bizonyítást a 6-9. gyakorlatra hagyjuk. □

6.8. lemma . *Tetszőleges r $(0 \leq r \leq f+1)$ menet után fennáll*

- *Ha* $|W_i(r)| \geq 2$, *akkor* $|OW_i(r)| \geq 2$.

Bizonyítás. A bizonyítás teljes indukcióval történik. Az $r = 0$ alapesetre az állítás nyilvánvaló. Tegyük fel, hogy a lemma igaz egy adott r-re. Megmutatjuk, hogy akkor igaz $(r+1)$-re is. Legyen $|W_i(r+1)| \geq 2$. Ha $|W_i(r)| \geq 2$, akkor az induktív feltevésből adódik, hogy $|OW_i(r)| \geq 2$, ebből viszont az következik, hogy $|OW_i(r+1)| \geq 2$, és ezt akartuk bizonyítani.

Most legyen $|W_i(r)| = 1$. Ekkor 6.7. lemmából következik, hogy $OW_i(r) = W_i(r)$. Bontsuk ketté a bizonyítást.

1. $|W_j(r)| = 1$ az összes P_j-re, melyektől P_i üzenetet kap a HalmazTerjed algoritmusban, az $(r+1)$-edik menetben.

   Ekkor a 6.7. lemma szerint az összes ilyen P_j-re $OW_j(r) = W_j(r)$ is fennáll, tehát $|OW_j(r)| = 1$. A 6.6. lemmából következik, hogy minden ilyen P_j-re $OW_j(r) \subseteq W_i(r+1)$. Ebből következik, hogy $OW_i(r+1) = W_i(r+1)$, amire az indukciós lépés bizonyításához szükségünk volt.
2. Legyen $|W_j(r)| \geq 2$, valamelyik olyan P_j-re, amelytől P_i üzenetet kap a HalmazTerjed algoritmusban, az $(r+1)$-edik menetben.

   Ekkor az indukciós feltevésből $|OW_j(r)| \geq 2$ adódik. Majd a 6.6. lemmából következik, hogy $|OW_i(r+1)| \geq 2$, amit igazolni akartunk.

□

6.9. lemma . *Tetszőleges r $(0 \leq r \leq f+1)$ menet után, a menetek és a döntési változók értékei megegyeznek a* HalmazTerjed *és az* OptHalmazTerjed *algoritmusokban.*

Bizonyításvázlat. A bizonyítás szempontjából az az érdekes, hogy tetszőleges P_i folyamatra igazoljuk, hogy az $(f+1)$-edik menet után mindkét algoritmusban ugyanazt a döntést hozza. Ez következik a 6.7. és 6.8. lemmából, és a két algoritmus döntési szabályaiból. □

6.10. tétel . *Az* OptHalmazTerjed *algoritmus megoldja a megegyezési problémát megállási hibák esetében.*

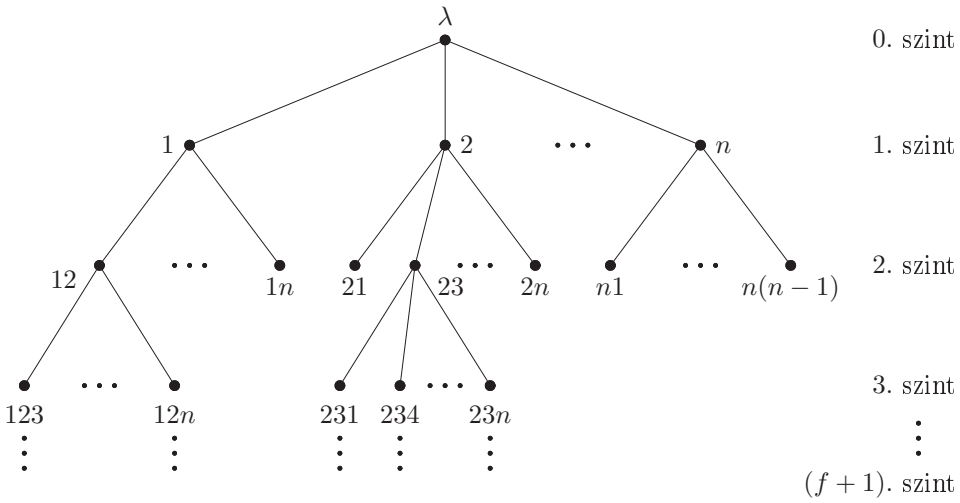
Bizonyítás. Az állítás következik 6.9. lemmából és a 6.4. tételből (amely a HalmazTerjed algoritmus helyességének tétele). □

Más módszerek a kommunikációs bonyolultság csökkentésére. Vannak más lehetőségek is a HalmazTerjed algoritmus kommunikációs bonyolultságának csökkentésére. Emlékezzünk vissza például arra, hogy ha a V halmazon értelmezve van egy teljes rendezés, a döntési szabályt módosíthatjuk úgy, hogy egyszerűen válasszuk W halmaz legkisebb elemét. Ekkor a HalmazTerjed algoritmust módosíthatjuk úgy, hogy a csúcspontok csak az általuk megismert legkisebb értéket jegyzik meg és továbbítják, nem az összes értéket. Ennek az algoritmusnak $O((f+1)n^2b)$ a kommunikációs bitszáma, ami korrekt módon bizonyítható a HalmazTerjed algoritmusra (a módosított döntési szabállyal) vonatkozó szimulációval. Ez az algoritmus kielégíti a 6.1. fejezetben adott erősebb érvényességi feltételt is.

6.2.3.. Exponenciális információgyűjtő algoritmusok

Ebben a részben olyan, az *exponenciális információgyűjtés* (*EIGY*) stratégián alapuló algoritmusokat mutatunk be, melyek megoldják megegyezési problémát megállási hibák esetében. Az exponenciális információgyűjtésen alapuló algoritmusokban a folyamatok több meneten keresztül küldik és továbbítják a kezdeti értékeket, miközben az általuk megismert, különféle kommunikációs utakon érkezett értékeket egy *EIGY fa* nevű adatszerkezetben rögzítik. Végül egy közösen elfogadott szabály szerint döntenek a fájukban tárolt értékek alapján.

Az *EIGY* algoritmusok általában költségesek a megegyezés megoldásához a megállási hiba modellben, mind a kommunikációhoz szükséges bitek számát tekintve, mind a felhasznált helyi tárolómennyiséget tekintve. A fő ok, amiért itt tárgyaljuk ezt a stratégiát, hogy ugyanezt az *EIGY* fa adatszerkezetet felhasználhatjuk a bizánci megegyezési probléma megoldásához a 6.3.2. szakaszban. A megállási hiba esete bevezet minket az adatszerkezet használatába. A második oka ezen stratégia bemutatásának megállási hibák esetére, hogy az egyszerű *EIGY* megállási hiba algoritmusok könnyen alkalmazhatók a megegyezési probléma megoldására a bizánci hiba modell *hitelesített bizánci hiba* néven ismert, leszűkített változatában. Az *EIGY* algoritmusok alapvető adatszerkezete egy megcímkézett $T = T_{n,f}$ *EIGY fa*, melyben a gyökértől induló utak folyamatok egy láncolatát ábrázolják, melyeken át a kezdeti értékek elterjednek; mindegyik ábrázolt lánc különböző folyamatokból áll. A T fának $f+2$ szintje van, a 0-adik szinttől (a gyökértől) az $(f+1)$-edik szintig (a levelekig). Minden k-adik $(0 \leq k \leq f)$ szinten lévő csúcsnak pontosan $n-k$ gyereke van. T mindegyik csúcsát megcímkézzük egy a folyamatoktól függő szöveggel az alábbi módon. A gyökér címkéje az üres szöveg, jelölje λ, továbbá bármely $i_1 \dots i_k$ szöveggel címkézett csúcsnak pontosan $n-k$ gyereke van, melyek címkéje $i_1 \dots i_k j$, ahol j végigfut $\{1, \dots, n\} - \{i_1, \dots, i_k\}$ elemein. Lásd a 6.1. ábrát.



6.1.. ábra. A $T_{n,f}$ *EIGY* fa.

A megállási hibákra vonatkozó *EIGY* algoritmusban, melyet EIGYSTOP-nak hívunk, a folyamatok az értékeket egyszerűen az összes lehetséges útra szétküldik. Mindegyik folyamat fenntartja az *EIGY* fa egy $T = T_{n,f}$ példányát. A számítás pontosan $f+1$ meneten keresztül zajlik. A számítás folyamán az egyes folyamatok feldíszítik a fájuk csúcsait a V halmazba tartozó értékekkel, vagy a *null* értékkel, a k-adik menet végére k-adik szintig elhelyezkedő összes csúcsot feldíszítik. P_i fájának gyökerére P_i bemeneti értéke kerül. Amennyiben P_i fájának egy csúcsa az $i_1 \dots i_k$ $(1 \leq k \leq f+1)$ szöveggel van megcímkézve, és a $v \in V$ értékkel feldíszítve, akkor ez azt jelenti, hogy P_{i_k} a k-adik menetben azt mondta P_i-nek, hogy $P_{i_{k-1}}$ a $(k-1)$-edik menetben azt mondta P_{i_k}-nak, hogy ..., hogy P_{i_1} az első menetben azt mondta P_{i_2}-nek, hogy P_{i_1} kezdeti értéke v. Másrészről, ha egy $i_1 \dots i_k$ szöveggel megcímkézett csúcsot a *null* értékkel díszítjük fel, akkor az azt jelenti, hogy a $P_{i_1}, P_{i_2}, \dots, P_{i_k}, P_i$ folyamatok közti kommunikáció hiba miatt megszakadt. $f+1$ menet után a folyamatok a saját maguk által feldíszített fát használják, hogy egy V halmazba tartozó döntést hozzanak, egy (fentebb leírt) közösen elfogadott döntési szabály alapján. Most az algoritmus részletesebb tárgyalása következik.

Az algoritmus most következő leírásában és néhány későbbiben is kényelmesebb, ha úgy tekintjük, mintha valamennyi P_i folyamat önmagának is küldhetne üzenetet a többi folyamatnak küldött üzenet mellett; ez segít bennünket abban, hogy az algoritmusok leírása egységesebb legyen. Ezek az üzenetek a modellben technikailag nincsenek engedélyezve, de kárt nem okozunk, ha megengedjük őket, mert a színlelt átvitelek helyi számításokkal szimulálhatók.

EIGYSTOP algoritmus (vázlatosan)

Legyen minden folyamatnak, az összes olyan x szövegre, mely a T fa egy

csúcsának címkéjeként előfordul, egy *érték*(x) nevű változója. A *érték*(x) változó tárolja azt az értéket, mellyel a folyamat az x címkéjű csúcsot feldíszíti, a *érték*(λ) változót a folyamat saját kezdeti értékére állítja.

Első menet: P_i folyamat elküldi *érték*(λ)-t az összes folyamatnak, beleértve önmagát is. Majd P_i feljegyzi a beérkező információkat:

1. ha P_i-hez a $v \in V$ üzenetet érkezik P_j-től, akkor P_i a saját *érték*(j) változóját v-re állítja;
2. ha nem érkezik P_i-hez P_j-től olyan üzenet, melynek értéke a V halmazból való, akkor P_i a *érték*(j) változóját *null*-ra állítja.

k-adik menet $(2 \leq k \leq f+1)$: P_i szétküldi az (x, v) párokat, ahol x egy T-beli $(k-1)$-edik szinten lévő csúcsnak a címkéje, ami nem tartalmazza az i indexet, $v \in V$ és $v =$ *érték*(x).[1] Majd P_i feljegyzi a beérkező információkat:

1. ha xj a T fában a k-adik szinten lévő csúcsnak a címkéje, ahol x szöveg folyamatok indexeinek sorozata, a j pedig egy index, és a P_i-hez P_j-től érkező üzenet szerint *érték*$(x) = v \in V$, akkor P_i a *érték*(xj)-t v-re állítja;
2. ha xj a T fában a k-adik szinten lévő csúcsnak a címkéje, és nem érkezik P_i-hez olyan üzenet P_j-től, mely szerint *érték*(x) egy V halmazbeli érték, akkor P_i a *érték*(xj) változóját *null*-ra állítja.

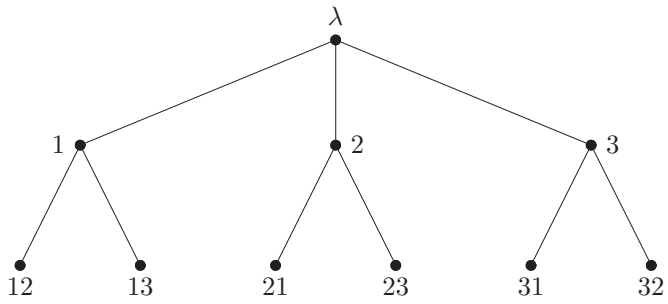
Az $(f+1)$-edik menet végén P_i egy döntési szabályt alkalmaz. Nevezetesen, W halmaz elemei legyenek P_i folyamat fáját díszítő nem *null érték*-ek. Ha W halmaz egyelemű, akkor P_i döntése ez az egyedüli elem; egyébként pedig P_i a v_0 döntést hozza.

Nem nehéz belátni, hogy a fákat a már korábban bemutatott értékek fogják díszíteni. Tehát a P_i folyamat fájának gyökerét P_i kezdeti értéke díszíti. Továbbá, ha P_i fájának egy csúcsa az $i_1 \dots i_k$, $1 \leq k \leq f+1$ szöveggel van megcímkézve, és a $v \in V$ értékkel feldíszítve, akkor ez azt jelenti, hogy P_{i_k} a k-adik menetben azt mondta P_i-nek, hogy $P_{i_{k-1}}$ a $(k-1)$-edik menetben azt mondta P_{i_k}-nak, hogy …, hogy P_{i_1} az első menetben azt mondta P_{i_2}-nek, hogy P_{i_1} kezdeti értéke v. Valamint, ha P_i fájának egy az $i_1 \dots i_k$, $1 \leq k \leq f+1$ szöveggel megcímkézett csúcsa a *null* értékkel van feldíszítve, akkor az azt jelenti, hogy P_{i_k} nem küldött üzenetet P_i-nek a k-adik menetben, mely továbbította volna az $i_1, \dots, i_{k-1}$-nek megfelelő értéket.

6.2.1. példa. Az EIGYStop algoritmus működése

Vizsgáljuk meg egy példán, hogyan működik az EIGYStop algoritmus, tekintsük azt az esetet, amikor három folyamatunk van $(n = 3)$, amelyikből az egyik lehet, hogy hibás $(f = 1)$. Ekkor a protokoll 2

[1] Ha eleget akarunk tenni a formális modellünknek, melyben P_i folyamat csak egy-egy üzenetet küldhet a többieknek egy adott meneten belül, azokat az üzeneteket, melyeknek a címzettje azonos, összecsomagoljuk egy nagy üzenetbe.



6.2.. ábra. A $T_{3,1}$ *EIGY* fa szerkezete.

menetben zajlik, a fának 3 szintje van. A $T_{3,1}$ *EIGY* fa szerkezete a 6.2. ábra szerinti.

Legyen P_1 és P_2 folyamatok kezdeti értéke 0, míg P_3 folyamaté 1. Tegyük fel, hogy P_3 folyamat hibázik: miután az első menetben elküldte az üzenetet P_1-nek, hibássá válik, és P_2-nek nem küld üzenetet. A 6.3. ábra mutatja, hogy ezek után milyen értékekkel lesznek a folyamatok fái kitöltve.

Figyeljük meg, hogy P_2-nek egészen addig nem jut tudomására, hogy P_3 folyamat kezdeti értéke 1, amíg a második menetben meg nem hallja P_1-től.

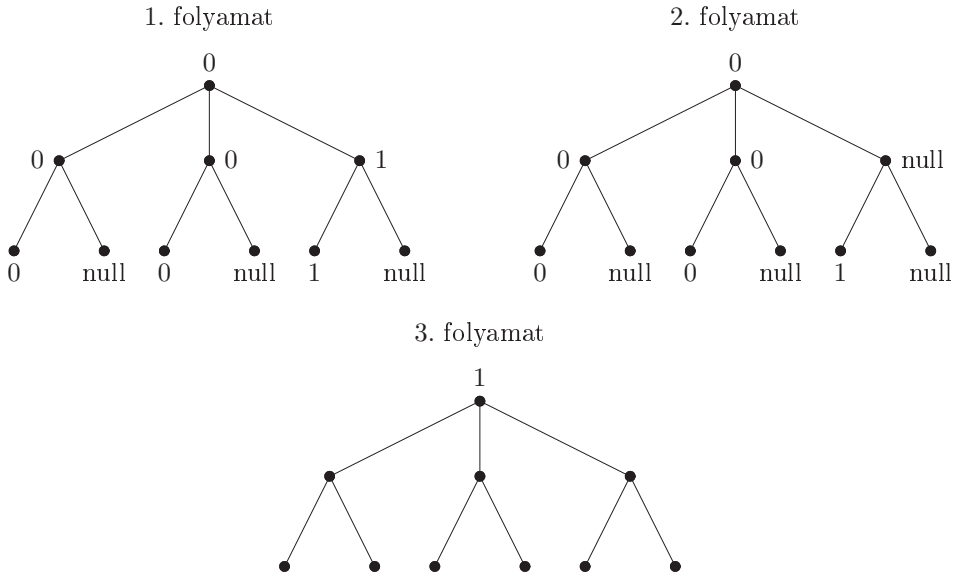
Hogy az EIGYSTOP algoritmus helyességét beláthassuk, először két lemmát mondunk ki, melyek a különböző fák értékeire vonatkoznak. Az első lemma az inicializálást és a fák szomszédos szintjein lévő folyamatokhoz tartozó *érték*-ek közti összefüggést írja le.

6.11. lemma . *Az* EIGYSTOP *algoritmusban az* $(f+1)$*-edik menet után fennáll, hogy:*

1. *érték*$(\lambda)_i$ P_i *bemeneti értéke;*
2. *ha* xj *egy csúcs címkéje, és érték*$(xj)_i = v \in V$*, akkor érték*$(x)_j = v$*;*
3. *ha* xj *egy csúcs címkéje, és érték*$(xj)_i =$ *null, akkor vagy érték*$(x)_j =$ *null, vagy* P_j *hibás lett, mielőtt* P_i*-nek üzenetet küldött volna az* $(|x|+1)$*-edik menetben.*

Bizonyítás. A bizonyítást a 6-12. gyakorlatra hagyjuk. □

A második lemma a fa nem feltétlenül szomszédos szintjein lévő csúcsaihoz tartozó *érték*-ek közötti összefüggést írja le. Az első két feltétel a fában tetszőleges helyen szereplő értékek eredetét mutatja. A harmadik egy technikai feltétel, mely azt állítja, hogy bármely a fában megjelenő v-nek meg kell jelennie egy olyan csúcsnál, melynek címkéje nem tartalmazza az i indexet. Lazán fogalmazva, ez azt jelenti, hogy amikor P_i először hall egy adott értékről, az nem lehet annak az eredménye, hogy az értéket saját magának küldte el.



6.3.. ábra. Az EIGYSTOP algoritmus végrehajtása; P_3 folyamat hibázik az első menetben.

6.12. lemma . *Az* EIGYSTOP *algoritmusban az* $(f+1)$*-edik menet után fennáll, hogy:*

1. *ha* y *egy csúcs címkéje,* $\textit{érték}(y)_i = v \in V$ *és* xj *az* y *prefixe, akkor* $\textit{érték}(x)_j = v$*;*
2. *ha a* $v \in V$ *érték megjelenik, valamely folyamat érték változóinak értékeiből álló halmaz elemeként, akkor van olyan* i*, hogy* $v = \textit{érték}(\lambda)_i$*;*
3. *ha* $v \in V$ *megjelenik, mint a* P_i *folyamat érték változóinak értékeiből álló halmaz egy eleme, akkor van olyan* y *címke, mely nem tartalmazza* i*-t és* $v = \textit{érték}(y)_i$*.*

Bizonyítás. Az állítás első része a 6.11. lemma második részének ismételt alkalmazásával adódik.

A második rész bizonyításához tegyük fel, hogy $v = \textit{érték}(y)_i$. Ha $y = \lambda$, akkor készen vagyunk. Egyébként jelölje j az első indexet y-ban. A lemma első részéből következik, hogy $v = \textit{érték}(\lambda)_j$.

A harmadik rész bizonyításához tegyük fel az állítás ellenkezőjét, hogy v egyedül olyan címkékhez tartozó *érték*-ként jelenik meg, melyekben az i szerepel. Legyen y a legrövidebb olyan címke, melyre $v = \textit{érték}(y)_i$. Ekkor y-nak van egy xi alakú prefixe. De ebből az első rész szerint következik, hogy $\textit{érték}(x)_i = v$, ami ellentmond y kiválasztásának. □

A következő lemma adja a kulcsot a megegyezési feltételhez.

6.13. lemma . *Ha* P_i *és* P_j *folyamatok hibamentesek, akkor* $W_i = W_j$*.*

Bizonyítás. Feltehetjük, hogy $i \neq j$. Belátjuk, hogy a tartalmazási reláció mindkét irányból fennáll.

1. $W_i \subseteq W_j$.
   Tegyük fel, hogy $v \in W_i$. Ekkor a 6.2. lemmából adódik, hogy $v = \textit{érték}(x)_i$ fennáll egy olyan x esetében, mely az i indexet nem tartalmazza. Tekintsük a következő két esetet.
   (a) $|x| \leq f$.
   Ekkor $|xi| \leq f + 1$, és mivel az x szöveg nem tartalmazza az i indexet, a (hibamentes) P_i folyamat a $|xi|$-edik menetben közvetíti P_j folyamatnak a v értéket. Ebből következik, hogy $\textit{érték}(xi)_j = v$, tehát $v \in W_j$.
   (b) $|x| = f + 1$.
   Ekkor, mivel a hibás folyamatok száma legfeljebb f, és az x szövegben az összes index különböző, létezik egy olyan P_l hibamentes folyamat, amelynek indexe szerepel az x-ben. Ezért x-nek van egy yl alakú prefixe, ahol y egy szöveg. Innen a 6.12. lemma szerint adódik, hogy $\textit{érték}(y)_l = v$. Mivel P_l hibamentes, ezért az $|yl|$-edik menetben közvetíti P_j-nek a v értéket. Ebből viszont a $\textit{érték}(yl)_j = v$ adódik, tehát ismét azt kapjuk, hogy $v \in W_j$.
2. $W_j \subseteq W_i$.
   Szimmetrikus az előző esetre.

A két eset együttesen bizonyítja a szükséges egyenlőséget. □

6.2.2. példa. A 6.1.3. lemma bizonyításának esetei

A 6.2.1. példa szemlélteti a 6.13. lemmában tárgyalt (a) és (b) esetet. Elsőként a P_1 folyamat díszíti fel a fáját az 1 értékkel az első menetben, és mivel nem ez az utolsó menet, ezért az (a) eset értelmében P_2 a második menetben díszíti fel a fáját az 1 értékkel. Pontosabban $\textit{érték}(3)_1 = 1$, így $\textit{érték}(31)_2 = 1$.

Másrészről viszont, P_2 folyamat először az utolsó, második menetben díszíti a fáját az 1 értékkel, elvégezve a $\textit{érték}(31)_2 = 1$ értékadást. Ebből következik, hogy egy hibamentes folyamat indexe, ebben az esetben az 1, megjelenik a csúcs címkéjében. Innen a (b) eset értelmében az 1 érték a P_1 fájában a 31 címkéjű csúcsnál jelenik meg. Tehát $\textit{érték}(31)_2 = 1$, így $\textit{érték}(31)_1 = 1$.

6.14. tétel . *Az* EIGYSTOP *algoritmus megoldja a megegyezési problémát megállási hibák esetére.*

Bizonyítás. A befejezés nyilvánvalóan adódik a döntési szabályból.

Az érvényesség bizonyításához tételezzük fel, hogy az összes kezdeti érték egyenlő v-vel. Ekkor a 6.12. lemma szerint az egyedüli értékek, melyekkel a folyamatok a fáikat díszítik, a v és a *null*. A W_i halmazok egyike sem üres, minthogy P_i kezdeti értékét tartalmazzák. Ebből következik, hogy mindegyik W_i halmaz

csak a $\{v\}$ halmazzal lehet egyenlő, amiből a döntési szabály alapján az egyetlen lehetséges döntési érték a v.

A megegyezést illetően, legyen P_i és P_j két tetszőleges folyamat, mely döntést hozott. Minthogy a döntés a végrehajtás végén történik, ez azt jelenti, hogy P_i és P_j folyamatok hibamentesek. Ebből a 6.13. lemma szerint következik, hogy $W_i = W_j$. Ekkor a döntési szabályból adódóan P_i és P_j azonos döntést hoz. □

Bonyolultságelemzés. A menetek száma $f + 1$, az elküldött üzenetek száma $O((f + 1)n^2)$. (Ez az egyesített üzenetek száma, melyet tetszőleges folyamat tetszőleges menetben küld a többi folyamatnak, mint szóló üzenet.) A kommunikáció *bitjeinek* száma exponenciális a hibák számának függvényében: $O(n^{f+1}b)$.

Másfajta döntési szabály. Mivel az EIGYSTOP algoritmus biztosítja, hogy a hibamentes folyamatok fájában előforduló értékek halmaza ugyanaz a W halmaz, több másfajta döntési szabály is helyesen működhet. Például, ha a V halmaz értékein adott egy teljes rendezés, akkor az összes folyamat választhatja a W halmaz legkisebb elemét. Ahogy korábban láttuk, ennek az az előnye, hogy a 6.1. alfejezetben említett erősebb érvényességi feltételt is kielégíti.

A HALMAZTERJED algoritmusnál látott módszerrel csökkenthető az EIGYSTOP algoritmus kommunikáció igénye. Ahogy már láttuk, elegendő, ha P_i folyamat csak abban az esetben ismeri a W_i halmaz elemeit pontosan, amikor $|W_i| = 1$. Így ismét elegendő, ha a folyamat csak az általa megismert első két értéket közvetíti.

Az OPTEIGYSTOP algoritmus (vázlatosan)

A folyamatok ugyanazt teszik, mint az EIGYSTOP algoritmusban, kivéve, hogy minden egyes P_i folyamat legfeljebb két értéket közvetít. Az első közreadás az első menetben történik, amikor P_i a saját kezdeti értékét közvetíti. A második közreadás az első olyan r $(2 \leq r \leq f + 1)$ menet valamelyikében zajlik, amelyben igaz, hogy P_i már a menet elején ismer egy saját kezdeti értékétől eltérő v értéket (ha egyáltalán létezik ilyen menet). Ekkor P_i közreadja ezt az új v értéket, azzal az $(r - 1)$-edik szinten lévő csúcsnak a címkéjével, mely a v értékkel van feldíszítve. (Ha két vagy több lehetséges (x, v) páros van, akkor közülük bármelyiket választhatja közvetítésre.)

Ugyanúgy, mint az EIGYSTOP algoritmusban, legyen W azon nem *null értéke*-k halmaza, melyek P_i folyamat fáját díszítik. Ha a W halmaz egyelemű, P_i döntése a W-beli elem lesz, különben pedig v_0.

Bonyolultságelemzés. Az OPTEIGYSTOP algoritmus $f + 1$ menetet használ. Az üzenetek száma legfeljebb $2n^2$, mivel mindegyik folyamat feljebb két nem *null* üzenetet küld a többi folyamatnak. $O(n^2(b + (f + 1)\log n))$ a kommunikáció bitjeinek száma: az üzenetek érték részéhez $O(b)$ bit szükséges, míg a címke rész $O((f + 1)\log n)$ bitből áll.

Az OPTEIGYSTOP algoritmus helyességét az EIGYSTOP algoritmusra vonatkozó szimulációs kapcsolat alapján bizonyíthatjuk. A bizonyítás hasonló az OPTHALMAZTERJED algoritmus helyességének bizonyításához. Másik lehetőség,

ha az OptEIGYStop algoritmus helyességének bizonyítását az OptHalmaz-Terjed algoritmusra vonatkoztatva végezzük el. A részleteket a 6-17. gyakorlatra hagyjuk.

6.2.4.. Bizánci megegyezés hitelesítéssel

Bár a fejezetben leírt *EIGY* algoritmusok csak a megállási hibák eltűrésére hivatottak, mégis alkalmasak rosszabb természetű hibák elviselésére is. Nem tudnak viszont megbirkózni a bizánci hiba modell összes nehézségével, mely modellben a folyamatok önkényesen viselkedhetnek. Mégis, megfelelnek a bizánci hiba modell egy érdekes leszűkítésében, melyben a folyamatoknak egy extra lehetőségük van, *hitelesíthetik* a kommunikációjukat *digitális aláírás* használatával. A P_i folyamat digitális aláírása egy transzformáció, melyet P_i a kimenő üzeneteire alkalmazhat, hogy bizonyíthassa, hogy az üzenet valóban tőle származik. P_i aláírásának előállítására – P_i együttműködése nélkül – egyetlen másik folyamat sem képes. A digitális aláírások valódi lehetőségek, melyek a modern kommunikációs hálózatokban alkalmazhatók.

A hitelesítéses bizánci hiba modell formális definícióját nem adjuk meg – mivel nem ismerünk rá szép definíciót – csak vázlatosan írjuk le. Ebben a modellben feltesszük, hogy a folyamatok digitális aláírást használhatnak, hogy hitelesítsék bármely kimenő üzenetüket. Az irodalomban általában feltételezik, hogy a kezdeti értékek egy közös forrásból származnak, mely szintén aláírással látja el azokat; itt most azt feltételezzük, hogy mindegyik hibamentes folyamat egy egyszerű, a forrás által aláírt bemenő értéket tartalmazó kezdő állapotból indul, míg mindegyik hibás folyamat egy tetszőleges, a forrás által aláírt (szignált) kezdeti értéket tartalmazó tetszőleges állapotból indul. A hibás folyamatok tetszőleges üzeneteket küldhetnek és tetszőleges állapot átmeneteket hajthatnak végre; az egyetlen megkötés, hogy nem képesek a hibamentes folyamatok és a forrás aláírásának előállításra.

A helyesség feltételei ebben a modellben a bizánci megegyezésben szokásos befejezési és megegyezési feltételek, és a következő érvényességi feltétel.

Érvényesség. Ha mindegyik folyamat pontosan ugyanazon, a forrás által szignált $v \in V$ kezdeti értékkel kezd, akkor csak a v a lehetséges döntési érték a hibamentes folyamatok számára.

Nem nehéz belátni, hogy az EIGYStop és OptEIGYStop algoritmusok azzal a módosítással, hogy minden üzenet szignált, és csak a helyesen szignált üzeneteket fogadjuk el, megoldják a megegyezési problémát a hitelesítéses bizánci hiba modellben. A bizonyítás hasonló, mint a megállási hiba modellben, a 6-18. gyakorlatra hagyjuk.

6.3.. Algoritmusok bizánci hibák kezelésére

Ebben a fejezetben, a bizánci megegyezésre mutatunk algoritmusokat, speciálisan n-csúcsú, teljes gráfok esetére. Egy olyan algoritmussal kezdjük, mely az expo-

nenciális információgyűjtést használja. Majd megmutatjuk, hogy egy olyan algoritmus, mely megoldja a bizánci megegyezés problémát egy kételemű $V = \{0, 1\}$ értékhalmazra, hogyan használható „szubrutinként" a bizánci megegyezés megoldására egy általános V értékhalmaz esetében. Végül bemutatunk egy csökkentett kommunikációs bonyolultságú bizánci megegyezés algoritmust.

Ezen algoritmusok közös tulajdonsága, hogy *több, mint háromszor annyi* folyamat szükséges, mint amennyi a hibák száma: $n > 3f$. Ez a körülmény más, mint a megállási hiba esetében, mivel ott nem volt speciális kikötés az n és f közötti összefüggésre. Ez a korlát a folyamatok számát tekintve jelzi a bizánci hiba modell fokozott bonyolultságát. Sőt, a 6.7. alfejezetben meg fogjuk látni, hogy ez a korlát velejárója a problémának. Ez elsőre meglepőnek tűnhet, mivel úgy képzelnénk, hogy $2f + 1$ folyamat képes eltűrni f bizánci hibát a többségi szavazás algoritmus valamely változatát használva. (Van egy szabvány hibatűrő technika, mely *háromszoros-modul redundancia* néven ismert, abban egy adott feladat három példányban fut és a többségi eredményt fogadjuk el; azt gondolhatnánk, hogy ez használható a bizánci megegyezés megoldásához egy hibás folyamat esetében, de látni fogjuk, hogy ez nem lehetséges.)

6.3.1.. Egy példa

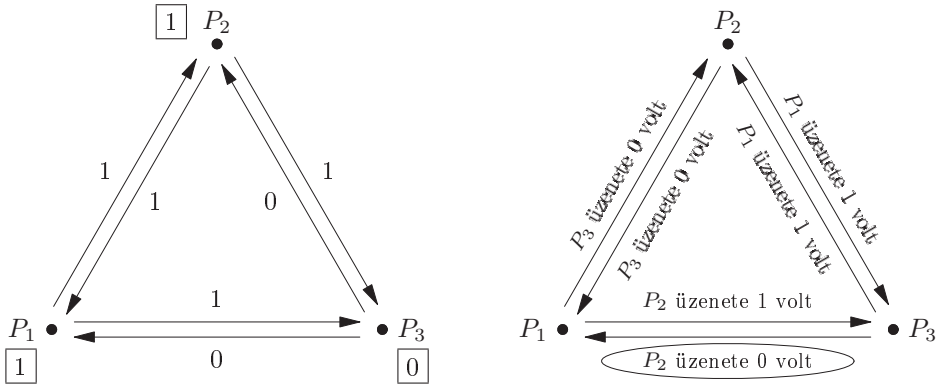
Mielőtt az *EIGY* bizánci megegyezés algoritmust bemutatjuk, gondolkodjunk el azon, hogy miért bonyolultabb a bizánci megegyezési probléma, mint a megegyezési probléma megállási hibák esetében. Egy példán keresztül megvilágítjuk (bár nem bizonyítjuk), hogy három folyamat nem képes megoldani a bizánci megegyezést, ha felmerül annak lehetősége, hogy az egyikük hibázik.

Tegyük fel, hogy P_1, P_2 és P_3 folyamat megoldja a bizánci megegyezési problémát, egy hibát eltűrve. Tegyük fel például, hogy két menet van, a folyamatok a második végén döntenek, valamint egyénre szabott, meghatározott módon cselekszenek: az első menetben mindegyik közreadja a saját kezdeti értékét, míg a második menetben mindegyik jelenti a többinek, hogy mit hallott az első menetben a harmadiktól. Tekintsük a következő végrehajtási sorozatot.

> *Az α_1 végrehajtási sorozat*
> P_1 és P_2 hibátlanok, és az 1 kezdeti értékkel indulnak, P_3 viszont hibázni fog, és a 0 kezdeti értékkel indul. Az első menetben mindegyik folyamat az igazságnak megfelelően küldi el saját értékét. A második menetben P_1 és P_2 az igazságnak megfelelően jelenti, hogy mit hallott az első menetben, míg P_3 a P_1-nek (hibásan) azt üzeni, hogy az első menetben P_2 üzenete 0 volt, egyébként helyesen viselkedik. A 6.4. ábrán láthatjuk az α_1 végrehajtási sorozatban küldött, minket érdeklő üzeneteket. Ebben a végrehajtási sorozatban az érvényességi szabály értelmében P_1 és P_2 mindketten az 1 döntést hozzák.

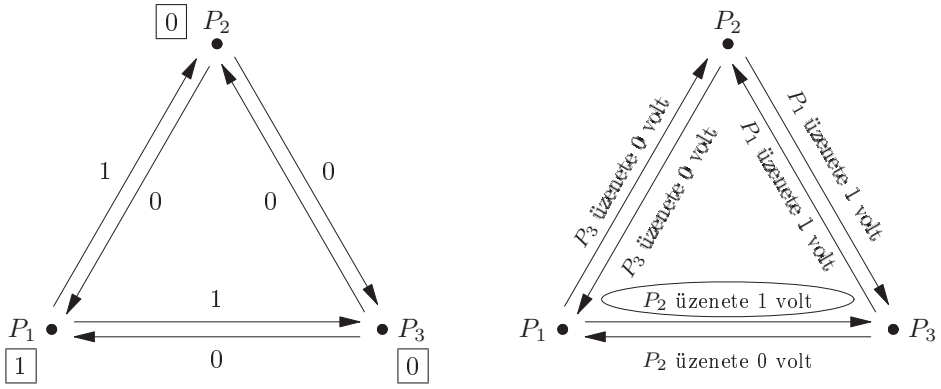
Most tekintsük a második végrehajtási sorozatot.

> *Az α_2 végrehajtási sorozat*
> A végrehajtás szimmetrikus α_1-re. Most P_2 és P_3 hibátlanok, és a



6.4.. ábra. Az α_1 végrehajtási sorozat – hibás üzenet kering.

0 kezdeti értékkel indulnak, P_1 viszont hibázni fog, és az 1 kezdeti értékkel indul. Az első menetben mindegyik folyamat az igazságnak megfelelően küldi el saját értékét. A második menetben P_2 és P_3 az igazságnak megfelelően jelenti, hogy mit hallott az első menetben, míg P_1 a P_3-nak (hibásan) azt üzeni, hogy az első menetben P_2 üzenete 1 volt, egyébként helyesen viselkedik. A 6.5. ábrán láthatjuk az α_2 végrehajtási sorozatban küldött, minket érdeklő üzeneteket. Ebben a végrehajtási sorozatban az érvényességi szabály értelmében P_2 és P_3 mindketten az 0 döntést hozzák.

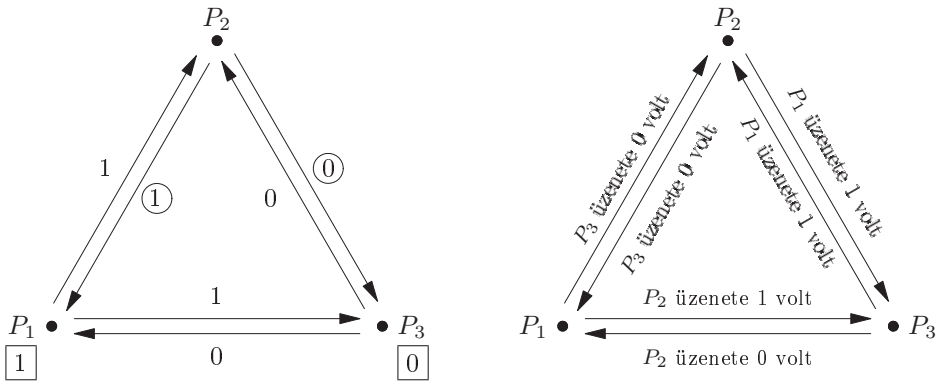


6.5.. ábra. Az α_2 végrehajtási sorozat – hibás üzenet kering.

Nézzük a harmadik végrehajtási sorozatot, mely ellentmondásra vezet.

Az α_3 végrehajtási sorozat
Most legyen P_1 és P_3 hibátlan, az egyik 1, a másik 0 értékkel indul. P_2

hibásan P_1-nek azt mondja, hogy 1 a kezdeti értéke, P_3-nak pedig azt, hogy 0. A második menetben mindegyik folyamat helyesen viselkedik. A helyzetet a 6.6. ábra mutatja.



6.6.. ábra. Az α_3 végrehajtási sorozat – egymással ütköző üzenetek keringenek.

Vegyük észre, hogy mindkét menet alatt P_2 ugyanazokat az üzeneteket küldi P_1-nek az α_3-ban, mint az α_1-ben, és ugyanazokat az üzeneteket küldi P_3-nak az α_3-ban, mint az α_2-ben. Ebből könnyen belátható, hogy P_1 számára az α_3 és α_1 megkülönböztethetetlen, azaz $\alpha_3 \overset{1}{\sim} \alpha_1$, és hasonlóan $\alpha_3 \overset{3}{\sim} \alpha_2$. Mivel P_1 az α_1-ben 1-est dönt, ugyanígy tesz α_3-ban, és mivel P_3 döntése 0 az α_2-ben, ugyanígy tesz α_3-ban. Ez viszont megsérti a megegyezési szabályt α_3-ban, mely ellentmond annak, hogy P_1, P_2 és P_3 megoldják a bizánci megegyezési problémát. Megmutattuk, hogy ebben a felállásban nincs olyan algoritmus, mely meg tudja oldani a bizánci megegyezést.

Megjegyezzük, hogy P_1 például megmondhatná, hogy α_3-ban valamelyik folyamat hibás, mivel P_2 azt mondja P_1-nek, hogy a kezdeti értéke 1, de P_3 azt mondja P_1-nek, hogy neki P_2 azt mondta, hogy 0 a kezdeti értéke. A probléma ott van, hogy P_1 nem tudja eldönteni hogy P_2 és P_3 közül melyik a hibás.

A példa nem bizonyítja azt, hogy három folyamat nem tudja megoldani a bizánci megegyezést egyszeri hibalehetőség mellett. Ennek az az oka, hogy az érvelés előfeltétele volt, hogy az algoritmus két menetet használ, és az üzenetek egy adott típusba tartoznak. Lehetséges azonban a példa kiterjesztése több menetre és tetszőleges üzenetekre. Meg fogjuk látni a 6.4. alfejezetben, hogy elgondolásunk odáig kiterjeszthető, hogy $n > 3f$ számú folyamat szükséges a bizánci megegyezés megoldásához, f hibát feltételezve.

6.3.2.. EIGY algoritmus a bizánci megegyezésre

Most bemutatunk egy *EIGY* algoritmust a bizánci megegyezésre, a neve legyen EIGYBIZ. Ellentétben az EIGYSTOP algoritmussal, EIGYBIZ előfeltétele, hogy a folyamatok száma nagy a hibákéhoz képest, nevezetesen $n > 3f$. Ez a 6.3.1.

és a 6.4. alfejezetekben leírt korlátok miatt szükséges. Mielőtt elolvasnánk az algoritmus leírását, próbáljunk egy saját megoldást készíteni az $n = 7$ és $f = 2$ esetre.

Az EIGYBiz algoritmus n számú folyamat és f számú hiba esetében ugyanazt a $T_{n,f}$ *EIGY* fa adatszerkezetet használja, mint az EIGYStop. Lényegében ugyanazt a terjesztési stratégiát használjuk, mint az EIGYStop-ban; az egyetlen különbség, hogy az a folyamat, amely egy „helytelen formájú" üzenetet kap, kijavítja az információt, hogy elfogadható legyen. A döntési szabály egészen más, jóllehet – ez már többé nem az az eset, melyben a folyamatok megbízhatnak az összes olyan értékben, amely a fájukban valahol megjelenik. Most a folyamatok kénytelenek tenni valamit, hogy leleplezzék a hibás üzenetben érkező értékeket.

Az EIGYBiz algoritmus (vázlatosan)
A folyamatok $f + 1$ meneten keresztül terjesztik az értékeket, pontosan úgy, mint az EIGYStop algoritmusban, a következő kivételekkel. Ha P_i folyamat valaha is kapna valamely másik P_j-től egy üzenetet, mely nem felel meg a leírásnak (például a tartalma teljesen használhatatlan, vagy P_j fájában ugyanahhoz a csúcshoz kettős értéket rendel), akkor P_i „eldobja" az üzenetet, azaz úgy tesz, mintha P_j semmit sem küldött volna az adott menetben.

Az $(f + 1)$-edik menet végén P_i megigazítja a *érték* hozzárendeléseket, a *null* értékeket az alapértékként adott v_0 értékkel helyettesíti.

Ezután, hogy P_i meghatározhassa a döntését, elindul a kiigazított, feldíszített fájában a levelektől felfelé, hogy minden egyes csúcsot feldíszítsen egy *új_ érték* értékkel a következők szerint. Minden x címkéjű levélre, *új_ érték*(x) := *érték*(x). Minden x címkéjű nem levél csúcsra *új_ érték*(x) definíció szerint legyen az az *új_ érték*, melyet az x csúcs gyerekeinek szigorúan vett többsége tárol, pontosabban legyen az a $v \in V$ elem, melyre *új_ érték*$(xj) = v$ az xj alakú csúcsok többségénél, feltéve, hogy ilyen többség létezik. Ha nem létezik a többség, P_i az *új_ érték*$(x) := v_0$ beállítást hajtja végre. P_i végső döntése *új_ érték*(λ).

Az EIGYBiz algoritmus helyességének bizonyítását néhány előzetes állítással kezdjük. Az első azt mondja ki, hogy az összes hibátlan folyamat olyan értékek alapján jut megegyezésre, melyek közvetlenül hibátlan folyamatoktól származnak.

6.15. lemma . *Az* EIGYBiz *algoritmus* $(f+1)$*-edik menete után fennáll a következő. Ha* P_i*,* P_j *és* P_k *hibátlan folyamatok és* $i \neq j$*, akkor érték*$(x)_i = $ *érték*$(x)_j$ *minden* x *címkére, mely* k*-ra végződik.*

Bizonyítás. Ha $k \notin \{i, j\}$, akkor az állítás abból a tényből adódik, hogy P_k hibamentes, ezért ugyanazt az üzenetet küldi P_i-nek és P_j-nek az $|x|$-edik menetben. Ha $k \in \{i, j\}$, akkor hasonlóan következik abból a megállapodásból, mely szerint a folyamatok saját maguknak is közvetítik az értékeket. □

A következő lemma azt állítja, hogy az összes hibátlan folyamat azon *új_ érték* értékek alapján jut megegyezésre, melyeket olyan csúcsokra számítottunk ki, amelyek címkéje egy hibamentes folyamat indexére végződik.

6.16. lemma . *Az* EIGYBIZ *algoritmus* $(f+1)$*-edik menete után fennáll a következő. Tegyük fel, hogy* x *címke egy hibátlan folyamat indexével végződik. Ekkor van olyan* $v \in V$*, hogy* $\textit{érték}(x)_i = \textit{új\_ érték}(x)_i = v$ *minden hibátlan* P_i *folyamatra teljesül.*

Bizonyítás. A bizonyítás indukcióval történik a fa címkéi alapján, a levelektől felfelé haladva – tehát az $f+1$ hosszúságtól az 1-ig csökkenően.

Alapeset: legyen x egy levél, azaz $|x| = f+1$. Ekkor a 6.15. lemmából következik, hogy az összes hibátlan P_i folyamatra a $\textit{érték}(x)_i$ érték ugyanaz; hívjuk ezt a közös értéket v-nek. Továbbá az is igaz, hogy $\textit{új\_ érték}(x)_i = v$, minden hibátlan P_i-re, a *új_ érték* levelekre adott definíciója alapján. Így v a kívánt érték.

Indukciós lépés: legyen $|x| = r$, $1 \leq r \leq f$. Ekkor a 6.15. lemmából következik, hogy az összes hibátlan P_i folyamatra a $\textit{érték}(x)_i$ érték ugyanaz; hívjuk ezt az értéket v-nek. Ezért az összes hibátlan P_l folyamat ugyanazt a v, x-hez tartozó értéket küldi el az $(r+1)$-edik menetben, így $\textit{érték}(xl)_i = v$ minden hibamentes P_i és P_l folyamatra. Az indukciós feltevésből következik, hogy az $\textit{új\_ érték}(xl)_i = v$ is teljesül minden hibamentes P_i és P_l folyamatra.

Azt állítjuk, hogy az x csúcs gyerekeinél a többség címkéje hibátlan folyamat indexére végződik. Ez igaz, mert x gyerekeinek száma pontosan $n - r \geq n - f$. Valamint feltettük, hogy $n > 3f$, ez a szám szigorúan nagyobb, mint $2f$, és minthogy a gyerekek közül legfeljebb f-nek a címkéje végződik hibás folyamat indexével, a szükséges többség megvan.

Ebből következik, hogy bármely hibamentes P_i-re az x csúcs xl gyerekeinek többségére $\textit{új\_ érték}(xl)_i = v$ fennáll. Ekkor az algoritmusban használt többségi szabályból következik, hogy $\textit{új\_ érték}(x)_i = v$ a hibátlan P_i folyamatoknál. Így v a kívánt érték. □

Ellenőrizzük az érvényességet.

6.17. lemma . *Ha mindegyik hibátlan folyamat ugyanazzal a* $v \in V$ *kezdeti értékkel kezd, akkor egyedül* v *a lehetséges döntési érték a hibátlan folyamatoknál.*

Bizonyítás. Ha az összes hibátlan folyamat v-vel kezd, akkor az összes hibátlan folyamat az első menetben közreadja v-t, ezért $\textit{érték}(j)_i = v$ minden hibátlan P_i és P_j folyamatra. A 6.16. lemmából következik, hogy $\textit{új\_ érték}(j)_i = v$ minden hibátlan P_i és P_j folyamatra. Ekkor az algoritmusban használt többségi szabályból adódik, hogy $\textit{új\_ érték}(\lambda)_i = v$ minden hibátlan P_i-re. Így P_i döntése v, amire szükségünk volt. □

A megegyezési tulajdonság bizonyításához két új definícióra van szükségünk. Az első: egy gyökeres fa csúcsainak egy C halmaza *útvonal lefedés*, ha minden gyökértől levélhez vezető útvonal legalább egy C-beli csúcsot tartalmaz.

A második: tekintsük az EIGYBIZ algoritmus egy tetszőleges α végrehajtási sorozatát. A fa egy x csúcsa *közös* az α-ban, ha az α végrehajtási sorozat $(f+1)$-edik menetében az összes hibamentes P_i ugyanazon $\textit{új\_ érték}(x)_i$ értékkel rendelkezik. A fa csúcsainak egy halmaza (például egy útvonal lefedés) *közös* az α-ban, ha a halmazba tartozó csúcsok mindegyike közös az α-ban. Vegyük észre, hogy a 6.16. lemma szerint, ha P_i hibamentes, akkor minden x-re az xi egy közös csúcs.

6.18. lemma . *Az* EIGYBIZ *algoritmus tetszőleges végrehajtásának* $(f+1)$*-edik menetében létezik egy olyan útvonal lefedés, mely közös az* α*-ban.*

Bizonyítás. Legyen C azon xi alakú csúcsok halmaza, ahol P_i hibamentes. Ahogy fentebb már észrevettük, minden C-beli csúcs közös. Megmutatjuk hogy C egy útvonal lefedés. Tekintsünk egy tetszőleges, a gyökértől egy levélhez vezető útvonalat. Ez pontosan $f+1$ nem gyökér csúcsot tartalmaz, és T felépítéséből adódóan mindegyik csúcs címkéje különböző folyamat indexével végződik. Mivel legfeljebb f hibás folyamat van, az útvonalon kell egy olyan csúcsnak lennie, mely címkéje hibátlan folyamat indexére végződik. Ez a csúcs viszont eleme C-nek. □

A következő lemma megmutatja, hogyan terjeszkednek felfelé a fában a közös csúcsok.

6.19. lemma . *Az* EIGYBIZ *algoritmus* $f+1$ *menete után igaz az alábbi. Legyen* x *egy tetszőleges címke az EIGY fában. Ha létezik közös útvonal lefedés az* x *gyökerű részfában, akkor az* x *közös.*

Bizonyítás. A bizonyítás indukcióval történik a fa címkéi alapján a levelektől a gyökér felé haladva.

Alapeset: legyen x egy levél. Ekkor az x részfa útvonal lefedése csak magából az x csúcsból áll. Így x közös, ami a bizonyítandó volt.

Indukciós lépés: legyen $|x| = r$, $0 \leq r \leq f$. Tegyük fel, hogy az x részfájának van egy C közös útvonal lefedése. Ha x eleme C-nek, akkor x közös, és kész vagyunk, ezért legyen $x \notin C$.

Vegyük x egy *tetszőleges* xl gyerekét. Mivel $x \notin C$, így C generál egy útvonal lefedést az xl gyökerű részfában. Ezért az indukciós feltevés szerint xl közös. Mivel xl tetszőlegesen kiválasztott gyereke volt x-nek, x összes gyereke közös. Ekkor a *új_érték*(x) definíciójából következik, hogy x közös. □

Egyszerű következményként kapjuk az alábbit.

6.20. lemma . *Az* EIGYBIZ *algoritmus* $f+1$ *menete után a* λ *gyökér közös.*

Bizonyítás. A 6.18. és 6.19. lemmákból adódik. □

Most összekötve a részeket, lássuk a fő helyességi tételt.

6.21. tétel . *Az* EIGYBIZ *algoritmus megoldja a bizánci megegyezési problémát* n *folyamat és* f *hiba esetében, ha* $n > 3f$ *teljesül.*

Bizonyítás. A befejezés nyilvánvaló. Az érvényesség a 6.17. lemmából következik. A megegyezés a 6.20. lemmából és a döntési szabályból adódik. □

Bonyolultságelemzés. A költségek ugyanazok, mint az EIGYSTOP algoritmus esetében: $f+1$ menet, $O((f+1)n^2)$ üzenet és $O(n^{f+1}b)$ a kommunikáció bitszáma. Ezenkívül új követelményként megkívánjuk, hogy a folyamatok száma legyen nagy a hibákhoz képest: $n > 3f$.

6.3.3.. A bináris bizánci megegyezésen alapuló általános bizánci megegyezés

Ebben a szakaszban megmutatjuk, hogyan használható szubrutinként egy olyan algoritmus, mely megoldja a bizánci megegyezést $\{0,1\}$ bemenetre, az általános bizánci megegyezés megoldásához. A plusz költség két többlet-menet, $2n^2$ többlet-üzenet és $O(n^2b)$ bit a kommunikációra. Ez tekintélyes megtakarításhoz vezethet a kommunikációhoz szükséges bitek számát tekintve, mivel nem szükséges, hogy V-beli értékeket küldjünk, elegendő bináris értékek küldése a szubrutin végrehajtása alatt. Bár ez előrelépés, de nem elegendő arra, hogy a kommunikációhoz szükséges bitek száma exponenciálisról polinomiálisra csökkenjen f függvényében.

Az algoritmus neve, a tervezői után, TurpinCoan. Az algoritmus felteszi, hogy $n > 3f$. Ahogy korábban, most is feltesszük, hogy a folyamat önmagának ugyanúgy küldhet üzenetet, mint a többi folyamatnak.

TurpinCoan algoritmus (vázlatosan)

Minden egyes folyamatnak lokális változói az x, y, z és *szavaz*, x kezdetben a folyamat bemeneti értékét veszi fel, y, z és *szavaz* tetszőleges kezdőértékkel indul.

Első menet: P_i folyamat elküldi saját x változójának értékét az összes folyamatnak, beleértve önmagát is. Ha az ebben a menetben érkezett üzenetekben egy adott v érték $\geq n-f$ példányban fordul elő, akkor P_i az $y := v$ beállítást hajtja végre, egyébként az $y :=$ *null* értékadást.

Második menet: P_i folyamat elküldi saját y változójának értékét az összes folyamatnak, beleértve önmagát is. Ha az ebben a menetben érkezett üzenetekben egy adott V-beli érték $\geq n-f$ példányban fordul elő, akkor P_i a *szavaz* $:= 1$ beállítást hajtja végre, egyébként a *szavaz* $:= 0$ értékadást. Ugyanekkor, P_i a z változóját a menetben kapott üzenetekben szereplő nem *null* értékek közül a leggyakoribbra állítja be, döntetlen esetében tetszőlegesen döntve; ha mindegyik üzenet *null*, akkor z határozatlan értékű marad.

r-edik menet ($r \geq 3$)*:* A folyamatok a bináris bizánci megegyezés algoritmusát futtatják, *szavaz* értékét bemenő értékként használva. Ha P_i döntése 1 a szubrutinban, és z értéke definiált, akkor az algoritmus végső döntése z, egyébként az alapértékként megadott v_0 a döntés.

Egy kulcsfontosságú tény a TurpinCoan algoritmusnál a következő.

6.22. lemma . *Legfeljebb egy olyan $v \in V$ érték van, melyet a hibamentes folyamatok egymásnak küldenek a második menetben.*

Bizonyítás. Tegyük fel, az állítás ellenkezőjét, hogy a P_i és P_j hibamentes folyamatok a második menetben külön-külön a v és w értékeket tartalmazó üzenetet küldik, $v, w \in V$ és $v \neq w$. Ekkor P_i legalább $n-f$ számú, v-t tartalmazó első menetbeli üzenetet kapott. Minthogy legfeljebb f hibás folyamat lehet, és a hibátlan folyamatok ugyanazt az üzenetet küldik az összes folyamatnak az első menetben,

így P_j legalább $n-2f$ darab v-t tartalmazó üzenetet kap. Minthogy $n > 3f$, P_j-hez legalább $f+1$ darab v-t tartalmazó üzenet érkezik.

Viszont abból, hogy P_j a második menetben w-t küld, az következik, hogy legalább $n-f$ darab első menetbeli w-t tartalmazó üzenetet kapott, ezek összege legalább $(f+1)+(n-f) > n$ üzenet. Azonban a P_j folyamat az első menetben csak n üzenetet kaphatott, tehát ellentmondásra jutottunk. □

6.23. tétel . *A* TurpinCoan *algoritmus megoldja az általános bizánci megegyezési problémát a bináris bizánci megegyezés algoritmusát szubrutinként használva, ha $n > 3f$ fennáll.*

Bizonyítás. A befejezés egyszerűen belátható.

Az érvényesség belátásához be kell bizonyítanunk, hogy ha az összes hibamentes folyamat ugyanazzal a v kezdeti értékkel indul, akkor az összes hibamentes folyamat döntése v lesz. Ezért tegyük fel, hogy mindegyik hibamentes folyamat v értékkel kezd. Ezután az összes $\geq n-f$ hibamentes folyamat sikeresen szétküldi az első menetbeli v-t tartalmazó üzeneteket valamennyi folyamatnak. Így az első menetben az összes hibamentes folyamat az y változóját v értékre állítja. Majd a második menetben mindegyik hibátlan folyamat legalább $n-f$ darab v-t tartalmazó üzenetet kap, amiből az következik, hogy a z változójukat v-re állítják, *szavaz* változójukat pedig 1-re. Mivel az összes hibamentes folyamat az 1 bemenetet alkalmazza a bináris bizánci megegyezés szubrutinjában, mindannyiuk döntése 1 lesz a szubrutinban, az algoritmus érvényességi feltételének megfelelően. Ez viszont azt jelenti, hogy a fő algoritmusban mindegyikük döntése v lesz, ami épp az érvényesség feltétele.

Végül belátjuk a megegyezést. Ha a szubrutin döntési értéke 0, akkor a v_0 értéket választja valamennyi hibátlan folyamat a végső döntésként, így a megegyezés alapértelmezés szerint fennáll.

Ezért tegyük fel azt, hogy a szubrutin döntése az 1 érték. Ekkor a szubrutinra vonatkozó érvényességi szabály szerint néhány hibamentes P_i folyamatra a szubrutint kezdetekor a $szavaz_i = 1$ fennáll. Ez azt jelenti, hogy P_i legalább $n-f$ darab, egy adott $v \in V$ értéket tartalmazó, második menetbeli üzenetet kap, és minthogy legfeljebb f hibás folyamat van, így P_i legalább $n-2f$ darab v-t tartalmazó üzenetet kap a hibamentes folyamatoktól. Továbbá, ha P_j egy tetszőleges hibátlan folyamat, akkor a P_j is legalább $n-2f$ darab v-t tartalmazó, második menetbeli üzenetet kap ugyanazon hibátlan folyamatoktól. A 6.22. lemma szerint csak egy adott $v \in V$ értéket küldhetnek a hibátlan folyamatok a második menetben. Ezért P_j nem kaphat f-nél több olyan üzenetet, melynek egy másik, v-től különböző V-beli elem az értéke (ezek a hibás folyamatok üzenetei). Mivel $n > 3f$, így $n-2f > f$, ezért a v érték a leggyakoribb a P_j folyamathoz érkezett második menetbeli üzenetek között. Ebből következik, hogy P_j a második menetben a $z := v$ értékadást végzi el. Mivel a szubrutin döntési értéke 1, ez azt jelenti, hogy P_j döntése v. Minthogy ez tetszőleges hibamentes P_j-re fennáll, a megegyezést beláttuk. □

A TurpinCoan algoritmus bizonyításánál a hibás folyamatok darabszámára adott korlátot felhasználtuk arra, hogy segédállításokat kapjunk a különböző folyamatok nézetei közti hasonlóságról a végrehajtás során. Ezt a fajta érvelést

egyéb egyetértési algoritmusra vonatkozó bizonyításnál is használjuk, például a *közelítő megegyezésnél* a 7.2. fejezetben.

Bonyolultságelemzés. A menetek száma $r+2$, ahol r a bináris bizánci megegyezés szubrutin meneteinek száma. A többlet-kommunikáció, amit a TurpinCoan algoritmus a szubrutinon kívül használ, $2n^2$ üzenet, mindegyik legfeljebb b bites, tehát a bitszám összesen $O(n^2b)$ bit.

6.3.4.. A kommunikációs költség csökkentése

Bár a TurpinCoan algoritmus alkalmas arra, hogy valamelyest csökkentsük a bizánci megegyezés kommunikációs költségének a bitszámát, de a költség továbbra is exponenciális függvénye lesz f-nek, a hibák számának. A bizánci hiba modellben nehezebb olyan algoritmust találni, mely polinomiális a hibák számára nézve, mint a megállási hibákat tartalmazó modellben. Ebben a részben mutatunk rá egy példát; ez az algoritmus a futási idő költségét tekintve nem lesz optimális, de nagyon egyszerű és néhány érdekes technikát alkalmaz. Az algoritmus speciálisan a $\{0,1\}$ értékű bizánci megegyezésre vonatkozik, a 6.3.3. szakasz eredményei alapján láthatjuk, hogyan lehet ezt az algoritmust az általános esetre alkalmazni.

Az algoritmus a *következetes üzenetszórás* működési módszert használja az összes kommunikációjához. Ez a működési módszer egy módja annak, hogy a különböző folyamatok által kapott üzenetek között bizonyos mennyiségű összefüggést biztosítsunk. A következetes üzenetszórást alkalmazva egy P_i folyamat *közreadhat* egy (m,i,r) alakú üzenetet az r-edik menetben, és ezt az üzenetet a folyamatok (beleértve P_i-t magát is) tetszőleges ezutáni menetben *elfogadhatják*. A következetes üzenetszórás működési módszer a következő három feltételnek tesz eleget.

1. Ha egy hibátlan folyamat az r-edik menetben közread egy (m,i,r) üzenetet, akkor a hibátlan folyamatok ezt az üzenetet az $(r+1)$-edik menetig elfogadják (azaz vagy az r-edik, vagy az $(r+1)$-edik menetben).
2. Ha a P_i hibátlan folyamat nem ad közre az r-edik menetben egy (m,i,r) üzenetet, akkor az (m,i,r) üzenetet soha egyetlen hibátlan folyamat sem fogadja el.
3. Ha egy hibátlan P_j folyamat elfogad egy (m,i,r) üzenetet, mondjuk az r'-edik menetben, akkor ezt az összes hibátlan folyamat elfogadja az $(r'+1)$-edik menetig.

Az első feltétel kimondja, hogy a hibátlan folyamatok közreadott üzeneteit gyorsan elfogadják, a második szerint hibátlan folyamatnak tévesen soha nem tulajdonítható üzenet. A harmadik feltétel szerint egy hibátlan folyamat által elfogadott üzenetet (akár hibás, akár nem hibás a küldő) nem sokkal később az összes többi hibátlan folyamat is elfogadja.

A következetes üzenetszórás módszere könnyen kivitelezhető.

A KövetkezetesÜzenSzórás algoritmus (vázlatosan)

Az (m,i,r) üzenet r-edik menetbeli közreadásához P_i küld egy („*kezd*”$,m,i,r$) üzenetet az összes folyamatnak az r-edik menetben. Ha P_j

kap egy (*„kezd”*, m, i, r) üzenetet P_i-től az r-edik menetben, akkor küld egy (*„echó”*, m, i, r) üzenetet az összes folyamatnak az $(r+1)$-edik menetben.

Ha tetszőleges $r' \geq r+2$ menet előtt P_j legalább $f+1$ folyamattól kapott (*„echó”*, m, i, r) üzenetet, akkor P_j (*„echó”*, m, i, r) üzenetet küld az r'-edik menetben (ha addig még nem tette meg).

Ha tetszőleges $r' \geq r+1$ menet végéig P_j legalább $n-f$ folyamattól kapott (*„echó”*, m, i, r) üzenetet, akkor P_j elfogadja a kommunikációt az r'-edik menetben (ha addig még nem tette meg).

6.24. tétel . *A* KövetkezetesÜzenSzórás *algoritmus megoldja a következetes üzenetszórás problémát, ha* $n > 3f$ *fennáll.*

Bizonyítás. Bebizonyítjuk, hogy a három tulajdonság fennáll.

1. Tegyük fel, hogy a hibátlan P_i folyamat közreadja az (m, i, r) üzenetet az r-edik menetben. Ekkor P_i az (*„kezd”*, m, i, r) üzenetet küldi az r-edik menetben, és az $\geq n-f$ számú hibátlan folyamat mindegyike az (*„echó”*, m, i, r) üzenetet küldi az $(r+1)$-edik menetben. Majd az $(r+1)$-edik menet végén a hibátlan folyamatok mindegyike (*„echó”*, m, i, r) üzenetet kap legalább $n-f$ folyamattól, és így elfogadja az üzenetet.
2. Ha a hibátlan P_i folyamat nem ad közre egy (m, i, r) üzenetet az r-edik menetben, akkor nem küld (*„kezd”*, m, i, r) üzeneteket, így a hibátlan folyamatok soha nem küldenek (*„echó”*, m, i, r) üzenetet. Ekkor a hibátlan folyamatok soha nem fogják elfogadni ezt az üzenetet, mivel az elfogadás feltétele, hogy az adott folyamathoz legalább $n-f > f$ számú folyamattól *echó* üzenet érkezzen.
3. Tegyük fel, hogy az (m, i, r) üzenetet a hibamentes P_j folyamat elfogadta az r'-edik menetben. Továbbá, hogy P_j (*„echó”*, m, i, r) üzenetet kap legalább $n-f$ folyamattól az r'-edik menetig. Az $n-f$ folyamat között legalább $n-2f \geq f+1$ a hibátlan folyamatok száma. Mivel a hibátlan folyamatok ugyanazt az üzenetet küldik az összes folyamatnak, így mindegyik hibamentes folyamat legalább $f+1$ számú (*„echó”*, m, i, r) üzenetet kap az r'-edik menetig. Ebből következik, hogy az $(r'+1)$-edik menetig mindegyik hibamentes folyamat küld egy (*„echó”*,m,i,r) üzenetet, tehát mindegyik folyamat legalább $n-f$ darab (*„echó”*,m,i,r) üzenetet kap az $(r'+1)$-edik menetig. Ezért az üzenetet valamennyi hibátlan folyamat elfogadja az $(r'+1)$-edik menetig.

□

Bonyolultságelemzés. Egyetlen üzenet következetes szórása $O(n^2)$ üzenetet igényel.

Most bemutatunk egy egyszerű bizánci megegyezés algoritmust, mely a következetes üzenetszórást használja a teljes kommunikációja során. A PoliBiz algoritmus csak az 1 kezdeti értékekről küld szét információt. Egy növekvő küszöbértéket használ az üzenetszóráshoz.

A PoliBiz algoritmus (vázlatosan)

Az algoritmus $f+1$ szintet használ, minden szint két menetből áll. Az elküldött üzenetek (következetes üzenetszórást használva) $(1, i, r)$ alakúak, ahol i egy folyamat indexe, r egy páratlan szám, a menet száma. Tehát az üzenetek elküldése csak az adott szint első menetében történik, és az elküldött információ értéke csak 1 lehet.

Annak feltételei, hogy a P_i folyamat közreadjon egy üzenetet, a következők. Az első menetben P_i akkor adja közre az $(1, i, 1)$ üzentet, ha P_i kezdeti értéke 1. A $(2s-1)$-edik menetben, mely az s-edik szint első menete, ahol $2 \leq s \leq f+1$, P_i pontosan akkor adja közre az $(1, i, 2s-1)$ üzenetet, ha P_i legalább $f+s-1$ számú, különböző folyamatoktól érkezett üzenetet elfogadott a $(2s-1)$-edik menet előtt, és eddig P_i még nem adott közre üzenetet.

A $2(f+1)$-edik menet végén P_i pontosan akkor hoz 1-es döntést, ha a $2(f+1)$-edik menet végéig legalább $2f+1$ különböző folyamat üzenetét elfogadta. Egyébként P_i döntése 0 lesz.

6.25. tétel . *A* PoliBiz *algoritmus megoldja a bizánci megegyezést, ha* $n > 3f$ *fennáll.*

Bizonyítás. A befejezés nyilvánvaló.

Az érvényességet tekintve két eset lehetséges. Az első, amikor az összes hibátlan folyamat az 1-es értékkel kezd, ekkor legalább $n - f \geq 2f + 1$ folyamat ad közre üzenetet az első menetben. A következetes üzenetszórás első tulajdonságából adódóan az összes hibamentes folyamat elfogadja ezeket az üzeneteket a második menetig, így a hibamentes folyamatok mindegyike legalább $2f+1$ különböző folyamattól fogad el üzenetet a második menet végéig. Ez elegendő ahhoz, hogy a hibátlan folyamatok mindegyike 1-est döntsön.

A másik eset, ha az összes folyamat a 0 kezdeti értékkel kezd, ekkor a hibamentes folyamatok egyetlen üzenetet sem adnak közre. Ennek az az oka, hogy legalább $f+1$ folyamatnak el kell fogadnia az üzenetet, hogy az közreadhatóvá váljon, aminek elérése lehetetlen anélkül, hogy előzetesen egy hibátlan folyamat közre ne adná. (A következetes üzenetszórás második tulajdonságát használjuk fel itt.) Ebből következik, hogy az összes hibamentes folyamat döntése 0 lesz.

Végül megvizsgáljuk a megegyezést. Legyen a hibátlan P_i folyamat döntése 1; elegendő megmutatni, hogy az összes többi hibamentes folyamat döntése is 1. Mivel P_i döntése 1, P_i-nek legalább $2f+1$ számú különböző folyamattól kell elfogadnia üzenetet a $2(f+1)$-edik menet végéig. Legyen I azoknak a folyamatoknak a halmaza, melyek ezek közül hibamentesek, ekkor $|I| \geq f+1$.

Ha mindegyik I-beli folyamat kezdeti értéke 1, akkor ezt mindannyian közreadják az első menetben, és a következetes üzenetszórás első tulajdonsága miatt valamennyi hibamentes folyamat elfogadja ezeket az üzeneteket a második menetig. Majd a harmadik menet[2] előtt, a hibamentes folyamatok mindegyike legalább $f+1$ különböző folyamattól érkező üzenetet fogadott el, mely elegendő ahhoz, hogy a harmadik menetre ez közreadhatóvá váljon, és ismét a következetes üzenetszórás első tulajdonsága miatt, az összes hibátlan folyamat elfogadja

[2]Feltesszük, hogy $f \geq 1$, ezért áll itt épp a harmadik menet.

ezt az üzenetet a negyedik menetig. Így mindegyik hibamentes folyamat legalább $n - f \geq 2f + 1$ különböző folyamat üzenetét fogadja el a negyedik menet végéig, emiatt a döntésük 1 lesz, amit bizonyítani akartunk.

Másik esetben tegyük fel, hogy az egyik I-bel folyamatnak, legyen az P_j, a kezdeti értéke nem 1. Ekkor P_j a $2s-1$ menet valamelyikében ad közre üzenetet, ahol $2 \leq s \leq f+1$, ami azt jelenti, hogy P_j legalább $f+s-1$ különböző folyamat üzenetét fogadja el a $(2s-1)$-edik menet előtt; mi több az üzenetek egyike sem származik önmagától. Ekkor a következetes üzenetszórás harmadik tulajdonsága miatt ezen $f + s - 1$ számú folyamat üzenetét az összes hibamentes folyamat elfogadja a $(2s-1)$-edik menet végéig, és így az első tulajdonságból következik, hogy a P_j által közreadott üzenetet az összes hibamentes folyamat elfogadja a $2s$-edik menet végéig. Ebből következik, hogy valamennyi hibamentes folyamat elfogadja legalább $(f + s - 1) + 1 = f + s$ különböző folyamat üzenetét a $2s$-edik menet végéig.

Ekkor két eset lehetséges. Ha $s = f + 1$, akkor mindegyik hibamentes folyamat legalább $2f + 1$ különböző folyamat üzenetét fogadja el a $2(f + 1)$-edik menet végéig, mely elegendő, hogy biztosítsa, hogy a folyamatok mindegyikének 1 legyen a döntése. A másik esetben $s \leq f$, akkor az összes hibamentes folyamat elegendően sok üzenetet fogad el a $(2s + 1)$-edik menet előtt ahhoz, hogy közreadjon a $(2s + 1)$-edik menetben, ha eddig még nem tette volna. Majd a következetes üzenetszórás első tulajdonsága szerint, az összes hibátlan folyamat elfogadja valamennyi hibátlan folyamat üzenetét a $(2s + 2)$-edik menet végéig. Ez ismét elegendő ahhoz, hogy valamennyien 1-est döntsenek, amire szükségünk volt. □

Bonyolultságelemzés. A PoliBiz algoritmus meneteinek száma $2f + 2$. Legfeljebb n a közreadások száma, melyek mindegyike $O(n^2)$ üzenet jelent; így az üzenetek száma $O(n^3)$. A bitek száma üzenetenként $O(\log n)$, mivel az üzenetek a folyamatok indexeit tartalmazzák. Így a kommunikációs bonyolultság bitszáma $O(n^3 \log n)$.

Kapcsolat a hitelesített bizánci hiba modellel. Ha a közönséges bizánci modellhez hozzáadjuk a következetes üzenetszórás képességét, egy olyan modellhez jutunk, ami valamiképp hasonló a 6.2.4. szakaszban vázlatosan ismertetett hitelesített bizánci hiba modellhez. Jóllehet a kettő nem pontosan ugyanaz. Például a következetes üzenetszórás csak üzenetszórásra szolgál és nem az egyedi üzenetküldésre. Még kifejezőbben, a következetes üzenetszórás nem védi meg P_i folyamatot attól, hogy közreadjon egy üzenetet, melyben (hibásan) azt állítja, hogy egy adott P_j folyamat egy bizonyos üzenetet küldött; a hibátlan folyamatok mindannyian elfogadják ezt az üzenetet, akkor is, ha a tartalma téves állítás. A hitelesített bizánci hiba modellben, a digitális aláírás lehetővé teszi, hogy a folyamatok az ilyen üzeneteket azonnal elutasítsák. Bár a modellek némileg különbözők, a következetes üzenetszórás elég erős ahhoz, hogy felhasználható legyen a közönséges bizánci modell néhány olyan algoritmusának megvalósításához, melyeket a hitelesített bizánci hiba modellhez terveztek.

6.4.. A bizánci megegyezés folyamatainak száma

Eddig olyan algoritmusokat mutattunk be, melyek megoldják a megegyezési problémát egy teljes hálózati gráfban, megállási, sőt bizánci hibák jelenlétében. Láthattuk, hogy ezek igen költséges algoritmusok. Megállási hibák esetére az általunk adott algoritmusok közül a legjobb a OptHalmazTerjed algoritmus, melynek költségei $f+1$ menet, $2n^2$ üzenet és $O(n^2b)$ bit a kommunikációra. A bizánci esetben az EIGYBiz algoritmus $f+1$ menetet használ, a kommunikációs költsége exponenciális nagyságrendű, míg a PoliBiz algoritmus $2(f+1)$ menetet használ és a kommunikációs költsége polinomiális nagyságrendű. Mindkét bizánci algoritmus megkívánja, hogy $n > 3f$ fennálljon.

A fejezet hátralevő részében megmutatjuk, hogy nem véletlenek ezek a magas költségek. Ebben a részben először bebizonyítjuk, hogy az $n > 3f$ megszorítás szükségszerű a bizánci megegyezési probléma tetszőleges megoldásában. A következő két alpont ezzel kapcsolatos eredményeket tartalmaz: a 6.5. alfejezet megadja az összefüggőség pontos mértékét arra, hogy tetszőleges nem teljes hálózati gráf esetében a bizánci megegyezés megoldható legyen, míg a 6.6. alfejezetben megmutatjuk, hogy az $n > 3f$ korlát kiterjed a bizánci megegyezésnél gyengébb állításokat tartalmazó problémára is. A fejezet utolsó részében belátjuk, hogy a menetek darabszámára megadott $f+1$ alsó korlát úgyszintén szükséges, még a megállási hibákat tartalmazó egyszerűbb esetben is.

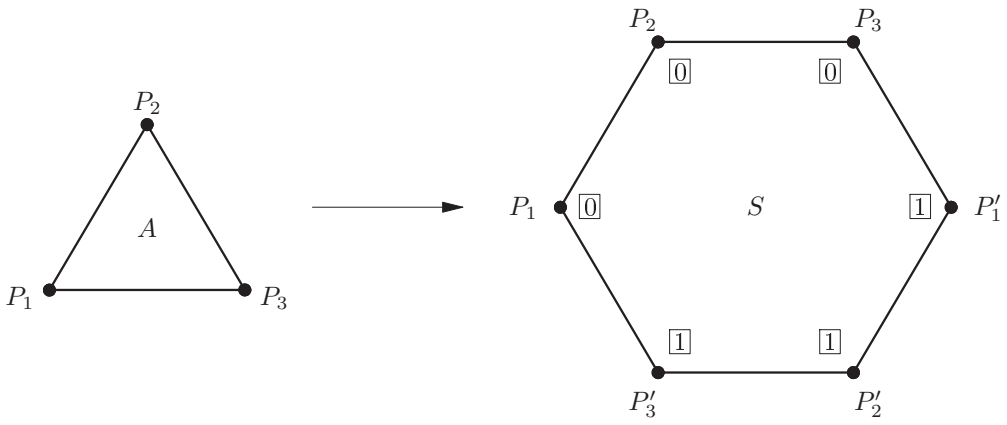
A bizonyítást, hogy f hiba jelenlétében $n \leq 3f$ folyamat nem képes megoldani a bizánci hibát, a legegyszerűbb speciális esettel kezdjük: megmutatjuk, hogy három folyamat nem tudja megoldani a bizánci megegyezést, ha fennáll egy hibának a lehetősége. Ezt az eredményt előrevetítette a 6.3.1. szakaszban leírt példa, bár a példa önmagában nem adott bizonyítást. Most egy általános eredményt adunk, olyan tetszőlegesen választott n, f értékekre, melyekre $n \leq 3f$ fennáll, úgy, hogy „redukáljuk" a problémát a három egy ellen típusú felállásra.

6.26. lemma . *Három folyamat nem képes a bizánci megegyezés megoldására egy hiba jelentkezése esetében.*

Bizonyítás. Indirekten: tegyük fel, hogy az A algoritmus megoldja a bizánci megegyezési problémát a P_1, P_2, P_3 folyamatokra, akkor is, ha a három közül az egyik hibázhat. Alkossunk egy új S rendszert az A két példányából, és megmutatjuk, hogy S ellentmondásosan viselkedik. Ebből az következik, hogy a feltételezett A algoritmus nem létezik.

Egész pontosan vegyük az A-beli folyamatok két-két példányát, és rendezzük őket egy hatszög alakú S rendszerbe. A folyamatok egyik példányát (a másolt példányt) a 0 bemeneti értékkel indítjuk, a másik példányt (az eredetit) az 1 bemeneti értékkel. Az elrendezés a 6.7. ábrán látható.

Mit mondhatunk az S rendszerről formálisan? Ez egy szinkron rendszer egy hatszög alakú hálózati gráfban, a 2. fejezet általános modelljéhez tartozik. Jegyezzük meg, *nem* állítjuk azt, hogy ez a rendszer megoldja a bizánci megegyezési problémát – nem az a fontos számunkra, hogy mit tud, a lényeg, hogy ez egyfajta szinkron rendszer. Nem fogjuk vizsgálni az S-beli folyamatok működési hibáit.



6.7.. ábra. Az A két példányából összeállított S rendszer.

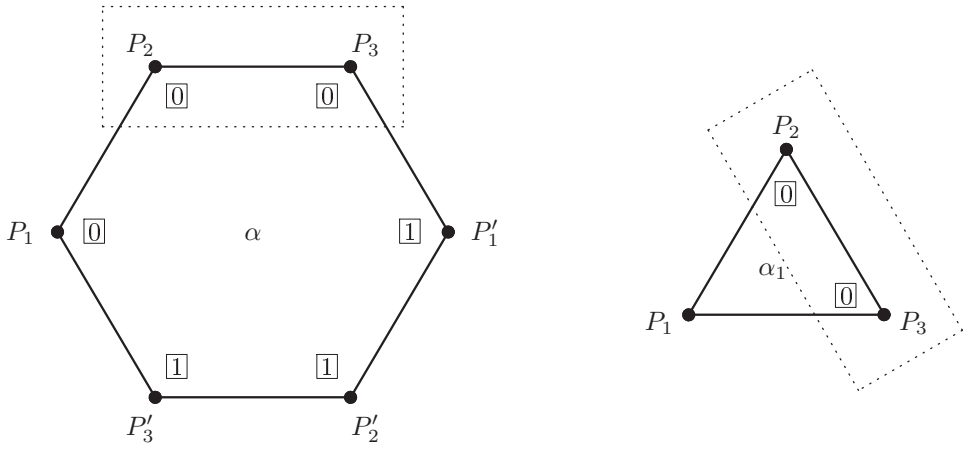
Emlékezzünk rá, hogy azokban a rendszerekben, melyeket a bizánci megegyezési probléma megoldásának tekintettünk, feltettük, hogy mindegyik folyamat ismeri a teljes hálózati gráfot. Például A-ban P_1 ismeri a P_2 és P_3 neveket, és feltételezi, hogy pontosan három csúcspont van, P_1, P_2 és P_3, melyek háromszöget alkotnak. S-ben nem azt feltételezzük, hogy a folyamatok az egész hatszögű gráfot ismerik, hanem csak azt, hogy a folyamatok a szomszédaik egy helyi nevével rendelkeznek. Például S-ben a P_1 azt tudja, hogy két szomszédja van, akiknek P_2 és P_3 a nevük, bár valójában az egyikük P_3'. P_1 nem tudja, hogy a csúcsok két példányban vannak a hálózatban. A helyzet hasonló a 4. fejezetbelihez, ahol a folyamatoknak csak helyi ismereteik voltak a hálózati gráf saját maguk körüli részéről. Azt mondhatjuk, hogy az S-beli hálózat a folyamatok számára gyakorlatilag úgy néz ki, mintha A-beli lenne.

Nem várunk el az S rendszertől semmiféle különleges viselkedést. Azt viszont igen, hogy S bármely bemeneti érték hozzárendelésre *egy adott*, jól definiált viselkedést mutasson. Úgy fogunk ellentmondásra jutni, hogy megmutatjuk, nem lehetséges ilyen jól definiált viselkedése S-nek a fentebb említett bemeneti értékek esetére.

Tegyük fel tehát, hogy S-ben a folyamatok a 6.7. ábrán látható kezdeti értékkel indulnak, a folyamatok másolt példányai a 0-val, az eredeti példányok az 1-gyel; legyen α az eredményül kapott végrehajtási sorozat S-ben.

Elsőként a P_2 és P_3 folyamatok szemszögéből vizsgáljuk az α végrehajtási sorozatot. A P_2 és P_3 folyamatoknak úgy tűnik, hogy egy A, háromszög elrendezésű rendszerben futnak, egy olyan végrehajtási sorozatban, legyen ez α_1, melyben a P_1 folyamat hibás. Az α és az α_1 megkülönböztethetetlen a P_2 és P_3 folyamatok számára, $\alpha \overset{2}{\sim} \alpha_1$ és $\alpha \overset{3}{\sim} \alpha_1$ a „megkülönböztethetetlenség” 2.4. alfejezetben megadott definíciója szerint. Lásd a 6.8. ábrát. Az α_1-ben P_1 egy különös fajta hibás viselkedést mutat – úgy viselkedik, mintha az α-beli P_1', P_2', P_3' és P_1 kombinációja lenne. Bár ez a viselkedés különleges, de megengedhető egy A-beli hibás

folyamatnak, bizánci hibákat feltételezve.



6.8.. ábra. A P_2 és P_3 folyamatok számára az α és α_1 megkülönböztethetetlen végrehajtások.

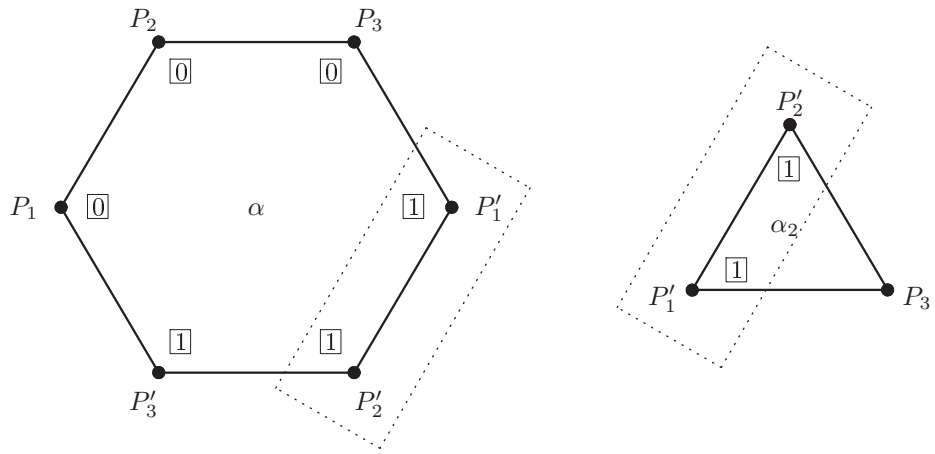
Tudjuk, hogy α_1 egy olyan A-beli végrehajtási sorozat, amelyben a P_1 hibás, P_2 és P_3 a 0 kezdeti értékkel kezd, és mivel feltevésünk szerint A megoldja a bizánci megegyezést, a bizánci megegyezés helyességi követelményeiből adódik, hogy végül α_1-ben P_2-nek és P_3-nak 0-ás döntést kell hoznia. Minthogy α megkülönböztethetetlen α_1-től P_2 és P_3 számára, mind kettőjük döntése az α-ban is 0 lesz.

Most tekintsük az α végrehajtási sorozatot a P_1' és P_2' folyamatok szemszögéből. A P_1' és P_2' folyamatoknak úgy tűnik, hogy egy A, háromszög elrendezésű rendszerben futnak, egy olyan végrehajtási sorozatban, legyen ez α_2, melyben P_3 hibás. Azaz $\alpha \overset{1'}{\sim} \alpha_2$ és $\alpha \overset{2'}{\sim} \alpha_2$. Lásd a 6.9. ábrát. A fentivel azonos érvelésből adódóan P_1' és P_2' végül 1-es döntést hoz α-ban.

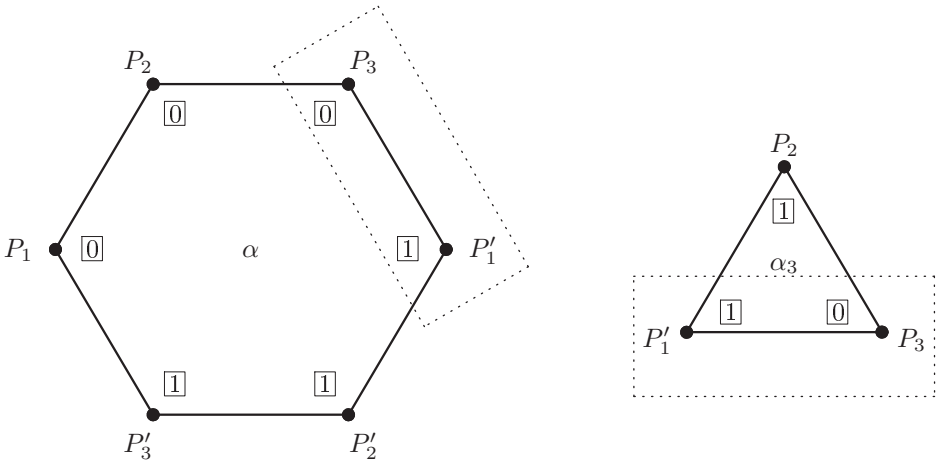
Végül tekintsük az α végrehajtási sorozatot a P_3 és P_1' folyamatok szemszögéből. A P_3 és P_1' folyamatoknak úgy tűnik, hogy egy A, háromszög elrendezésű rendszerben futnak, egy olyan végrehajtási sorozatban, legyen ez α_3, melyben P_2 folyamat hibás. Azaz $\alpha \overset{3}{\sim} \alpha_3$ és $\alpha \overset{1'}{\sim} \alpha_3$. Lásd a 6.10. ábrát. A bizánci megegyezés helyességi feltételeiből adódik, hogy P_3 és P_1' folyamatoknak végül döntést kell hozniuk α_3-ban, és ugyanazt kell dönteniük. Mivel P_3 a 0 kezdeti értékkel indul és P_1 az 1 kezdeti értékkel, nincs elvárás, hogy melyik érték mellett döntenek, de a megegyezési feltétel szerint megegyeznek. Ezért α-ban is ugyanazt döntik.

Ez viszont ellentmondás, mivel azt már tudjuk, hogy α-ban P_3 folyamat a 0 döntést hozza, P_1' pedig az 1-et. □

Most a 6.26. lemmát használva bebizonyítjuk, hogy a bizánci megegyezés megoldása lehetetlen $n \leq 3f$ folyamattal. Ehhez megmutatjuk, hogy ha létezne $n \leq 3f$ folyamattal megoldás, mely f számú hibát eltűr, akkor abból következik, hogy létezik megoldás három folyamatra és egy bizánci hibára, ami ellentmond a



6.9.. ábra. Az P_1' és P_2' folyamatok számára az α és α_2 megkülönböztethetetlen végrehajtási sorozatok.



6.10.. ábra. A P_3 és P_1' folyamatok számára az α és α_3 megkülönböztethetetlen végrehajtási sorozatok.

6.26. lemmának.

6.27. tétel . *A bizánci megegyezés problémának nincs megoldása n folyamatra f bizánci hiba előfordulása esetében, ha $2 \leq n \leq 3f$.*

Bizonyítás. Arra a speciális esetre, amikor $n = 2$, egyszerűen belátható, hogy nincs megoldás. Vázlatosan elmondva, tegyük fel, hogy az egyik folyamat 0-val kezd, a másik 1-gyel. Ekkor mind egyikőjük –annak lehetőségét megengedve,

hogy a másik hibás– saját értéke szerint dönt, hogy kielégítse az érvényesség feltételét. Azonban, ha egyikőjük sem hibás, ez megsérti a megegyezési feltételt. Így feltehetjük, hogy $n \geq 3$.

Tegyük fel, indirekten, hogy A egy megoldása a bizánci megegyezésnek $3 \leq n \leq 3f$ mellett. Megmutatjuk, hogyan transzformálható A egy olyan B megoldásba, mely három, P_1, P_2, P_3 folyamattal egy hibát eltűrve megoldja a bizánci megegyezést. Mindegyik B-beli folyamat nagyjából az A-beli folyamatok egy harmadát fogja szimulálni.

Az A-beli folyamatokat három nemüres alhálózatra bontjuk, legyenek ezek I_1, I_2 és I_3, mindegyikük legfeljebb f méretű. A B-beli P_i folyamat a következő módon fogja szimulálni I_i-t.

> **B:**
> A P_i folyamatok mindegyike nyomon követi az összes I_i-beli folyamat állapotát, I_i minden eleméhez hozzárendeli saját kezdeti értékét, és szimulálja az I_i-beli folyamatok lépéseit, csakúgy mint az I_i-beli párok közti üzeneteket. Azokat az üzeneteket, melyek I_i-beli folyamatok egy másik alhálózathoz küldenek, P_i elküldi ahhoz a folyamathoz, mely a másik alhálózatot szimulálja. Ha tetszőleges számú folyamat v döntést hoz I_i-ben, akkor P_i döntése is v lesz. (Amennyiben egynél több ilyen érték lenne, P_i tetszőlegesen választ közülük.)

Bebizonyítjuk, hogy B helyesen oldja meg a bizánci megegyezést három folyamatra. Válasszuk A-ban pontosan azokat a folyamatokat hibásnak, amelyeket a B-beli hibásan viselkedő folyamat szimulál[3]. Rögzítsük B egy tetszőleges α végrehajtási sorozatát, melyben legfeljebb egy folyamathiba lép fel, és legyen α' az az A-beli végrehajtás, melyet ez szimulál. Mivel mindegyik B-beli folyamat legalább f számú A-beli folyamatot szimulál, legfeljebb f hibás folyamat van α'-ben. Továbbá feltettük, hogy A megoldja a bizánci megegyezést n folyamat és legfeljebb f hiba esetében, így a szokásos megegyezési, érvényességi és befejezési feltételek a bizánci megegyezésre fennállnak α'-ben.

Bebizonyítjuk, hogy ezek a feltételek α-ra is átvihetők. A befejezés igazolásához legyen P_i egy B-beli hibátlan folyamat. Ekkor P_i legalább egy A-beli P_j folyamatot szimulál, és P_j biztosan hibátlan, mivel P_i is az. Az α'-re fennálló befejezési feltételből következik, hogy P_j-nek végül döntenie kell; és amint ezt megtette, P_i is dönt (ha eddig még nem tett volna így).

Az érvényesség belátásához tudjuk, hogy amennyiben a B-beli összes folyamat egy adott v értékkel kezd, akkor az összes hibamentes A-beli folyamat is v értékkel kezd. Az α' érvényességéből következik, hogy az α'-beli hibátlan folyamatok döntése csak v lehet. Ekkor csak v lehet a döntése az α-beli hibátlan folyamatoknak is.

A megegyezés fennállásához tegyük fel, hogy P_i és P_j hibátlan folyamatok B-ben. Ekkor ezek csak A-beli hibátlan folyamatokat szimulálnak. Az α'-re fennálló megegyezésből adódik, hogy ezek a szimulált folyamatok mind megegyeznek, tehát P_i és P_j is megegyezik.

[3]Felhasználjuk azt a technikai sajátosságot, hogy a bizánci hibás folyamatok teljesen helyesen is viselkedhetnek, hogy ez az osztályozás fenntartható legyen.

Összefoglalva azt kapjuk, hogy B megoldja a bizánci megegyezési problémát három folyamatra egy hiba eltűrése mellett. Ez ellentmond a 6.26. lemmának. □

6.5.. Bizánci megegyezés általános gráfokban

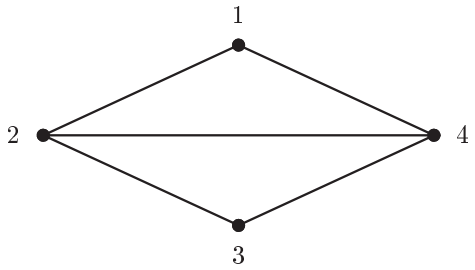
A fejezetben eddig a megegyezési problémákat csak teljes gráfokban vizsgáltuk. Az n csúcsú teljes gráfokra a 6.3. és 6.4. alfejezetekben beláttuk, hogy a bizánci megegyezés csak akkor oldható meg, ha $n > 3f$ fennáll. Ebben a részben általános hálózati gráfban vizsgáljuk a bizánci megegyezési problémát. Pontosan jellemezzük azokat a gráfokat, amelyekben a probléma megoldható.

Vegyük először azt, amikor a hálózat gráfja egy legalább 3 csúccsal rendelkező fa, ekkor nincs esélyünk a bizánci megegyezési probléma megoldására, ha akár egy folyamat is hibásan viselkedik, hisz bármely hibás folyamat, ami nem levél, lényegében „szétválaszthatja" a fa egyik részében lévő folyamatokat a másik rész folyamataitól. A különböző részben elhelyezkedő hibátlan folyamatok képtelenek a valós kommunikációra, még kevésbé a megegyezésre. Hasonlóan kézenfekvőnek látszik, hogy ha f számú csúcs elegendő a gráf szétválasztására, akkor a bizánci megegyezés nem oldható meg f hibás folyamatra.

A megérzés formalizáláshoz a következő gráfelméleti fogalmat használjuk. Egy G gráf *összefüggőségének* mértéke, jelölje *össz_függ*(G), definíció szerint azon csúcsok számának minimuma, melyeket eltávolítva a gráfból, egy nem összefüggő gráfot, vagy a triviális egy csúcsból álló gráfot kapjuk. A G gráf *c-összefüggő*, ha *össz_függ*$(G) \geq c$.

6.5.1. példa. Összefüggőség

Ha egy fának legalább két csúcsa van, akkor az összefüggősége 1. Az n csúcsú teljes gráf összefüggősége $n - 1$. A 6.11. ábrán egy 2 összefüggőségű gráfot látunk. Ha a 2-es és 4-es csúcsot eltávolítjuk, akkor az eredmény az egymástól elválasztott 1-es és 3-as csúcs.



6.11.. ábra. Egy G gráf, melyre *össz_függ*$(G) = 2$.

A gráfelmélet egy klasszikus tételét fogjuk használni, melyet *Menger tételének* neveznek.

6.28. tétel . (Menger tétele.) *A G gráf akkor és csak akkor c-összefüggő, ha G bármely két csúcsa legalább c csúcs-független úttal van összekötve.*

Most már jellemezhetjük azokat a gráfokat, melyekben a bizánci megegyezés egy adott számú hiba mellett megoldható. A jellemzéshez a gráf csúcsainak számát és az összefüggőség mértékét vizsgáljuk. A jellemzés megoldhatatlansági részének bizonyításához hasonló módszereket használunk, mint amit a 6.4. alfejezetben használtunk a hibás folyamatok számára adott alsó korlátnál.

6.29. tétel . *A bizánci megegyezési probléma egy n csúcsú G hálózati gráfban f hiba eltűrése mellett akkor és csak akkor oldható meg, ha mindkét alábbi feltétel teljesül:*

1. $n > 3f$;
2. $össz\_függ(G) > 2f$.

Bizonyítás. A 6.27. tételben már beláttuk, hogy $n > 3f$ folyamat szükséges, hogy a bizánci megegyezés megoldható legyen egy teljes gráfban. Könnyen beláthatjuk, hogy az $n > 3f$ feltétel fennállása tetszőleges (nem feltétlenül teljes) gráfok esetében is szükséges; hiszen ha egy algoritmus $n \leq 3f$ mellett nem teljes gráfban megoldaná a problémát, akkor azt használhatnánk egy n-csúcsú teljes gráfban is.

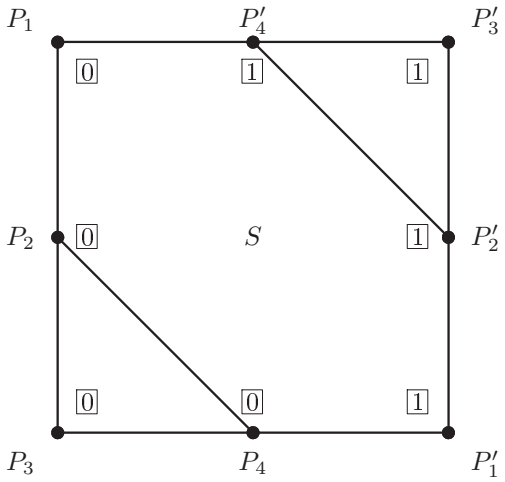
Most megmutatjuk, hogy igaz az állítás *ha* iránya, nevezetesen, a bizánci megegyezés lehetséges, ha $n > 3f$ és $össz\_függ(G) > 2f$. Mivel G gráf $2f+1$-összefüggő, Menger tételéből, a 6.28. tételből következik, hogy G bármely két csúcsa között legalább $2f+1$ csúcs-független út van. Bármely két hibamenetes P_i és P_j között megvalósítható megbízható kommunikáció úgy, hogy P_i üzenetet küld P_j-nek a köztük lévő $2f+1$ számú útvonalon. Mivel a hibás folyamatok száma legfeljebb f, a P_j folyamathoz a különböző útvonalakon érkező üzenetek többsége helyes lesz.

Ha a hibamentes folyamatok közül bármely kettő között megbízható a kommunikáció, akkor a bizánci megegyezés megoldható tetszőleges olyan algoritmus szimulációjával, mely egy n-csúcsú teljes gráfban megoldja a problémát. A megbízható kommunikáció fentebb megadott megvalósítását használjuk a teljes gráfbeli csúcstól csúcsig történő adatátvitel helyett. Természetesen a bonyolultság megnövekszik, de most ez nem lényeges, – az algoritmus helyesen működik.

Térjünk rá a bizonyítás érdekesebb részére, belátjuk, hogy a bizánci megegyezés csak akkor oldható meg, ha $össz\_függ(G) > 2f$. Leegyszerűsítve a problémát, azt az esetet vizsgáljuk, amikor $f = 1$; a nagyobb f értékekre történő, hasonlóan elvégezhető bizonyítást a 6-41. gyakorlatra hagyjuk.

Legyen G egy olyan gráf, melyre $össz\_függ(G) \leq 2$, és amelyben az A algoritmussal megoldható a bizánci megegyezés egy hiba esetében. Ekkor két olyan csúcsa van G-nek, melyek vagy szétválasztják G csúcsait, vagy egy egy-csúcsú gráfra redukálják. Ha egy-csúcsú gráfra redukálják, akkor G csak három csúcsból áll, és azt már tudjuk, hogy egy három csúcsból álló gráfban egy hiba esetében nem oldható meg a bizánci megegyezés. Így feltehetjük, hogy a két csúcs szétválasztja G csúcsait.

Ekkor a 6.11. ábrához hasonló a kép, kivéve hogy az 1-es és 3-as csúcs helyettesíthető tetszőleges összefüggő részgráfokkal, és a 2-es és 4-es csúcs össze lehet kötve az összefüggő részgráfok mindegyik csúcsával. (Hiányozhat a 2-es és 4-es csúcsot összekötő él is, de ez csak rontaná a helyzetet.) Ismét az egyszerűség kedvéért azt az esetet vizsgáljuk, amikor az 1-es és 3-as csak egyszerű csúcsok. Egy S rendszert építünk A két példányának összerakásával. A folyamatok egyik példányát a 0 bemenő értékkel indítjuk, a másikat 1-gyel, ahogy a 6.12. ábrán látjuk. A 6.26. lemma bizonyításához hasonlóan, S az adott bemenetre jól definiált módon viselkedik. Ismét úgy jutunk ellentmondásra, hogy belátjuk, nem lehetséges ez a viselkedés.



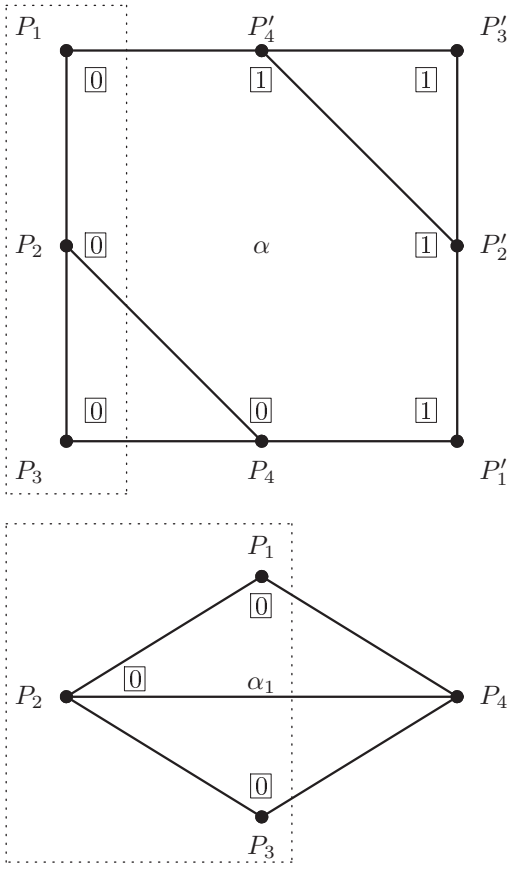
6.12.. ábra. Az A két példányának kombinációjaként kapjuk meg az S algoritmust.

Tegyük fel tehát, hogy S-ben a folyamatok a 6.12. ábrán látható kezdeti értékekkel indulnak, azaz a másolat folyamatok 0-val, az eredetiek 1-gyel; legyen α az eredményül kapott végrehajtás.

Tekintsük az α végrehajtási sorozatot a P_1, P_2 és P_3 folyamatok szemszögéből. A folyamatoknak úgy tűnik, mintha az A rendszerben futnának egy α_1 végrehajtási sorozatban, amelyben P_4 hibás. Lásd a 6.13. ábrát. Ekkor a bizánci megegyezés helyességének feltételeiből adódik, hogy végül α_1-ben P_1, P_2 és P_3 döntése 0 lesz. Minthogy α megkülönböztethetetlen α_1-től a P_1, P_2 és P_3 folyamatok számára, mind hármójuk döntése az α-ban is 0 lesz.

Majd tekintsük az α végrehajtást a P'_1, P'_2 és P'_3 folyamatok szemszögéből. E három folyamatnak úgy tűnik, mintha az A rendszerben futnának egy α_2 végrehajtási sorozatban, amelyben P_4 hibás. Lásd a 6.14. ábrát. Az előzőhöz hasonló bizonyítással azt kapjuk, hogy P'_1, P'_2 és P'_3 folyamatok döntése 1 lesz az α-ban.

Végül tekintsük az α végrehajtási sorozatot a P_3, P_4 és P'_1 folyamatok szemszögéből. Ezeknek a folyamatoknak úgy tűnik, mintha az A rendszerben futnának egy α_3 végrehajtási sorozatban, amelyben P_2 hibás. Lásd a 6.15. ábrát. A bizánci

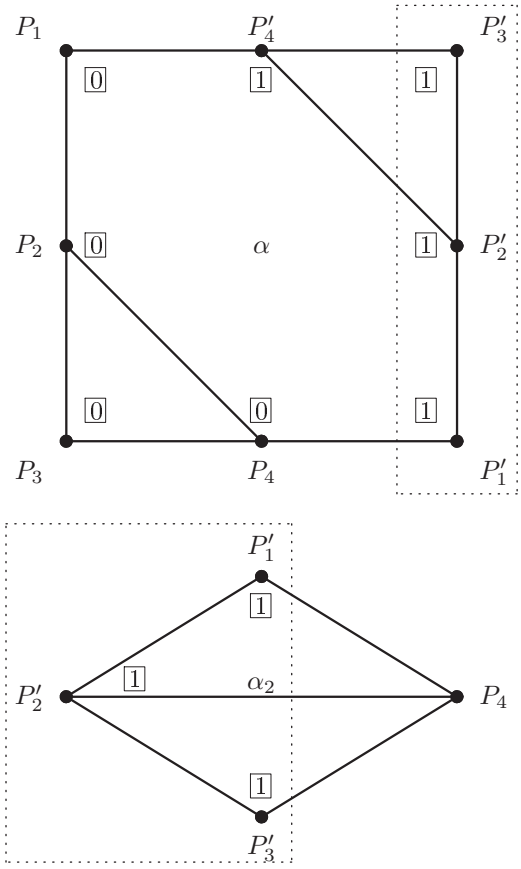


6.13.. ábra. Az α és α_1 megkülönböztethetetlen P_1, P_2 és P_3 folyamatok számára.

megegyezés helyességének feltételeiből adódik, hogy végül e három folyamat döntést hoz az α_3-ban, és döntésük azonos lesz. Ugyanez igaz α-ban.

De ez ellentmondás, mivel már beláttuk, hogy P_3 döntése csak 0 lehet α-ban, P'_1 döntése pedig csak 1. Ebből következik, hogy nem lehet megoldani a bizánci megegyezést a G gráfban $\mathit{össz\_függ}(G) \leq 2$ és $f = 1$ esetében.

Az eredmények $f > 1$ esetre történő általánosításához ugyanezeket a diagramokat használhatjuk úgy, hogy a P_2 és P_4 csúcsokat az I_2 és I_4, legfeljebb f csúcsból álló halmazokkal helyettesítjük, a P_1 és P_3 csúcsokat pedig tetszőleges I_1 és I_3, csúcsokat tartalmazó halmazokkal. Az összes I_2 és I_4-beli csúcsot eltávolítva, I_1 és I_3 elszakad egymástól. A 6.11. ábra éleit az I_1, I_2, I_3 és I_4-beli csúcsokból álló csoportok közötti élek kötegeinek tekinthetjük. □



6.14.. ábra. Az α és α_2 megkülönböztethetetlen P_1', P_2' és P_3' folyamatok számára.

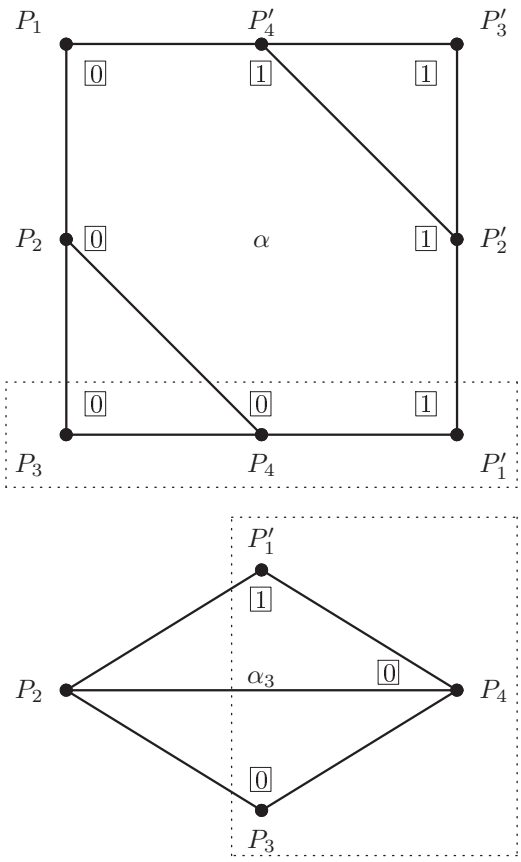
6.6.. Gyenge bizánci megegyezés

Ugyanaz az általános bizonyítási séma, amit a 6.4. és 6.5. alfejezetekben használtunk, hogy bebizonyítsuk a megoldhatatlansági eredményeket a bizánci megegyezésre $n \leq 3f$ vagy *össz_függ* $\leq 2f$ esetében, alkalmazható egyéb megegyezési problémák megoldhatatlansági eredményeinek bizonyításához. Például, ebben az alfejezetben megmutatjuk, hogyan alkalmazható ez a módszer a bizánci megegyezés egy gyengített változatára, melyet *gyenge bizánci megegyezésnek* hívnak.

Az egyetlen különbség a gyenge bizánci megegyezés és az általános bizánci megegyezés között az érvényességi feltételben van. A gyenge bizánci megegyezés érvényessége a következő.

Érvényesség. Ha nincs hibás folyamat és mindegyik folyamat ugyanazzal a $v \in V$ kezdeti értékkel indul, akkor csak v a lehetséges döntési érték.

Az általános bizánci modellben, ha az összes hibátlan folyamat ugyanazzal a



6.15.. ábra. Az α és α_3 megkülönböztethetetlen P_3, P_4 és P'_1 folyamatok számára.

v kezdeti értékkel indul, akkor mindannyiukra nézve kötelező a v döntés, *akkor is, ha vannak hibás folyamatok*. A gyenge bizánci megegyezésben csak akkor várjuk el tőlük a v döntést, ha nincsenek hibák.

Mivel a probléma kikötése gyengébb, mint az általános esetben, ezért az algoritmus, melyet az általános bizánci problémára adtunk, működik a gyenge bizánci megegyezésre is. Másrészről, a megoldhatatlansági eredmények nem vihetők át közvetlenül; elképzelhető, hogy a gyenge bizánci megegyezésre léteznek hatékonyabb algoritmusok. Jóllehet, ki fog derülni (egy apró technikai feltétel mellett), hogy a folyamatok darabszámára és a gráf összefüggésére fennálló korlát továbbra is igaz. (A technikai feltétel, amire szükségünk van: feltesszük, hogy $n \geq 3$, mivel a gyenge bizánci megegyezés $n = 2$ esetére van egy triviális algoritmus.)

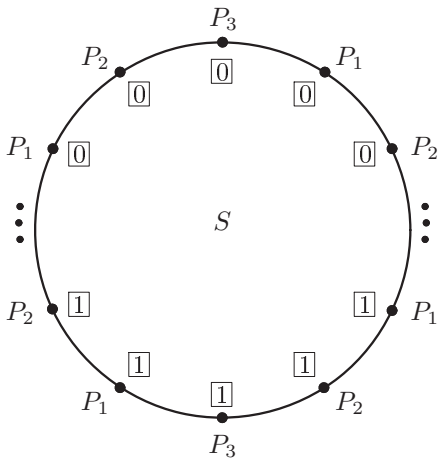
6.30. tétel . *Legyen $n \geq 3$. A gyenge bizánci megegyezési probléma akkor és csak akkor oldható meg egy n-csúcsból álló G hálózati gráfban f hiba eltűrése mellett, ha mindkét alábbi állítás teljesül:*

1. $n > 3f$;
2. $össz\_függ(G) > 2f$.

Bizonyítás. A *ha* irány következik azon protokollok létezéséből, melyet a 6.29. tételben az általános bizánci megegyezésre adtunk. Bebizonyítjuk, hogy három folyamat nem képes a gyenge bizánci megegyezés megoldására egy hibalehetőség mellett, és a 6-43. gyakorlatra hagyjuk az állítás $f > 1$ esetre történő kiterjesztésének, valamit az összefüggőségre vonatkozó állításnak a bizonyítását. Az egyszerűség kedvéért tegyük fel, hogy $V = \{0, 1\}$.

Legyen A egy három-folyamatos algoritmus, mely megoldja a gyenge bizánci megegyezést a P_1, P_2 és P_3 folyamatokra, még ha az egyik hibás is. Legyen α_0 A-nak az a végrehajtási sorozata, melyben mindhárom folyamat a 0 értékkel indul és hiba nem lép fel. A befejezési és érvényességi szabályokból következik, hogy végül mindhárom folyamat 0 döntést hoz az α_0-ban; legyen r_0 azon menetek sorszámai közül a legkisebb, melyre mindegyik folyamat döntésre jut. Hasonlóképpen legyen α_1 az a végrehajtás, melyben mindegyik folyamat 1-gyel kezd és hiba nem lép fel, így végül mindegyik folyamat döntése 1 lesz α_1-ben. Legyen r_1 az ehhez szükséges menetek száma és válasszuk r-nek az $r \geq \max\{r_0, r_1, 1\}$ értéket.

Készítsünk egy új S rendszert, az A rendszer $2r$ számú példányának egy kör mentén való elhelyezésével, azaz $6r$ folyamattal, melyből $3r$ a kör „felső felén" helyezkedik el, $3r$ pedig az „alsó felén". A felső félkörön lévőket 0 kezdeti értékkel indítjuk, az alsó félkörön lévőket 1-gyel. Az elrendezést a 6.16. ábrán láthatjuk. (Most nem bonyolítjuk az ábrát az eredeti és többi, több példányban szereplő A-beli folyamat megkülönböztető jelölésével.) Legyen α az eredményül kapott S-beli végrehajtás.



6.16.. ábra. Az A $2r$ darab példányának együtteseként megkapjuk az S rendszert.

A 6.26. lemma bizonyításához hasonlóan megmutathatjuk, hogy bármely két szomszédos S-beli folyamatnak ugyanarra a döntésre kell jutnia az α végrehajtási

sorozatban, mivel a két folyamat szempontjából úgy tekinthetjük, hogy egy háromszögben vannak elhelyezve, egy harmadik, hibás folyamattal összekapcsolva. Ebből az következik, hogy S mindegyik folyamata *ugyanazt* a döntést hozza az α végrehajtási sorozatban. Tegyük fel, az általánosság megsértése nélkül, hogy valamennyiük döntése 1.

Most, hogy ellentmondásra jussunk, belátjuk, hogy néhány, a felső félkörön elhelyezkedő folyamatnak 0 döntést kell hoznia. Legyen B tetszőleges, az S felső félkörén egymás után következő, $2r + 1$ darabszámú folyamatból álló „blokk"; ezek mindegyike a 0 kezdeti értékkel indul α-ban. Ekkor az összes B-beli folyamat ugyanabból az állapotból indul α-ban, mint az azonos nevű folyamat α_0-ban, és ugyanazokat az üzeneteket küldi az első menetben. Így, az első menetben az összes B-beli folyamat, *kivéve esetleg a blokk két szélén lévő egy-egy folyamatot*, ugyanazt az üzenetet kapja α-ban, mint a névrokonaik α_0-ban, így azonos állapotban maradnak és azonos üzenetet küldenek a második menetben mindkét végrehajtási sorozatban. A második menetben mindegyik B-beli folyamat, kivéve kettőt-kettőt a blokk két szélén, ugyanazt az üzenetet kapja, és ugyanabban az állapotban marad mindkét végrehajtási sorozatban. Ezt folytatva, a k-adik, $1 \leq k \leq r$ menetben az összes B-beli folyamat, kivéve k darabot a két szélen, ugyanazt az üzenetet kapja, és ugyanabban az állapotban marad α-ban és α_0-ban. Más szavakkal az α és α_0 k menetet vizsgálva megkülönböztethetetlen valamennyi B-beli folyamat számára, kivéve a két szélen k darabot. Vázlatosan szólva, ennek az a magyarázata, hogy az információnak nincs ideje arra, hogy a blokk két széléről eljusson ezekhez a folyamatokhoz.

Gyakorlatilag az α és α_0 megkülönböztethetetlen a B blokk közepén elhelyezkedő P_i számára az r-edik menetben. Minthogy P_i folyamat 0 döntést hoz az r-edik menet végére az α-ban, így tesz az α_0-ban is. Ez ellentmond a kiinduláskor tett feltevésünknek, hogy P_i 1 döntést hoz α-ban. □

6.7.. A menetek száma megállási hibák esetében

A fejezet befejezéseként megmutatjuk, hogy a megegyezési probléma nem oldható meg kevesebb, mint $f+1$ menet alatt, akár bizánci, akár megállási hibák esetében. Más szavakkal nem létezik olyan megegyezési protokoll egyik hibafajta esetére sem, melyben az összes hibamentes folyamat az f-edik menetig döntést hoz.

Módszerünk az lesz, hogy feltesszük, hogy létezik f-menetes megegyezési algoritmus, és a bizonyítás során ellentmondásra jutunk. Kényelmi okokból néhány megszorítást teszünk a feltételezett algoritmusra vonatkozóan, de egyik sem okozza az általánosság megcsorbítását. Elsőként feltesszük, hogy a hálózati gráf teljesen összefüggő; egy gyors, nem teljes gráfon futó algoritmus teljes gráfon is működik, így biztosan nem veszítünk az általánosságból ezzel a megszorítással. Feltesszük továbbá, hogy az összes olyan folyamat, amelyik döntést hoz, az ezt pontosan az f-edik menet végén teszi, majd rögtön megáll. Ebben az esetben a bizánci megegyezést megoldó algoritmus szükségképp a megállási problémára is jó (lásd a 6.1. alfejezet megjegyzését a két probléma közötti kapcsolatról). Így célunk, a megoldhatatlansági eredmény eléréséhez elegendő, ha csak a megállási

hiba probléma vizsgálatára szorítkozunk. Azt is feltesszük, hogy mindegyik folyamat mindegyik másik folyamatnak küld üzenetet valamennyi k, $1 \leq k \leq f$ menetben (hacsak, és amíg nem hibázik). Végül, vizsgálatunkat leszűkítjük a $V = \{0, 1\}$ értékhalmazra.

Az 5. fejezetben tárgyalt összehangolt támadási problémához hasonlóan, a bizonyítás kényelmessé válik a kommunikációs minta fogalmának használatával, mely annak jelölése, hogy az egyes menetekben melyik folyamat mely más folyamatoknak küld üzenetet. Az előző definíciót alkalmazva a teljes gráf esetére a kommunikációs mintát úgy definiáljuk, hogy bármely részhalmaza a következő halmaznak:

$$\{(i, j, k) : 1 \leq i, j \leq n, i \neq j, 1 \leq k\}.$$

A kommunikációs minta nem ábrázolja az üzenetek tartalmát, csak azt, hogy melyik folyamat mely folyamatoknak, melyik menetben küld üzenetet.

A kommunikációs mintára vonatkozólag három megszorítást teszünk. Először, mivel a vizsgált algoritmus f menetes, csak azokat a kommunikációs mintákat tekintjük, melyekben mindegyik (i, j, k) hármasban $k \leq f$ fennáll. Másodszor, minthogy a megállási hiba modellel dolgozunk, a lehetséges kommunikációs minták eleget tesznek a következő megkötésnek: ha egy tetszőleges (i, j, k) hármas hiányzik a mintából, akkor ez igaz minden olyan (i, j', k') hármasra, ahol $k' > k$. Azaz, ha P_i folyamat hiba miatt nem küld üzenetet a k-adik menetben, akkor egyetlen rákövetkező menetben sem fog. Harmadszor, mivel azokat a végrehajtási sorozatokat vizsgáljuk, amelyekben legfeljebb f hiba fordul elő, az összes szóba jövő kommunikációs minta legfeljebb f számú hibás folyamatot tartalmaz. (Egy P_i folyamatot *hibásnak* tekintünk egy kommunikációs mintában, ha néhány (i, j, k), $k \leq f$ alakú hármas hiányzik a mintából.) Azt mondjuk (csak a fejezet hátralévő részében), hogy egy kommunikációs minta *jó*, ha kielégíti ezt a három feltételt.

6.7.1. példa. Jó kommunikációs minta.

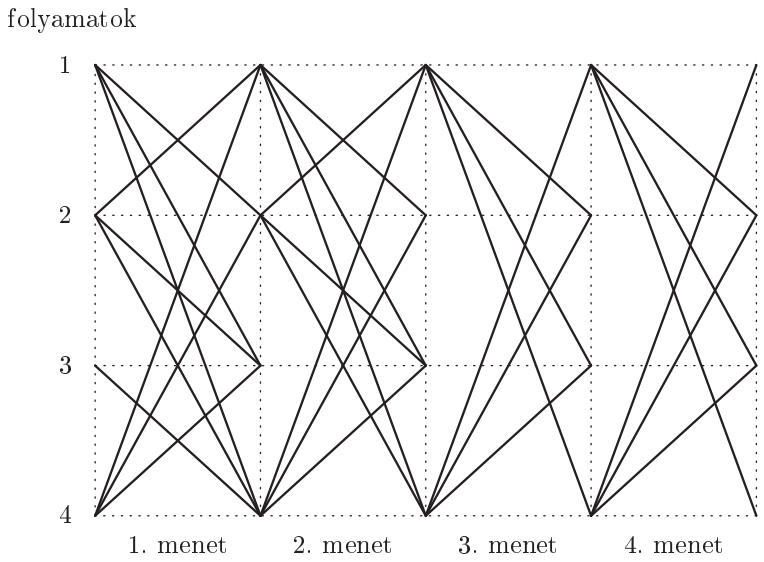
A jó kommunikációs minta egy példáját ábrázolja ($n = f = 4$ esetében) a 6.17. ábra. Ebben a mintában P_3 az első menetben P_4-nek küld üzenetet, de hiba miatt nem küld P_1-nek és P_2-nek. Tehát P_3 megáll az első menetben, és a későbbi menetekben már semmit sem küld. A P_2 folyamat is megáll a második menet végén. P_1 és P_4 hibátlanok.

Most definiáljuk a futam fogalmát, mely a következő kettő kombinációja:

1. bemeneti értékek hozzárendelése az összes folyamathoz;
2. egy jó kommunikációs minta.

(Ez hasonló ahhoz a fogalomhoz, amit az 5.2.1 részben ellenfélnek neveztünk.)

A megegyezési probléma egy bizonyos A algoritmusára minden egyes ρ futam természetes módon definiál egy megfelelő *végrehajtás*(ρ) végrehajtást. Nevezetesen a folyamatok kezdeti állapotát meghatározza a bemeneti állapot összetevők beállítása a ρ-ban megadott bemeneti értékek szerint; az elküldésre kerülő üzeneteket a ρ kommunikációs mintája határozza meg, mivel A üzenet függvényét alkalmazzuk a küldő folyamat eredeti állapotára; a kezdeti állapotokat követő



6.17.. ábra. Példa egy jó kommunikációs mintára.

állapotokat pedig A átmeneti függvénye határozza meg. (Miután egy folyamat hiba miatt többé nem küld üzenetet, arra már nem alkalmazzuk többet az állapotátmenet függvényt.)

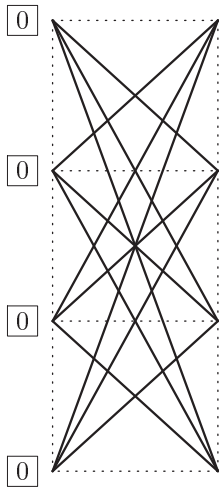
Az $f = 1$ speciális esettel kezdjük a bizonyítást, hogy ezzel egy sejtést adjunk az alsó korlátra.

6.31. tétel . *Legyen $n \geq 3$. Ekkor nincs olyan algoritmus, mely megoldja a megállási hiba problémát n folyamattal, egy hibát eltűrve úgy, hogy az összes hibátlan folyamat minden esetben az első menet végére döntést hozzon.*

Bizonyítás. Tegyük fel indirekten, hogy létezik egy ilyen A algoritmus, és az A kielégíti az alfejezet elején felsorolt összes feltételt.

Konstruáljunk egy láncot az A végrehajtási sorozataiból, melyek legfeljebb egy hibás folyamatot tartalmaznak, és (a) a láncbeli első végrehajtási sorozat egyedüli döntési értékként a 0-át tartalmazza, (b) a láncbeli utolsó végrehajtási sorozat egyedüli döntési értékként az 1-et tartalmazza, és (c) bármely két egymást követő végrehajtási sorozata megkülönböztethetetlen azon folyamat számára, mely mindkettőben hibátlan. Ekkor mivel a hibamentes P_i folyamat számára bármely két egymást követő végrehajtási sorozat megkülönböztethetetlen, P_i ugyanazt a döntést hozza mindkét végrehajtási sorozatban; ebből következik, hogy a két végrehajtási sorozatnak ugyanaz az érték az egyedüli döntési értéke. Ez azt jelenti, hogy a lánc *összes* végrehajtási sorozatának ugyanaz az érték az egyedüli döntési értéke, ami ellentmond az (a) és (b) kikötésnek.

Kezdjük a láncot azzal az *végrehajtás*(ρ_0) végrehajtási sorozattal, melyet az a ρ_0 futam határoz meg, amelyben az összes folyamat a 0 kezdeti értékkel indul, és



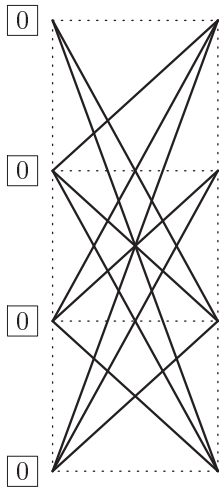
6.18.. ábra. A ρ_0 futam – minden bemenet 0, hibák nincsenek.

nincs hibás folyamat. Ezt a futamot ábrázolja a 6.18. ábra. Az érvényesség szerint *végrehajtás*(ρ_0)-ban az egyedüli döntési érték a 0, a következő végrehajtási sorozatot úgy kapjuk, hogy egy egyszerű üzenetet elhagyunk – amit P_1 folyamat P_2-nek küld. Az eredmény a 6.19. ábrán látható. Ez a végrehajtási sorozat megkülönböztethetetlen *végrehajtás*(ρ_0)-tól minden folyamat számára, kivéve P_1 és P_2 folyamatokat. Mivel $n \geq 3$, van legalább egy ilyen folyamat. Ez a folyamat hibamentes mindkét végrehajtási sorozatban.

Ezután eltávolítjuk P_1 folyamat P_3-nak küldött üzenetét; ez és az előző végrehajtási sorozat megkülönböztethetetlen minden folyamat számára, kivéve P_1 és P_3 folyamatokat, és legalább egy ilyen folyamat létezik. Ezt az utat folytatva, mindig egy üzenetet eltávolítva a P_1 folyamattól, az egymást követő menetek megkülönböztethetetlenek lesznek egy bizonyos hibátlan folyamat számára.

Mikor eltávolítottuk P_1 összes üzenetét, azzal folytatjuk, hogy megváltoztatjuk P_1 bemeneti értékét 0-ról 1-re. Természetesen az eredményül kapott végrehajtási sorozat megkülönböztethetetlen az előzőtől valamennyi folyamat számára, kivéve P_1-et, minthogy P_1 semelyiknek sem küld üzenetet a két végrehajtási sorozatban. Majd egyesével visszahelyezzük P_1 üzeneteit, és továbbra is igaz marad, hogy az egymást követő végrehajtási sorozatok páronként megkülönböztethetetlenek egynéhány hibamentes folyamat számára. Ezzel a módszerrel elérünk a *végrehajtás*(ρ_1)-hez, ahol ρ_1 definíció szerint az a végrehajtás, melyben P_1 bemeneti értéke 1, az összes többi folyamaté 0, és nincsenek hibák.

Majd ezt az eljárást megismételjük a P_2 folyamatra, elsőként egyesével eltávolítjuk az üzeneteit, utána kicseréljük P_2 bemeneti értékét 0-ról 1-re, és visszahelyezzük az üzeneteit. Eredményül kapjuk a *végrehajtás*(ρ_2) végrehajtást, ahol ρ_2 az a futam, melyben a P_1 és P_2 folyamat bemeneti értéke 1, a többié 0, és nincsenek hibák. Megismételve ezt a $P_3, \ldots, P_n$ folyamatra végül a *végrehajtás*(ρ_n)



6.19.. ábra. Az eredmény, ha egy üzenetet eltávolítunk ϱ_0 futamból.

végrehajtási sorozatot kapjuk, ahol ρ_n az a futam, melyben az összes folyamat az 1 értékkel kezd és nincsenek hibák.

Ezzel elkészítettük azt az *végrehajtás*(ρ_0)-tól *végrehajtás*(ρ_n)-ig tartó láncot, mely eleget tesz a (c) kikötésnek. Azonban az érvényességi feltételből következik, hogy az egyedüli döntési érték a *végrehajtás*(ρ_0)-ban 0, a *végrehajtás*(ρ_n)-ben viszont 1, ami ellentmond (a) és (b) kikötéseknek. Tehát megkaptuk azt a láncot, ami ellentmondáshoz vezet. □

Mielőtt az általános esetre rátérünk, még egy előzetes elemzést végzünk – az $f = 2$ esetre.

6.32. tétel . *Legyen $n \geq 4$. Ekkor nincs olyan algoritmus, mely megoldja a megállási hiba problémát n folyamattal, két hibát eltűrve úgy, hogy az összes hibátlan folyamat minden esetben a második menet végére döntést hozzon.*

Bizonyítás. Tegyük fel most is, hogy létezik ilyen algoritmus. Az előzőhöz hasonló módon, ismét készítünk egy láncot, mely eleget tesz ugyanazon (a), (b), (c) feltételeknek, mint amit az előző bizonyításban adtunk. Minden i, $0 \leq i \leq n$ esetében, ρ_i jelölje azt a futamot, melyben a $P_1, \ldots P_i$-nek 1 a bemenete, $P_{i+1}, \ldots P_n$-nek pedig 0, és nincsenek hibák. A lánc az *végrehajtás*(ρ_0)-val indul, *végrehajtás*(ρ_n)-nel fejeződik be, köztük az összes *végrehajtás*(ρ_i) végrehajtási sorozattal.

Az *végrehajtás*(ρ_0)-val kezdve, elsőként a P_1 folyamat kiiktatása a célunk. Amikor csak egy menettel foglalkoztunk, akkor egyszerűen egyesével eltávolítottuk P_1 üzeneteit. Most probléma nélkül megtehetjük, hogy egyesével eltávolítjuk P_1 második menetbeli üzeneteit, de ha eltávolítjuk P_1 egy másik, P_i folyamat számára küldött első menetbeli üzenetét egy lépésben, akkor már nem marad igaz,

hogy a két egymást követő menet megkülönböztethetetlen a többi hibamentes folyamat számára. Ez abból adódik, hogy P_i a második menetben értesítheti a többi folyamatot, hogy kapott-e üzenetet az első menetben P_1-től.

Ezt a problémát úgy oldjuk meg, hogy több lépésben távolítjuk el P_1 folyamat első menetbeli P_i-nek küldött üzenetét. A közbenső végrehajtási sorozatokban P_1 és P_i folyamatokat hibásnak tekintjük; ez megengedhető, hisz $f = 2$. Kezdjük tehát azzal a végrehajtási sorozattal, melyben P_1 üzenetet küld P_i-nek az első menetben és P_i hibátlan. Egyesével eltávolítjuk P_i második menetbeli üzeneteit amíg megkapjuk azt a végrehajtási sorozatot, amelyben P_1 küld P_i-nek üzenetet az első menetben, de P_i nem küld üzenetet a második menetben. Majd eltávolítjuk P_1 első menetbeli P_i-nek küldött üzenetét; az eredményül kapott végrehajtás megkülönböztethetetlen az előzőtől az összes folyamat számára, kivéve P_1 és P_i folyamatokat. Majd egyesével visszahelyezzük a P_i által küldött második menetbeli üzeneteket, amíg megkapjuk azt a végrehajtási sorozatot, amelyben P_1 nem küld P_i-nek üzenetet az első menetben, és P_i hibátlan. Ezzel elértük a célunkat, eltávolítottuk P_1 első menetbeli, P_i-nek küldött üzenetét, miközben az egymást követő végrehajtási sorozatok párosával megkülönböztethetetlenek maradtak egynéhány hibátlan folyamat számára.

Ezzel a módszerrel egyesével eltávolítjuk P_1 első menetbeli üzeneteit, amíg P_1 egyetlen üzenetet sem küld. Ekkor P_1 bemeneti értékét 0-ról 1-re cseréljük, ahogy az előbb is tettük. Folytatjuk az eljárást, csak „ellenkező irányban", egyesével visszahelyezve P_1 első menetbeli üzeneteit. Megismételve ezt az eljárást a $P_2, \ldots, P_n$ folyamatokra megkapjuk a kívánt láncot. □

Most bebizonyítjuk az általános tételt.

6.33. tétel . *Legyen $n \geq f + 2$. Ekkor nincs olyan algoritmus, mely megoldja a megállási hiba problémát n folyamattal, f hibát eltűrve úgy, hogy az összes hibátlan folyamat minden esetben a f-edik menet végére döntést hozzon.*

A 6.31. és 6.32. tételek bizonyításaiban megtalálhatjuk a 6.33. tétel bizonyításának főbb gondolatait. Az általános eset bizonyításához hosszabb láncokat készítünk, f darab folyamathibát megengedve. A bizonyítás formálisabb lesz, mint a 6.31. tétel és a 6.32. tétel bizonyítása. Ehhez néhány jelölést vezetünk be.

Elsőként, ha ρ és ρ' futamokban a P_i folyamat hibátlan, akkor a $\rho \overset{i}{\sim} \rho'$ azt jelenti, hogy *végrehajtás*$(\rho) \overset{i}{\sim}$ *végrehajtás*(ρ') – vagyis a ρ és ρ' futamok által generált végrehajtások megkülönböztethetetlenek P_i folyamat számára. Jelölje $\rho \sim \rho'$ azt, ha fennáll a $\rho \overset{i}{\sim} \rho'$ néhány P_i folyamatra, melyek ρ és ρ' futamok mindegyikében hibátlanok. Továbbá jelölje a $\sim$ reláció $\rho \approx \rho'$ tranzitív lezárását.

Majd vegyük észre, hogy mindazon kommunikációs mintának, mely a 6.31. és 6.32. tételek bizonyításaiban szereplő láncokban előfordul, van egy bizonyos egyszerű tulajdonsága. Ezt a tulajdonságot jellemezzük a következő definícióval. Azt mondjuk, hogy egy jó kommunikációs minta *szabályos*, ha minden k $(0 \leq k \leq f)$ értékre legfeljebb k darab folyamat hibázik (egy üzenetet sem küld) a k-adik menet végéig. Azt mondjuk, hogy a futam, vagy végrehajtás *szabályos*, ha a kommunikációs mintája szabályos.

Végül, ha ρ egy tetszőleges futam és $0 \leq k \leq f$, legyen $hn(\rho, k)$ az a futam – a ρ egy olyan változata, mely *hiba nélküli* k időpont után – melynek bemenete azonos ρ bemenetével, a kommunikációs mintája ugyanaz mint a ρ futamé az első k menetben, és a továbbiakban újabb hiba nem fordul elő. Nyilvánvaló módon adódnak a hn futamokra az alábbi tulajdonságok.

6.34. lemma . *Ha a ρ egy szabályos futam, akkor*

1. *tetszőleges k $(0 \leq k \leq f)$ esetében $hn(\rho, k)$ szabályos;*
2. *ha ρ' azonos ρ-val, kivéve, hogy néhány P_i folyamat, mely ρ-ban hibázik, ρ'-ben későbbi menetben lesz hibás, akkor ρ' szabályos;*
3. *ha a $(k+1)$-edik menetben egyik folyamat sem hibázik, akkor $hn(\rho, k) = hn(\rho, k+1)$.*

A 6.33. tétel bizonyításának lelke a következő erős lemma, mely kimondja, hogy *bármely két* azonos bemenettel rendelkező szabályos végrehajtási sorozat között létrehozhatunk egy láncot.

6.35. lemma . *Legyen A egy n-folyamatos algoritmus a megállási hiba problémára, mely f számú hibát eltűr, és melyben a hibamentes folyamatok minden esetben döntést hoznak az f-edik menet végéig. Legyen ρ és ρ' két szabályos futama az A algoritmusnak, melyekben a bemenet azonos, ekkor $\rho \approx \rho'$.*

Bizonyítás. A bizonyításhoz az alábbi paraméteres segédtételt fogjuk belátni. A lemma azonnal következik a $k = 0$ esetből.

6.36. segédtétel . *Legyen k egy egész szám, $0 \leq k \leq f$. Legyen ρ és ρ' két szabályos futama az A algoritmusnak, melyekben azonos a bemenet és azonos a kommunikációs minta is k meneten keresztül. Ekkor $\rho \approx \rho'$.*

Bizonyítás. A 6.36. állítás bizonyítása k szerinti fordított indukcióval történik, a $k = f$ esettel kezdjük, és a $k = 0$ esettel fejezzük be.

Alapeset: $k = f$. Ez az eset triviálisan adódik a feltevésből, hogy ρ és ρ' bemenete megegyezik, és kommunikációs mintájuk azonos az f-edik menetig, amiből az következik, hogy ρ és ρ' azonos.

Indukciós lépés: feltesszük, hogy $0 \leq k \leq f-1$, és az állítás igaz $k+1$ értékre. Elegendő belátnunk, hogy bármely szabályos ρ futamra fennáll, hogy $\rho \approx hn(\rho, k)$, mivel ezt kétszer alkalmazva megkapjuk a kívánt állítást. Rögzítsünk egy szabályos ρ futamot. A 6.34. lemmából következik, hogy $hn(\rho, k)$ szabályos.

Indukciós feltevésünk szerint $hn(\rho, k+1) \approx \rho$, tehát elegendő megmutatnunk, hogy $hn(\rho, k) \approx hn(\rho, k+1)$. Ha a $(k+1)$-edik menetben egyik folyamat sem hibázik, akkor a 6.34. lemmából következik, hogy $hn(\rho, k) = hn(\rho, k+1)$, és kész vagyunk. Tegyük fel tehát, hogy legalább egy folyamat hibázik a ρ futam $(k+1)$-edik menetében. Jelölje I azon folyamatok halmazát, melyek így tesznek.

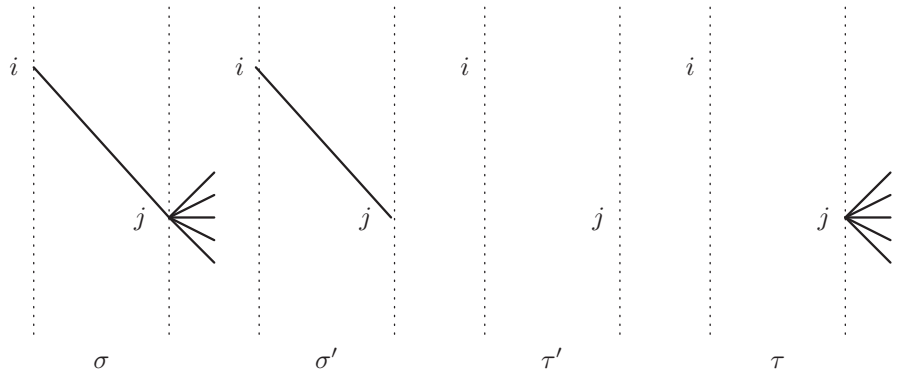
Legyen ρ_0 az a folyamat, mely azonos $hn(\rho, k)$-val, kivéve hogy, az összes I-beli folyamat hibázik az $(f+1)$-edik menet végéig. Ekkor a 6.34. lemma második állításából (ρ-ra alkalmazva) kapjuk, hogy ρ_0 szabályos.

Minthogy ρ_0 és $hn(\rho, k)$ szabályos futamok, melyek azonosak $k+1$ meneten keresztül, alkalmazhatjuk az indukciós feltevést, hogy megmutassuk, $\rho_0 \approx hn(\rho, k)$. Ezért annak bizonyításához, hogy $hn(\rho, k) \approx hn(\rho, k+1)$ fennáll, elegendő belátni, hogy $\rho_0 \approx hn(\rho, k+1)$.

Készíteni fogunk egy olyan szabályos futamokból álló láncot, mely összeköti a ρ_0 és $hn(\rho, k+1)$ futamokat. Az egyetlen különbség ρ_0 és $hn(\rho, k+1)$ között az, hogy néhány olyan üzenet, melyet ρ_0 futam $(k+1)$-edik menetében az I-beli folyamatok küldenek, hiányzik a $hn(\rho, k+1)$ futamban. Ezért egyesével eltávolítjuk ezeket az üzeneteket, úgy, hogy mást nem változtatunk a futamokban.

Példaként tekintsük a P_i által P_j-nek küldött üzenet eltávolítását, ahol $i \in I$. Legyen σ az a futam, mely tartalmazza az üzenetet, τ pedig az, amelyik nem; meg kell mutatnunk, hogy $\sigma \approx \tau$. Ha $k+1 = f$, akkor σ és τ megkülönböztethetetlenek P_i és P_j kivételével az összes folyamat számára; mivel $n \geq f+2$ és P_i hibás, ezért van legalább egy hibátlan folyamat. Tehát $\sigma \approx \tau$, amire szükségünk volt.

Másrészről, ha $k+1 \leq f-1$, akkor definiáljuk a σ' és τ' futamokat úgy, hogy külön-külön azonosak legyenek a σ és τ futamokkal, de P_j a $(k+2)$-edik menet elején hibás lesz (ha eddig még nem lett volna). Lásd a 6.20. ábrát.



6.20.. ábra. A P_i folyamat P_j számára küldött üzenetének eltávolítása a $(k+1)$-edik menetben a 6.36. tétel bizonyításában.

A σ' és τ' futamok szabályosak, mivel σ és τ mindegyike legfeljebb $k+1 \leq f-1$ hibát tartalmaz, és csak egy új hibát engedélyeztünk az új $(k+2)$-edik menetben. Továbbá az indukciós feltevés alapján $\sigma \approx \sigma'$ és $\tau \approx \tau'$. Fennáll $\sigma' \approx \tau'$ is, mert megkülönböztethetetlenek P_i és P_j kivételével az összes folyamat számára. Innen adódik, hogy $\sigma \approx \tau$.

Megmutattuk, hogy a ρ_0 és $hn(\rho, k+1)$ futamokat összekötő lánc létrehozható, így $\rho_0 \approx hn(\rho, k+1)$, ebből $\rho \approx hn(\rho, k)$, amire szükségünk volt. □

Korábban már említettük, hogy a 6.36. állításból közvetlenül adódik a 6.35. lemma. □

Most kiterjesztjük a 6.35. lemmát különböző bemenetek esetére.

6.37. lemma . *Legyen A egy n-folyamatos algoritmus a megállási hiba problé-*

mára, mely f számú hibát eltűr, és melyben a hibamentes folyamatok minden esetben döntést hoznak az f-edik menet végéig. Legyen ρ és ρ' két szabályos futama az A algoritmusnak, ekkor $\rho \approx \rho'$.

Bizonyítás. A 6.35. lemma szerint minden ρ futam összefügg a hiba nélküli változatával, azaz $\rho \approx hn(\rho, 0)$. Így az általánosság csorbítása nélkül feltehetjük, hogy a lemma szerinti ρ és ρ' mindketten hiba nélküliek.

Ha ρ és ρ' bemenete ugyanaz, akkor a két futam megegyezik, és nincs mit bizonyítani.

Tegyük fel, hogy ρ és ρ' bemenete különbözik egy adott P_i esetében; legyen a bemenet 0 ρ-ban és 1 ρ'-ben. Legyen σ és σ' külön-külön azonos ρ és ρ' futamokkal, kivéve, hogy P_i az induláskor hibás lesz. Ekkor a 6.35. lemmából következik, hogy $\rho \approx \sigma$ és $\rho' \approx \sigma'$. Valamint $\sigma \approx \sigma'$, mivel σ és σ' megkülönböztethetetlenek mindegyik folyamat számára, kivéve P_i-t. Ebből következik, hogy $\rho \approx \rho'$, amit bizonyítani akartunk.

Végül tegyük fel, hogy ρ és ρ' bemenete nemcsak egy folyamat esetében különbözik. Készítünk egy hiba nélküli folyamatokból álló láncot ρ-tól ρ'-ig úgy, hogy egy lépésben mindig pontosan egy folyamat bemenetét változtatjuk meg a láncban. Az így kapott láncban minden egyes lépésre alkalmazhatjuk az előző esetet. Így beláttuk, hogy $\rho \approx \rho'$. □

A 6.37. lemmára támaszkodva könnyen bizonyítható a 6.33. tétel. Azt már tudjuk, hogy az összes szabályos futam összekapcsolható láncokkal; most megvizsgáljuk a döntési értékeket ezekben a futamokban. Legyen $n > f$, a befejezési és megegyezési tulajdonságokból következik, hogy minden ρ futamra létezik egy egyedüli döntési érték, legyen ez a *dönt*(ρ), mely *végrehajtás*(ρ) során keletkezik. A következő lemma szerint azon futamokban, melyek között a $\sim$ vagy $\approx$ reláció fennáll, szükségszerűen ugyanaz a döntési érték születik.

6.38. lemma .

1. *Ha $\rho \sim \rho'$, akkor dönt(ρ) = dönt(ρ').*
2. *Ha $\rho \approx \rho'$, akkor dönt(ρ) = dönt(ρ').*

Bizonyítás. Az első rész bizonyításához emlékezzünk vissza, hogy $\rho \sim \rho'$ azt jelenti, hogy létezik egy olyan P_i folyamat, mely hibátlan a ρ és ρ' futamokban, így *végrehajtás*(ρ) $\overset{i}{\sim}$ *végrehajtás*(ρ'). Ebből következik, hogy P_i ugyanazt a döntést hozza *végrehajtás*(ρ)-ban és *végrehajtás*(ρ')-ben. Tehát *dönt*(ρ) = *dönt*(ρ').

A második rész az elsőből következik.

□

6.33. tétel bizonyítása. Tegyük fel, hogy létezik egy ilyen algoritmus, legyen ez az A; feltesszük, hogy A eleget tesz az alfejezet elején adott feltételeknek.

Legyen ρ_0 az A algoritmus egyik futama, melyben minden folyamat 0-val kezd, és nincsenek hibák, és ρ_1 egy másik futam, amelyben minden folyamat 1-gyel kezd, és nincsenek hibák. A 6.37. lemmából következik, hogy $\rho_0 \approx \rho_1$.

Ekkor a 6.38. lemma második állítása szerint $\mathit{dönt}(\rho_0) = \mathit{dönt}(\rho_1)$. Az érvényességi szabályból viszont az következik, hogy $\mathit{dönt}(\rho_0) = 0$ és $\mathit{dönt}(\rho_1) = 1$, ami ellentmondás. □

Gyengébb érvényességi feltétel. Vegyük észre, hogy a bizonyítás akkor is helytálló, ha az érvényességi feltételt gyengítjük, ahogy a 6.6. alfejezetben tettük, a gyenge bizánci megegyezésnél. Tehát azt is beláttuk, hogy a gyenge bizánci megegyezési probléma is legalább $f + 1$ menetes az $n \geq f + 2$ kikötés mellett.

6.8.. Megjegyzések a fejezethez

A fejezetben található eredmények nagy része a két alapműből származik, az egyik Pease, Shostak és Lamport [237], a másik Lamport, Shostak és Pease [187] munkája. A két cikk közli a bizánci megegyezéshez szükséges processzorok darabszámára vonatkozó $3f + 1$ alsó és felső korlátot, továbbá egy algoritmust a hitelesítéssel történő megegyezésre, mindegyik eredmény a teljesen összefüggő gráfok esetére vonatkozik. A második cikk a problémát nem folyamatokon, hanem a támadni készülő tábornokok példáján keresztül mutatja be. A *bizánci* elnevezést a második cikk vezeti be erre a hiba modellre.

Kissé bővebben, ez a két cikk definiálja a bizánci megegyezési problémát, mely a repülőgépek SIFT (Software-Implemented Fault Tolerance) irányító rendszerében felmerülő probléma absztrakciójaként vetődik fel [289]. A [237] cikkben leírt algoritmusok egy az *EIGY* fához hasonló exponenciális adatszerkezetet használnak; a bizánci megegyezés algoritmus az EIGYBiz-hez, a hitelesítést használó algoritmus pedig a az EIGYStop-hoz hasonló. A [187]-ben található algoritmusok nagyon hasonlítanak ezekhez, csak rekurzív megfogalmazásban. A [237]-ben közölt, a folyamatok darabszámára vonatkozó $n \leq 3f$ megoldhatatlansági eredmény magában foglalja az általunk is tárgyalt forgatókönyv pontos megadását. A [187]-ben található megoldhatatlansággal kapcsolatos bizonyítás mutatja be a 6.27. tétel bizonyításában használt három az egy ellen esetre való redukálást.

Dolev és Strong [93] által, a hitelesítéssel kibővített bizánci megegyezésre javasolt algoritmusok hasonlóak a HalmazTerjed és OptHalmazTerjed algoritmusokra. Dolev [94] vizsgálta a bizánci megegyezési problémát olyan gráfokra, melyekre nem kötjük ki a teljes összefüggőséget. Ő bizonyította a 6.29. tételben bemutatott összefüggőségi korlátokat, konkrét forgatókönyvek megadásán keresztül. Dolev, Reischuk és Strong [99] javasoltak „korán megálló" algoritmusokat bizonyos kedvező kommunikációs mintákra. Egyéb korán megálló algoritmusokat közöl Dwork és Moses [105] és Halpern, Moses és Waarts [145].

Bar-Noy, Dolev, Dwork és Strong definiálták az *EIGY* fa adatszerkezetet, valamint leírták az EIGYBiz algoritmust lényegében ugyanabban a formában, ahogy ebben a könyvben tárgyaltuk [39]. A TurpinCoan algoritmus leírása [279]-ben található.

A bizánci megegyezésre az első, kommunikációs bonyolultságot tekintve polinomiális algoritmust Dolev és Strong [101] javasolták; az algoritmust később Dolev, Fischer, Fowler, Lynch és Strong [96] továbbfejlesztették, korlátként a $2f + 3$ értéket megadva. Coan [82] készített egy kompromisszumos algoritmust,

mely csökkentette a menetek számat $(1+\epsilon)f$ értékre, tetszőleges $\epsilon > 0$ mellett. A következetes üzenetszórás alapfogalmat és a KövetkezetesÜzenSzórás algoritmust Srikanth és Toueg [269] vezeti be. A PoliBiz algoritmus Srikanth és Toueg [269] és Dolev és mások [96] által közölt algoritmusokon alapszik. További kutatások eredményeként Moses és Waarts [231], Berman és Garay [49] és Garay és Moses [133] előállítottak olyan $f+1$ menetes algoritmusokat a bizánci megegyezésre, melyek kommunikációs bonyolultsága polinomiális, az utolsó ezek közül eléri az $n = 3f+1$ legkisebb korlátot a folyamatok darabszámát tekintve. Sajnos, ezek az algoritmusok bonyolultak.

Említettük már, hogy a folyamatok darabszámára vonatkozó $n > 3f$ alsó korlát bizonyítása [237, 187] cikkekben, az összefüggőségre vonatkozó alsó korlát bizonyítása pedig [94]-ben található. Bár a könyvben szereplő bizonyítást Fischer, Lynch és Merritt [122] adták. Menger tételét Menger bizonyította [225] és Harary könyvében [147] jelent meg.

A gyenge bizánci megegyezést Lamport definiálja [178]. A gyenge bizánci megegyezéshez szükséges folyamatok számának alsó korlátjára vonatkozó eredmény Lamporttól származik [178], de a könyvünkben adott bizonyítás Fischer, Lynch és Merritt munkája[122].

A megegyezés eléréséhez szükséges menetek számára vonatkozó alsó becslést elsőként Fischer és Lynch [119] bizonyították a bizánci hiba modellben. Később ezt az eredményt Dolev és Strong [93], valamint DeMillo, Lynch és Merritt [88] kiterjesztik a hitelesítéses bizánci hiba modell esetére. A megállási hibák esetére való kiterjesztés Merrittnek [226] tulajdonítható, Dolev és Strong elképzeléseit [101] alapul véve. Az eredmény egy másfajta bizonyítását Dwork és Moses [105] adják; az ő bizonyításukban a különböző esetek futási idejének finomabb elemzését is megtaláljuk. Feldman és Micali [113] konstans idejű véletlenített megoldást adnak „titok megosztás" technikát használva.

Fischer dolgozatában [117] áttekinti a megegyezési problémával kapcsolatos korábbi eredmények nagy részét.

Igen tekintélyes fejlesztő munka folyik a Draper Laboratóriumban hibatűrő mikroprocesszorok és processzor hibadiagnózis algoritmusok tekintetében, a bizánci megegyezésre [172, 173] támaszkodva. Az eredményeket olyan biztonsági szempontból kiemelt alkalmazási területeken használják fel, mint az embernélküli víz alatti járművek, a nukleáris támadófegyverekkel felszerelt tengeralattjárók, valamint az atom-erőművek irányítása.

6.9.. Gyakorlatok

6-1. 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.

6-2. 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 megál-

lási hiba modellben is.

6-3. Bizonyítsuk be a 6.2. lemmát.

6-4. Kövessük nyomon a HalmazTerjed 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.

6-5. Tekintsük a HalmazTerjed 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.

6-6. Legfeljebb mennyi lehet a hibamentes folyamatok által hozott, egymástól különböző döntési értékek darabszáma, ha a HalmazTerjed algoritmus $f + 1$ menet helyett csak f menetben fut.

6-7.

(a) Találjunk egy másik lehetséges, helyesen működő döntési szabályt a HalmazTerjed algoritmusban, amelyik eltér szövegben megadottól.

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

6-8. Terjesszük ki a HalmazTerjed algoritmust, a helyesség bizonyítását, és az elemzést, tetszőleges (nem szükségképp teljes) összefüggő gráfokra.

6-9. Készítsük el az OptHalmazTerjed algoritmus kódját. Tegyük a szövegben adott bizonyítást teljessé a 6.5., 6.6. és 6.7. lemmák bizonyításával.

6-10. Tekintsük a következő egyszerű algoritmust a megállási hibák mellett történő megegyezésre, egy adott V értékhalmaz esetében. Legyen mindegyik folyamatnak egy *min_érték* változója, melyet induláskor a saját kezdeti értékére állít be. Az $f + 1$ menet mindegyikében a folyamatok 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 folyamat döntési értéke *min_érték* lesz. Készítsük el ennek az algoritmusnak a kódját és bizonyítsuk be (vagy direkt módon, vagy szimulációval), hogy helyesen működik.

6-11. Kövessük nyomon az EIGYStop 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.

6-12. Bizonyítsuk be a 6.11. lemmát.

6-13. Bizonyítsuk be a 6.12. lemma első állítását.

6-14. Tekintsük az EIGYStop algoritmust f hibára. Tegyük fel, hogy az algoritmus $f + 1$ menet helyett csak f menetben fut, ugyanazzal a döntési értékkel. Találjunk egy olyan végrehajtást mely megsérti a helyességi feltételeket.

6-15. Legfeljebb mennyi lehet a hibamentes folyamatok által hozott, egymástól különböző döntési értékek darabszáma, ha az EIGYStop algoritmus $f+1$ menet helyett csak f menetben fut.

6-16. Egy másfajta bizonyítási módszer a HalmazTerjed algoritmus helyességének bizonyítására, az EIGYStop algoritmusra vonatkozó szimulációs reláció módszere. Annak érdekében, hogy ezt megvalósítsuk, elsőként érdemes kiterjeszteni az EIGYStop algoritmust, és megengedni, hogy valamennyi P_i folyamat közreadja az összes menetben az összes értéket, nemcsak azon csúcsokhoz kapcsolt értékeket, melyek címkéje nem tartalmazza az i-t. Be kell látni, hogy a kiterjesztés nincs hatással a helyességre. Továbbá az EIGYStop leírásában néhány részletet ki kell egészíteni, így például a *menetek* és *döntések* változók megfelelő módon történő kezelését. Ezután a HalmazTerjed és a módosított EIGYStop algoritmusokat egymás mellett futtathatjuk, ugyanazzal a kezdeti értékhalmazzal indítva, és olyan hibákkal, melyek ugyanazon folyamatoknál pontosan azonos időben történnek.
Bizonyítsuk a HalmazTerjed algoritmus helyességét ezzel a módszerrel. A bizonyítás lelke az alábbi szimulációs kapcsolat lehet, melyben a két algoritmus állapotai szerepelnek egyező számú menet végrehajtása után.

6.9.1. állítás. *Tetszőleges r, $0 \leq r \leq f+1$ menet után az alábbi két állítás igaz.*

(a) A menetek és döntések változók értékei megegyeznek a két algoritmus állapotaiban.
(b) Minden i értékre a W_i halmaz a HalmazTerjed *algoritmusban egyenlő az* EIGYStop *algoritmusban P_i fájának csúcsait díszítő érték-ekből álló halmazzal.*

Figyeljünk, hogy azokhoz az EIGYStop algoritmust kiegészítő invariáns állításokhoz, melyek a szimuláció létrehozásához szükségesek, minden szükséges definíciót megadjunk, valamint a szükséges bizonyításokat is végezzük el.

6-17. Bizonyítsuk be az OptEIGYStop algoritmus helyességét az alábbi két módszer egyikével:
(a) az EIGYStop algoritmus szimulálásával, ahogy az OptHalmazTerjed algoritmus bizonyítását végeztük az HalmazTerjed algoritmusra vonatkozó szimulációval;
(b) az OptHalmazTerjed-re vonatkozó szimulációval.

6-18. Bizonyítsuk be az EIGYStop és OptEIGYStop algoritmusok helyességét a hitelesített bizánci hiba modellben. Néhány, az EIGYStop bizonyításához szükséges alapvető tényt fogalmaz meg a következő állítás, a 6.12. lemma állításaihoz hasonló módon.

6.9.2. állítás. *$f+1$ menet után fennállnak a következők.*

(a) Ha P_i és P_j hibamentes folyamatok, érték$(y)_i = v \in V$ és xj az y prefixe, akkor érték$(x)_j = v$.

(b) *Ha v eleme tetszőleges hibamentes folyamat érték-eiből álló halmaznak, akkor v kezdőértéke volt valamely folyamatnak.*
(c) *Ha P_i egy hibamentes folyamat és $v \in V$ benne van P_i érték-eiből álló halmazban, akkor van olyan y címke, mely i-t nem tartalmazza, és melyre $v = \text{érték}(y)_i$.*

Ezek a digitális aláírás tulajdonságaiból következnek.

6-19. *Kutatási probléma:* definiáljuk a hitelesített bizánci hiba modellt formálisan, és bizonyítsuk a hatékonysággal és a korlátokkal kapcsolatos eredményeket.

6-20. Mutassunk példát az EIGYSTOP algoritmus olyan végrehajtási sorozatára, mely bebizonyítja, hogy az EIGYSTOP algoritmus nem oldja meg a bizánci hiba problémát.

6-21. Tekintsük az EIGYBIZ algoritmust hét folyamattal és három menettel. Tetszőlegesen válasszunk két folyamatot, melyek hibásak lesznek, és adjunk véletlen bemeneti értéket valamennyi folyamatnak és azon üzeneteknek, melyeket a hibás folyamatok küldenek. Számítsuk ki a végrehajtási során előállított információt, és bizonyítsuk, hogy a helyességi feltételeket kielégítik.

6-22. Bizonyítsuk be, hogy az EIGYBIZ algoritmusban nem szükséges az, hogy az *EIGY* fa valamennyi csúcsa közös legyen.

6-23. Tekintsük az EIGYBIZ algoritmust. Keressünk olyan konkrét végrehajtásokat, melyek megmutatják, hogy az algoritmus rossz eredményt adhat, ha
(a) hét csúccsal, két hibával és két menetben fut;
(b) hat csúccsal, két hibával és három menetben fut.

6-24. A TURPINCOAN algoritmus az első és második menetben az $n - f$ küszöb értéket használja. Milyen másik két küszöb érték lenne alkalmas, hogy az algoritmus továbbra is helyesen működjön?

6-25. Tekintsük a TURPINCOAN algoritmust egy helyett *két*, F és G, hibás folyamatokat tartalmazó halmazzal. Mindkét halmaz legfeljebb f folyamatot tartalmaz. Az F-beli folyamatok helyesen működnek, kivéve az első és második menetet, melyben hibás üzenetet küldhetnek. A G-beli folyamatok a bináris bizánci megegyezés szubrutin végrehajtása alatt (és csak akkor) viselkedhetnek hibásan. Milyen helyességi feltételek biztosíthatók a kombinált algoritmust használva ezen feltételek mellett? Bizonyítsuk.

6-26. Legyen most $n > 4f$ a feltevésünk. Tervezzünk algoritmust, mely a bináris bizánci megegyezés szubrutinját használja, és megoldja a többértékű bizánci megegyezést. Az algoritmust a TURPINCOAN algoritmusból kiindulva készíthetjük el, a két plusz menet helyett csak egyet használva.

6-27. Mutassuk meg, hogy nem létezik felső korlát arra az időre a KÖVETKEZETESÜZENSZÓRÁS algoritmusban, amíg egy hibátlan folyamat elfogadhatja az (m, i, r) üzenetet. Azaz tetszőleges t-re állítsuk elő a KÖVETKEZETESÜZENSZÓRÁS algoritmus olyan végrehajtását, melyben néhány hibátlan folyamat az üzenetet az $r' \geq r + t$ menetben fogadja el.

6-28. Tervezhető-e algoritmus a következetes üzenetszórás megvalósítására a bizánci hiba modellben $f \gg 1$ hiba esetében, a modellt azzal a további tulajdonsággal kiegészítve, hogy a hibátlan folyamatok soha nem fogadják el az (m, i, r) üzenetet az $(r+1)$-edik menet után.
Adjunk egy ilyen algoritmust, és bizonyítsuk a helyességét, vagy indokoljuk meg, miért nem létezik ilyen algoritmus.

6-29. Írjuk le a PoliBiz algoritmus legrosszabb esetének megfelelő végrehajtási sorozatot, melyben létezik olyan hibamentes P_i folyamat, melyre az a legkorábbi menet, amikorra a folyamat elfogadja $2f+1$ számú, egymástól különböző folyamat üzenetét, pontosan a $2(f+1)$-edik menet.

6-30. A Zöldfülűek Számítástechnikai Részvénytársaság programozója úgy módosította a PoliBiz algoritmus megvalósítását, hogy a $2s-1$ alakú menetekben az elfogadás küszöb értéke $f+s-1$ helyett $s-1$, és a döntési küszöb értéke $2f+1$ helyett $f+1$. Helyes ez a módosítás? Bizonyítsuk, vagy mutassunk ellenpéldát.

6-31. Tervezzünk algoritmust a bizánci megegyezésre, melynek kommunikációs bonyolultsága polinomiális, bemeneti értékeinek halmaza általános, úgy, hogy nem használjuk szubrutinként a bináris bizánci megegyezést. Az algoritmus használhatja a következetes üzenetszórás működési módszert, de egy hatékonyabb megvalósítást kellene tervezni, mint a KövetkezetesÜzenSzórás algoritmus.

6-32. Tervezzünk egy algoritmust a megegyezésre megállási hibák esetében, mely kielégíti az alábbi *korai megállás* tulajdonságot: ha az algoritmus egy menetében csak $f' < f$ számú folyamat hibázik, akkor az az időpont, amelyre az összes hibamentes folyamat döntést hoz legfeljebb kf', ahol k egy konstans. Készítsünk hasonlót a bizánci megegyezésre.

6-33. Tervezzünk protokollt négy folyamatra, melyek egy teljesen összefüggő gráfban helyezkednek el, úgy, hogy *akár* egy bizánci hibát, *akár* három megállási hibát eltűrjön. Próbáljuk meg minimalizálni a menetek számát.

6-34. *Kutatási probléma:* találjunk ki a bizánci megegyezés megoldására *egyszerű* $f+1$ menetes protokollt, melyhez csak $3f+1$ folyamat szükséges, és kommunikációs bonyolultsága polinomiális.

6-35. Ennek a gyakorlatnak az a célja, hogy megvizsgáljuk a 6.26. lemma bizonyításában található konstrukciót, mely két háromszög alakú rendszert illeszt egybe úgy, hogy egy hatszögű rendszer az eredmény.

(a) Nagyon körültekintően írjuk le azt az A algoritmust egy három-folyamatos teljes gráfra, mely megoldja a hiba nélküli megegyezési problémát, azaz a bizánci megegyezés problémát arra a speciális esetre, ahol egyetlen folyamat sem hibázik.

(b) Most készítsük el az S rendszert, az A algoritmus két példányának összeillesztésével, ahogy a 6.26. lemma bizonyításában tettük. Írjuk le nagyon alaposan az S azon végrehajtását, melyben P_1, P_2 és P_3 a 0 bemenettel, P'_1, P'_2 és P'_3 az 1 bemenettel kezd.

(c) Megoldja S a hiba nélküli megegyezési problémát (a hatszög hálózatban)? Bizonyítsuk, hogy igen, vagy adjunk egy végrehajtást, mely bizonyítja, hogy

nem.

(d) Találhatunk-e olyan három-folyamatos A algoritmust, melynek *tetszőlegesen sok* példányát összeillesztve egy gyűrűbe, az eredményül kapott rendszer minden esetben megoldja a hiba nélküli megegyezési problémát?

6-36. Mennyi lesz a hibás folyamatok számának maximuma, melyet a bizánci megegyezés algoritmusai az alábbi hálózati gráfokban futva még eltűrnek?

(a) Egy n méretű gyűrű.

(b) Egy három dimenziós kocka, egy oldalán m olyan csúccsal, mely csúcsok csak a három dimenzióban elhelyezkedő szomszédaikkal vannak összekötve.

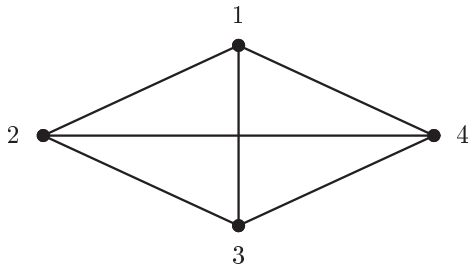
(c) Egy teljes páros gráf, mindkét komponensében m csúccsal.

6-37. Adjunk még alaposabb bizonyítást a bizánci megegyezés megoldhatatlanságára, amikor $n = 2$ és $f = 1$.

6-38. Elemezzük a bizánci megegyezés algoritmusának futási idejét, az üzenetek számát, a kommunikáció bitszámát a 6.29. tétel bizonyításában leírt általános gráfok esetében. Tudnánk-e javítani valamelyiken?

6-39. Mutassuk meg nagyon körültekintően, hogy a 6.29. tétel bizonyításában használt egyszerűsítések, amikor azt bizonyítottuk, hogy a bizánci megegyezés lehetetlen $f = 1$ és $\ddot{o}ssz\_f\ddot{u}gg(G) \leq 2$ esetében, megengedhetők. Azaz, lássuk be, hogy egy algoritmus létezése olyan esetre, amelyben az 1-es és 3-as csúcsokat tetszőleges, összefüggő rész-gráfokkal helyettesítjük, maga után vonja, hogy létezik algoritmus arra az esetre, amikor ezek egyszerű csúcspontok.

6-40. Fontoljuk meg újra azt a bizonyítást, hogy a bizánci megegyezés nem érhető el a 6.11. ábrán látható gráfban. Miért lenne hibás ez a bizonyítás, ha kiterjesztenénk a 6.21. ábrán látható gráfra?



6.21.. ábra. A hálózat gráfja a 6-40. gyakorlathoz.

6-41. Bizonyítsuk be, hogy a bizánci megegyezés $f > 1$ számú hiba esetében nem oldható meg abban a G gráfban, melyre $\ddot{o}ssz\_f\ddot{u}gg(G) \leq 2f$. A bizonyításhoz kétfajta módszert is követhetünk, vagy használjuk a folyamatokat csoportosító érvelést, ahogy azt a 6.29. tétel bizonyításának végén vázoltuk, vagy a 6.27. tételhez hasonlóan redukálást használunk.

6-42. Találjunk egyszerű algoritmust a gyenge bizánci megegyezésre abban a hálózati gráfban, mely két csúcspontból és egy a csúcsokat összekötő élből áll.

6-43. Tegyük teljessé a 6.30. tétel bizonyítását, megmutatva a megoldhatatlanságot a következő két esetre:
(a) amikor $n \leq 3f$ és $f > 1$;
(b) amikor $\ddot{o}ssz\_f\ddot{u}gg(G) \leq 2f$.

6-44. Tekintsük a *bizánci kivégző osztag* problémát, melynek definíciója a következő. Egy teljesen összefüggő hálózati gráfban helyezkedik el az n számú folyamat, melyeknek nincs bemeneti értékük, és változó a kezdési idejük. Azaz mindegyik folyamat egy *tétlen* állapotban kezd, mely nem tartalmaz információt, és amelyből csak *null* üzeneteket küld. Nem változik az állapota addig, amíg nem kap egy speciális *ébresztő* üzenetet kívülről, vagy egy nem *null* üzenetet egy másik folyamattól. A folyamat nem tudja, az éppen aktuális menet számát amikor felébred. A modell hasonló ahhoz, amit a 2.1. alfejezetben leírtunk, kivéve, hogy itt nem feltételezzük, hogy mindegyik folyamat kötelezően kap egy *ébresztő* üzenetet – csak a folyamatok egy tetszőleges részhalmaza. Továbbá megengedjük a bizánci hibákat.
A probléma a *tűz* parancsok kiadása a folyamatoknak, az alábbi feltételek mellett.

Megegyezés. Ha egy hibátlan folyamat *tűz* jelet kap valamely menetben, akkor az összes hibátlan folyamat ugyanabban a menetben kap *tűz* jelet, és hibátlan folyamat semelyik másik menetben sem kap *tűz* jelet.

Érvényesség. Ha az összes hibátlan folyamat *ébresztő* üzenetet kap, akkor az összes hibátlan folyamat azonnal *tüzel*; ha hibátlan folyamat nem kap *ébresztő* üzenetet, akkor hibátlan folyamat soha nem fog *tüzelni*.

(a) Tervezzünk algoritmust a bizánci tüzelő osztag probléma megoldására, ha $n > 3f$ igaz.
(b) Bizonyítsuk be, hogy a problémát nem lehet megoldani, ha $n \leq 3f$.

6-45. Mondjuk ki, majd bizonyítsuk a 6.33. tételt az $f = 3$ speciális esetre.

6-46. Fennáll-e továbbra is a 6.37. lemma, ha a folyamatoktól nem kívánjuk meg, hogy szabályosak legyenek. Bizonyítsuk, vagy adjunk ellenpéldát.

6-47. A 6.7. alfejezetben megmutattuk, hogy az f számú hibát eltűrő, megállási hiba melletti megegyezés nem oldható meg f menetben. Egy hosszú láncot hoztunk létre, mely összeköti azt a két menetet, amelyben mindegyik folyamat hibátlan és ugyanaz a bemenetük. Csak a lánc elkészítésének módját adtuk meg.

(a) Milyen hosszú a futamok lánca?
(b) Mennyivel lehetne rövidíteni a láncot, a megállási hibák helyett bizánci hibákat használva?

6-48. *Kutatási probléma:* adjunk alsó és felső korlátot a megállási probléma megoldásának és/vagy a bizánci probléma megoldásának futási idejére általános (nem szükségszerűen teljes) hálózati gráfokban.

7. fejezet

További megegyezési feladatok

Az előző két fejezetben a megegyezési feladattal foglalkoztunk: az ötödik fejezetben az összehangolt támadással, a hatodik fejezetben a megegyezéssel. Ebben a fejezetben három további megegyezési probléma elemzésével befejezzük a szinkron osztott megegyezés tárgyalását. Ezek a *k-megegyezés problémája*, a *közelítő megegyezés feladata* és az *osztott adatbázisok véglegesítési feladata*. A hatodik fejezethez hasonlóan csak a folyamatok hibáit vesszük figyelembe.

7.1.. k-megegyezés

Először a k-megegyezés problémáját vizsgáljuk, ahol k egy nemnegatív egész szám. Ez a 6. fejezetben elemzett egyszerű megegyezési probléma természetes általánosítása. Most azonban nem azt várjuk a folyamatoktól, hogy pontosan ugyanazt a meghatározott értéket válasszák várjuk a folyamatoktól, hanem megelégszünk azzal, hogy választásukat egy kis (k-elemű) halmaz elemeire korlátozzák.

Az eredeti motiváció matematikai jellegű volt: érdekesnek látszott annak vizsgálata, hogyan változnak az eredmények, ha ilyen egyszerű módon megváltoztatjuk a feltételeket. Gyakorlati helyzetek is elképzelhetők azonban, melyekben ilyen algoritmus hasznos lehet. Például tekintsük olyan közösen is használható erőforrások hozzárendelését, mint amilyenek egy kommunikációs hálózat üzenetszórási frekvenciái. Kívánatos lehet bizonyos számú folyamat olyan megegyezése, hogy nagy mennyiségű adat (például egy videoszalag) szórására adott, kisszámú frekvenciát használnak. Mivel a kommunikáció üzenetszórással valósul meg, tetszőleges számú folyamat megkaphatja az adatokat – ugyanazokat a frekvenciákat használva. A teljes kommunikációs terhelés minimalizálása érdekében célszerű a frekvenciák k számát alacsonyan tartani.

Ebben a fejezetben pontos alsó és felső korlátokat adunk meg a k-megegyezési probléma teljes hálózati gráfban való megoldásához szükséges menetek számára, csak megállási hibákat feltételezve. Ezeket a korlátokat 3 paraméter függvényében adjuk meg: a folyamatok n száma, a megengedett hibák f száma és a megengedett döntési értékek k száma.

7.1.1.. A feladat

A k-megegyezési problémánál az egyszerű megegyezési problémához hasonlóan feltesszük, hogy a hálózat egy n-csúcsú összefüggő, irányítatlan gráf, a hálózatban n folyamat $(P_1, P_2, \ldots, P_n)$ van és a folyamatok ismerik az egész gráfot. Minden folyamat egy rögzített V értékhalmazból vett bemenettel indul és végül ugyanebből az értékhalmazból választ egy kimenetet. (Most is feltesszük, hogy minden folyamatnak egy kezdőállapota van, amely minden bemeneti értéket tartalmaz.) Feltesszük, hogy legfeljebb f folyamat lehet hibás. Csak megállási hibákat engedünk meg. A szükséges feltételek a következők.

Megegyezés. A V értékhalmaznak van olyan k elemszámú W részhalmaza, hogy minden döntési érték W eleme.

Érvényesség. Minden folyamat döntési értéke valamely folyamat kezdeti értéke.

Befejezés. Ha egy folyamat nem hibás, akkor végül választ egy értéket.

A megegyezési feltétel a közönséges megegyezési problémánál használt megegyezési feltétel természetes általánosítása. Megjegyezzük, hogy megállási hibákra inkább a 6.1. alfejezet végefelé adott érvényességi feltételt használjuk, mint a hatodik fejezet nagyobb részében használt gyengébb feltételt; erre az erősebb feltételre a 7.1.3. szakaszban, az alsó korlát bizonyításakor lesz szükségünk. A közönséges megegyezési probléma az erősebb érvényességi feltétellel pontosan a k-megegyezési probléma $k = 1$ esete.

Ebben a fejezetben csak teljes gráfokra vonatkozó eredményeket tárgyalunk. Feltesszük, hogy a V halmazon teljes rendezés van értelmezve.

A 6.2.1. szakaszhoz hasonlóan azt mondjuk, hogy egy folyamat *aktív* r $(0 \leq r)$ menet után, ha nem hibásodik meg az r-edik menet végéig.

7.1.2.. Egy algoritmus

Most egy nagyon egyszerű algoritmust mutatunk be, melynek neve MinTerjed. Valójában ez pontosan a 6-9. gyakorlatban vázolt algoritmus, de kevesebb menetre van szüksége. A 6-9. gyakorlatban azt állítottuk, hogy ha ez az algoritmus $f + 1$ menetet fut, akkor garantálja a közönséges megállási megegyezést. Belátjuk, hogy a k-megegyezést akkor is biztosítani tudja, ha csak $\lfloor \frac{f}{k} \rfloor + 1$ menetet fut. Egyszerűen fogalmazva ez azt jelenti, hogy ha egy helyett k döntési értéket engedünk meg, akkor a futási idő a k-ad részére csökken.

MinTerjed algoritmus (vázlatosan)
Minden folyamat kezel egy *min_érték* változót, melybe induláskor a saját kezdeti értéket teszi. A folyamatok az $\lfloor \frac{f}{k} \rfloor + 1$ menet mindegyikében szétszórják *min_érték*-eiket, azután minden folyamat a helyettesíti saját *min_érték*-ét a beérkezett *min_érték*-ek és a saját régi *min_érték*-ének minimumával. A döntési érték a végső *min_érték*.

A kód a következő. (Szerkezetét hasonlítsuk össze a 6.2.1. szakaszban szereplő HalmazTerjed szerkezetével.)

MinTerjed algoritmus (formálisan)

Az üzeneti ábécé V.
állapotok$_i$
menetek $\in \mathbb{N}$, kezdetben 0
döntés $\in V \cup \{$*ismeretlen*$\}$, kezdetben *ismeretlen*
min_érték $\in V$, kezdetben P_i kezdeti értékét veszi fel

üzenetek$_i$
if *menetek* $\leq \lfloor \frac{f}{k} \rfloor$ **then** küldjük el *min_érték*-et minden más folyamatnak
átmenetek$_i$
menetek := *menetek* + 1
legyen m_j a P_j folyamattól kapott üzenet minden olyan folyamatra,
amelytől érkezett üzenet
min_érték := $\min(\{$*min_érték*$\} \cup \{m_j : j \neq i\})$
if *menetek* $= \lfloor \frac{f}{k} + 1 \rfloor$ **then** *döntés* := *min_érték*

Belátjuk az algoritmus helyességét; a bizonyítás hasonló ahhoz, ahogyan a 6.2.1. szakaszban a HalmazTerjed algoritmus helyességét beláttuk. Legyen $M(r)$ az aktív folyamatok *min_érték*-einek halmaza r menet után. Először azt látjuk be, hogy az egymást követő időpontokban az $M(r)$ halmaz csak csökkenhet.

7.1. lemma . *Ha* $1 \leq r \leq \lfloor \frac{f}{k} + 1 \rfloor$, *akkor* $M(r) \subseteq M(r-1)$.

Bizonyítás. Tegyük fel, hogy $m \in M(r)$. Akkor m egy olyan P_i folyamat *min_érték*$_i$ értéke, amely r menet után aktív. Ekkor vagy már az r-edik menet előtt is $m =$ *min_érték*$_i$ volt, vagy m éppen az r-edik menetben érkezik P_i-hez, mondjuk P_j-től. De ebben az esetben $r-1$ menet után *min_érték*$_j = m$ és P_j aktív $r-1$ menet után, mivel üzenetet küld az r-edik menetben. Ebből következik, hogy $m \in M(r-1)$. □

7.2. lemma . *Legyen* $d \in \mathbb{N}^+$. *Ha az* r*-edik* $(1 \leq r \leq \lfloor \frac{f}{k} \rfloor + 1)$ *menetben legfeljebb* $d-1$ *folyamat hibázik, akkor* $|M(r)| \leq d$, *azaz az* r*-edik menet után aktív az folyamatoknak legfeljebb* d *különböző min_érték-e van.*

Bizonyítás. Ellentmondás elérése érdekében tegyük fel, hogy r-edik menet alatt legfeljebb $d-1$ folyamat hibázott, és $M(r) > d$. Legyen $M(r)$ legnagyobb eleme m, és legyen $m' \neq m$ az $M(r)$ halmaz tetszőleges másik eleme. Ekkor a 7.1. lemma szerint $m' \in M(r-1)$. Legyen P_i tetszőleges aktív folyamat, melyre az $(r-1)$-edik menet után *min_érték*$_i = m'$. Ha P_i nem hibázik az r-edik menetben, akkor az r-edik menetben minden folyamat kap egy m'-t tartalmazó üzenetet P_i-től. Ez azonban nem fordulhat elő, mivel r menet után az egyik aktív folyamat *min_érték*-e m-mel egyenlő, és $m > m'$. Ebből következik, hogy P_i hibázik az r-edik menetben.

Az m' értéket azonban tetszőlegesen választottuk úgy, hogy az $M(r)$ halmaznak a legnagyobb elemtől különböző eleme legyen. Ezért $M(r)$ minden $m' \neq m$ eleméhez van olyan aktív folyamat, melynek *min_érték*-e $r-1$ menet után $m' \neq m$, és hibázik az r-edik menetben. Feltételünk szerint az r-edik menetben legfeljebb $d-1$ folyamat hibázik, ezért $M(r)$-nek legfeljebb $d-1$ olyan eleme lehet, amely m-től különbözik. Ezért $|M(r)| \leq d$, ami ellentmondás. □

Most bebizonyíthatjuk a fő helyességi tételt.

7.3. tétel . *A* MinTerjed *algoritmus a megállási hiba modellre megoldja a k-megegyezési problémát.*

Bizonyítás. A befejezés és az érvényesség közvetlenül adódik. Most bebizonyítjuk az új megegyezési feltétel teljesülését. Annak érdekében, hogy ellentmondásra jussunk, tegyük fel, hogy a különböző döntési értékek száma k-nál nagyobb egy olyan végrehajtási sorozatban, amelyben legfeljebb f hiba van. Akkor az aktív folyamatok *min_érték*-einek száma $\lfloor \frac{f}{k} \rfloor + 1$ menet után legalább $k+1$, azaz $|M(\lfloor \frac{f}{k} \rfloor + 1)| \geq k+1$. A 7.1. lemma szerint $|M(r)| \geq k+1$ minden r-re $(0 \leq r \leq \lfloor \frac{f}{k} \rfloor + 1)$. Ekkor a 7.2. lemmából következik, hogy az r-edik $(1 \leq r \leq \lfloor \frac{f}{k} \rfloor + 1)$ menetben legalább k folyamat hibázik. Ez azt eredményezi, hogy a hibák összes száma legalább $(\lfloor \frac{f}{k} \rfloor + 1)k$. Ez azonban határozottan nagyobb, mint f, ami ellentmondás. □

Bonyolultságelemzés. A menetek száma $\lfloor \frac{f}{k} \rfloor + 1$. Az üzenetek száma legfeljebb $(\lfloor \frac{f}{k} \rfloor + 1)n^2$, és az üzenetekben lévő bitek száma legfeljebb $\left(\lfloor \frac{f}{k} \rfloor + 1\right) n^2 b$, ahol b egy felső korlát a V halmaz egy elemének ábrázolásához szükséges bitek számára.

7.1.3.. Alsó korlát*

Ebben a szakaszban megmutatjuk, hogy az $\lfloor \frac{f}{k} \rfloor + 1$ korlát éles: ugyanis belátjuk, hogy a $|V| \geq k+1$ feltétel mellett egyúttal alsó korlát is. Ez az eredmény pontosan megadja azt a gyorsítást, melyet azzal érhetünk el, hogy egyetlen kimeneti érték helyett k értéket engedünk meg: az idő k-ad részére csökken. Amint az várható, a bizonyítás alapgondolatai a 6.33. tételben a közönséges megegyezésre vonatkozó alsó korlát bizonyításából származnak, de a bizonyítás maga bonyolultabb és érdekesebb. Ezek az ötletek elvezetnek bennünket az algebrai topológia birodalmába.

A fejezet hátralévő részében A olyan algoritmus, amely n folyamatra megoldja a k-megegyezési problémát, ha legfeljebb f folyamat követ el megállási hibát. Feltesszük, hogy az A algoritmus $r < \lfloor \frac{f}{k} \rfloor + 1$ menetben megáll; ekkor $r \leq \lfloor \frac{f}{k} \rfloor$. Ellentmondás elérése érdekében még azt is feltesszük, hogy $n \geq f+k+1$, ami azt jelenti, hogy legalább $k+1$ folyamat soha nem hibázik.

Az általánosság megszorítása nélkül feltehetjük, hogy a folyamatok pontosan az r-edik menet végén döntenek, és azonnal megállnak. Azt is feltesszük, hogy minden folyamat minden másik folyamatnak üzenetet küld a k-adik $(1 \leq k \leq r)$ menetben (amíg meg nem hibásodik). Végül feltesszük, hogy a V értékhalmaz

pontosan $k+1$ elemet $(0,\ 1,\ \ldots, k)$ tartalmaz, mivel mindezekre szükségünk van az ellentmondás eléréséhez.

Az ellentmondást úgy kapjuk meg, hogy megmutatjuk, A végrehajtási sorozatainak egyikében, melyben legfeljebb f folyamat hibásodhat meg, van $k+1$ olyan folyamat, amelyek $k+1$ különböző értéket választanak és ezzel megsértik a k-megegyezést.

Áttekintés. A 6.33. tétel bizonyítása szerint $f+1$ menet alsó korlát a közönséges megállási megegyezésre. A bizonyítás *láncérvelést* alkalmaz, amelyben végrehajtási sorozatok olyan láncát állítjuk elő, amely a csak 0 döntést megengedő végrehajtási sorozattal kezdődik, és a csak 1 döntést megengedő végrehajtási sorozattal végződik. Szeretnénk ezt a bizonyítást k más értékeire is kiterjeszteni. Sajnos, a k-megegyezési problémában – a közönséges megegyezési problémától eltérően – az adott végrehajtási sorozatban előforduló döntések nem határozzák meg a szorosan kapcsolódó végrehajtási sorozatokban szereplő döntéseket. Például, ha a közönséges megegyezési probléma megoldási algoritmus α és α' végrehajtási sorozatai megkülönböztethetetlenek a hibátlan P_i folyamat számára, akkor nemcsak P_i-nek kell α-ban és α'-ben ugyanazt a döntést hoznia, hanem az összes hibátlan folyamatnak is ugyanezt a döntést kell hoznia. A k-megegyezési algoritmusra az igaz, hogy ha α és α' megkülönböztethetetlenek P_i számára, akkor P_i döntései a két végrehajtási sorozatban azonosak, de a többi folyamat döntése meghatározatlan. Ha α és α' $n-1$ folyamat számára megkülönböztethetetlenek, a fennmaradó folyamat döntése még akkor is meghatározatlan.

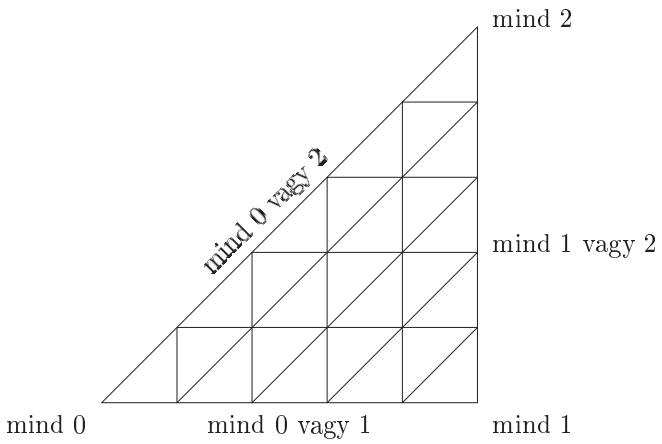
Kulcs gondolatunk az, hogy inkább egy k-dimenziós végrehajtási sorozathalmazt készítünk, mint egy egydimenziós láncot. Ennek a halmaznak a szomszédos végrehajtási sorozatai a kijelölt folyamatok számára megkülönböztethetetlenek. A végrehajtási sorozatok kezelésére felhasznált k-dimenziós alakzatot *Bermuda-háromszögnek* nevezzük (mivel bármely feltételezett k-megegyezési algoritmus eltűnik valahol a belsejében).

7.1.1. példa. Bermuda-háromszög

A 7.1. ábra egy Bermuda-háromszöget mutat a $k=2$ esetben. Az ábra egy nagy háromszöget mutat, amelyet kisebb háromszögekre bontottunk.

Ha $k>2$, akkor a háromszög k-dimenziós változatára van szükségünk. Szerencsére ilyen általánosítás már ismert az algebrai topológiában: k-dimenziós *szimplex* a neve. Például egy egydimenziós szimplex egy él, egy kétdimenziós szimplex egy háromszög, és egy háromdimenziós szimplex egy tetraéder. (Háromnál több dimenzió esetében már sokkal nehezebb a szimplexeket elképzelni.)

Most tetszőleges k-ra induljunk ki a k-dimenziós euklideszi térben adott k-dimenziós szimplexből. A szimplex *rácspontokat* tartalmaz, amelyek az euklideszi tér egész koordinátájú pontjai. A *k-dimenziós B Bermuda-háromszöget* úgy kapjuk, hogy ezt a szimplexet a rácspontok segítségével felbontjuk, és így kis k-dimenziós kis szimplexek halmazát kapjuk.



7.1.. ábra. Bermuda-háromszög a $k = 2$ esetben.

A bizonyítás során először B minden csúcsához (rácspontjához) hozzárendelünk egy végrehajtási sorozatot. A B Bermuda-háromszög $k + 1$ sarokcsúcsához olyan végrehajtási sorozatokat rendelünk, amelyekben a folyamatok a $\{0, \ldots, k\}$ halmaz ugyanazon elemét kapják bemenetként, és amelyekben nincsenek hibák. Például a $k = 2$ esetben a bal alsó sarokcsúcshoz rendelt végrehajtási sorozatban minden folyamat bemenete 0, a jobb alsó csúcshoz rendelt végrehajtási sorozatban minden bemenet 1, és a jobb felső csúcshoz rendeltben minden bemenet 2 (lásd a 7.1. ábrát). Továbbá, B tetszőleges oldalán (tetszőleges dimenziószám esetében) lévő bármely x csúcshoz rendelt végrehajtási sorozatban csak azok a bemenetek fordulhatnak elő, amelyek előfordulnak az oldallap csúcsaihoz rendelt végrehajtási sorozatban. Például a $k = 2$ esetben az alsó élen lévő csúcsokhoz rendelt végrehajtási sorozatokban lévő bemenetek mindegyike eleme a $\{0, 1\}$ halmaznak.

Ezután B minden csúcsához hozzárendeljük egy olyan folyamat indexét, amely az adott csúcshoz rendelt végrehajtási sorozatban nem hibás. Ezt a hozzárendelést úgy végezzük, hogy minden T kis szimplexhez egyértelműen legyen egy olyan α végrehajtási sorozat, amelyben legfeljebb f hiba van, és amely az alábbi értelemben kompatibilis T sarokpontjaival.

1. A T sarokcsúcsaihoz rendelt folyamatok α-ban nem hibásak.
2. Ha T valamelyik sarokcsúcsához az α' végrehajtási sorozatot és a P_i folyamatot rendeltük, akkor α és α' megkülönböztethetetlenek P_i számára.

A végrehajtási sorozatoknak és folyamatoknak ez a hozzárendelése B csúcsaihoz rendelkezik néhány szép tulajdonsággal. Tegyük fel, hogy α-t és P_i-t hozzárendeljük az x csúcshoz. Ha x a B-nek egy sarokcsúcsa, akkor α-ban minden folyamat ugyanazzal a bemenettel kezd, ezért az érvényességi feltétel szerint P_i-nek α-ban ezt az értéket kell választania. Ha x a B-nek egy külső élén van, akkor minden folyamat az adott él két végpontjához tartozó két érték valamelyikével kezd; az érvényességi feltételből következik, hogy P_i-nek ezen két érték valame-

lyikét kell választania. Általánosan fogalmazva, ha x a B-nek egy (tetszőleges dimenziójú) oldalán helyezkedik el, akkor α-ban minden folyamat az adott oldal valamelyik sarokpontjához tartozó értékkel kezd; ezért P_i az érvényességi feltétel szerint ezen értékek közül választ. Végül, ha x B-nek belső pontja, akkor P_i a $k+1$ érték bármelyikét választhatja.

A végrehajtási sorozatoknak és folyamatoknak a most leírt módon történő csúcsokhoz rendeléséhez szükség van arra, hogy a menetek r száma minden végrehajtási sorozatban legfeljebb $\lfloor \frac{f}{k} \rfloor$ legyen, azaz teljesüljön, hogy $f \geq rk$. Ez azért van így, mert a végrehajtási sorozatokat a 6.33. tétel bizonyításában használt láncérvelés k-dimenziós általánosításával rendeljük a csúcsokhoz.

Miután a végrehajtási sorozatokat és indexeket hozzárendeltük a csúcsokhoz, a csúcsokat a $\{0, 1, \ldots, k\}$ halmazból vett „színekkel" kiszínezzük. Nevezetesen, ha az x csúcshoz az α végrehajtási sorozatot és a P_i folyamatot rendeltük, a csúcsot azzal az értékkel színezzük, amelyet P_i választ α-ban. Ennek a színezésnek a következő tulajdonságai vannak.

1. A B háromszög $k+1$ sarokcsúcsának különbözők a színei.
2. A B háromszög külső élén fekvő pontok színe megegyezik az él valamelyik végpontjának színével.
3. Általánosabban, a B bármely (tetszőleges dimenziójú) oldalán fekvő csúcsainak színe megegyezik az oldal valamelyik sarokcsúcsának színével.

A k-szimplexeknek pontosan ezekkel a tulajdonságokkal rendelkező színezését az algebrai topológiában *Sperner-színezésnek* hívják.

Itt alkalmazni fogunk egy nevezetes kombinatorikai eredményt, melyet először 1928-ban bizonyítottak be. A Sperner-lemma szerint egy kis szimplexekre bontott k-dimenziós szimplex Sperner-színezésének tartalmaznia kell legalább egy olyan kis szimplexet, melynek $k+1$ csúcsa $k+1$ különböző színre van színezve. Esetünkben ez a szimplex megfelel egy egy olyan végrehajtási sorozatnak, amelyben legfeljebb f hiba van, és amelyben $k+1$ folyamat $k+1$ különböző értéket választ. Ez azonban ellentmond a k-megegyezési probléma megegyezési feltételének.

Ebből következik, hogy a remélt algoritmus nem létezik, azaz a k-megegyezési probléma megoldására nincs olyan algoritmus, amely f hiba esetében $r \leq \lfloor \frac{f}{k} \rfloor$ menet után megállna. Az alfejezet hátralévő része további részleteket tartalmaz.

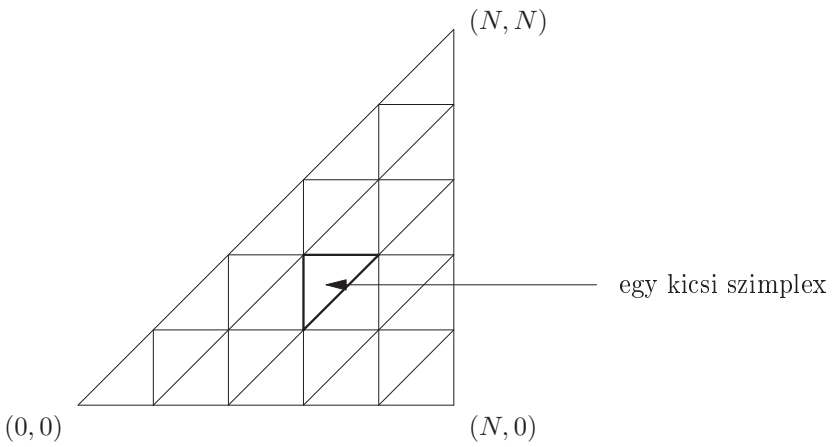
Definíciók. Felhasználjuk a kommunikációs mintának a 6.7. alfejezetben megadott meghatározását. Újra definiáljuk a *jó* kommunikációs mintát: olyan kommunikációs minta, amelyben $k \leq r$ minden (i, j, k) hármasra, és a hiányzó hármasok konzisztensek a megállási hiba modellel. (Azaz a jó kommunikációs mintának a 6.7. alfejezetben megadott definíciójából az első két feltételt használjuk – csak most a menetek számának felső korlátja f helyett r. Egyelőre nem korlátozzuk a hibák számát.) A jó kommunikációs minta új definícióját használva a *futam* és adott ρ-ra a *végre*(ρ) definíciók ugyanazok, mint a 6.7. alfejezetben. Azt mondjuk, hogy a P_i folyamat egy futam t-edik menete után *csendes*, ha a $(t+1)$-edik vagy nagyobb sorszámú menetekben nem küld üzenetet.

Bermuda-háromszög. A k-dimenziós euklideszi térben értelmezett k-szimplexszel kezdjük, melynek sarokcsúcsai a k hosszúságú $(0, \ldots, 0)$, $(N, 0, \ldots, 0)$, $(N, N, 0, \ldots, 0)$, …, $(N, \ldots, N)$ vektorok, ahol N egy nagyon

nagy egész szám. A B Bermuda-háromszög ez a szimplex a kis szimplexekre való alábbi felbontásával együtt. B csúcsai a szimplexekben lévő rácspontok, azaz az $x = (x_1, \ldots, x_k)$ alakú csúcsok, ahol a vektorok koordinátái 0 és N közötti egész számok, melyekre $x_1 \geq x_2 \geq \ldots \geq x_k$. A kis szimplexeket a következőképpen definiáljuk: válasszunk ki egy tetszőleges rácspontot, és minden dimenzióban minden pozitív irányban tegyünk meg egy lépést, tetszőleges sorrendben. A bejárás során érintett $k+1$ csúcs megadja egy kis szimplex csúcsait.

7.1.2. példa. A Bermuda-háromszög csúcsainak koordinátái

A 7.2. ábra egy kétdimenziós Bermuda-háromszöget ábrázol.



7.2.. ábra. Kétdimenziós Bermuda-háromszög.

***B* címkézése végrehajtási sorozatokkal és futamokkal.** Ebben a részben leírjuk, hogyan rendelünk végrehajtási sorozatokat B csúcsaihoz (azaz a csúcsokat „megcímkézzük" a végrehajtási sorozatokkal. Ezt úgy végezzük el, hogy először a futamokban lévő bizonyos (i, t) (folyamat, menetszám) párokhoz *jeleket* rendelünk. Ezek a jelek úgy értelmezhetőek, hogy megengedik a P_i folyamatnak, hogy a t-edik menetben vagy azután hibázzon. Ugyanahhoz az (i, t) párhoz több jel is rendelhető.

Pontosabban, az *l-futamot* minden $l > 0$ értékre úgy definiáljuk, hogy az minden t $(1 \leq t \leq r)$ értékre pontosan l jellel úgy van kiegészítve, hogy ha a P_i folyamat a t-edik menetben hibázik, akkor van olyan jel, amely egy (i, t') párhoz $(t') \leq t)$ van hozzárendelve. Ezek szerint egy l-futam pontosan lr jelet tartalmaz. Csak az $l = 1$ és az $l = r$ eset, azaz az 1-futamok és az r-futamok érdekelnek bennünket. A *hibamentes l-futamot* úgy definiáljuk, hogy az olyan l-futam, amelyben nincs hiba, és amelyben minden jel $(1, t)$ alakú párhoz van rendelve (azaz csak a P_1 folyamat hibázhat).

Mivel minden kiegészített futamot futamból állítottunk elő, minden kiegészített futam nyilvánvaló módon alakítható át végrehajtási sorozattá. A *végre*(ρ) jelölést, amelyet korábban futamokra vezettünk be, kiterjesztjük arra az esetre, amikor ρ kiterjesztett futam. B csúcsainak végrehajtási sorozatokkal való címkézése most k-futamokkal történő címkézést jelent.

Négy műveletet definiálunk l-futamokra, melyek mindegyike csak kis változásokat okoz. Ezek a műveletek csak egyetlen hármast tudnak eltávolítani vagy hozzáadni, vagy egyetlen folyamat bemeneti értékét tudják módosítani, vagy szomszédos indexű folyamatok között tudnak egy jelet mozgatni ugyanabban a menetben. Ezek a műveletek nagyon hasonlítanak a 6.33. tétel bizonyításában alkalmazottakhoz. Ezeket a műveleteket a következőképpen definiáljuk.

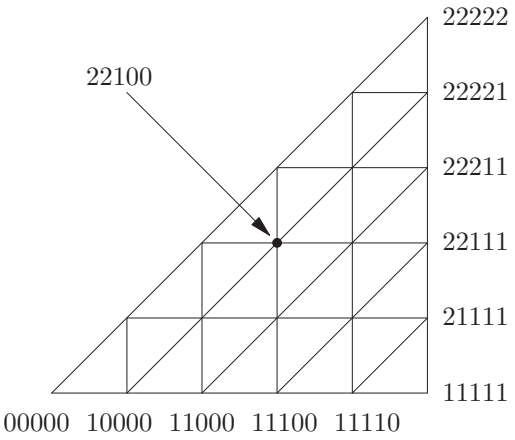
1. `eltávolít`(i,j,t), ahol i és j folyamatindexek és t menetszám, $1 \leq t \leq r$. Ez a művelet eltávolítja az (i,j,t) hármast (ami azt jelenti, hogy P_i a t-edik menetben P_j-nek üzenetet küldött), ha van ilyen hármas, egyébként nincs hatása. Csak akkor alkalmazható, ha t menet után P_i és P_j is csendes, és van olyan (i,t') $(t' \leq t)$ pár, amelyhez jel van kapcsolva.
2. `hozzáad`(i,j,t). Ez a művelet hozzáadja az (i,j,t) hármast, ha még nincs hozzáadva a futamhoz, egyébként nincs hatása. Csak akkor alkalmazható, ha P_i és P_j mindketten csendesek t menet után, és P_i aktív $t-1$ menet után.
3. `változtat`(i,v). Ez a művelet P_i bemeneti értékét v-re változtatja, és nincs hatása, ha a bemeneti érték már v. Csak akkor alkalmazható, ha P_i csendes 0 menet után és az $(i,1)$ pár rendelkezik jellel.
4. `mozgat`(i,j,t). Ez a művelet a (i,t) pártól a (j,t) párhoz mozgat egy jelet, ahol j vagy $i+1$ vagy $i-1$. Csak akkor alkalmazható, ha (i,t)-nél van jel és minden hibának engedélye van más jelektől.

A definíciókból következik, hogy ha ezeket a műveleteket l-futamra alkalmazzuk, akkor az eredmény is l-futam lesz.

Definiáljuk minden $v \in \{1,\ldots,k\}$ értékre az `eltávolít, hozzáad, változtat` és `mozgat` műveletek *sorozat*(v) sorozatát, amely bármely ρ hibamentes 1-futamra alkalmazható, és azt olyan hibamentes 1-futammá alakítja, melyben minden folyamat bemenete v. Valójában minden ρ hibamentes 1-futamra ugyanaz a *sorozat*(v) alkalmazható. Az átalakítás a 6.33. tétel bizonyításában alkalmazott módszerekkel végezhető el; a fő különbség a hibákra engedélyt adó jelek tényleges mozgása. Ebben a konstrukcióban a folyamatok bemenetei egyenként változnak v-re, a P_1 folyamattal kezdve. A korábbiakhoz hasonlóan ez a konstrukció r menetben r hibát használ fel.

Kiderül, hogy a *sorozat*(v) sorozatok megkonstruálhatók úgy, hogy különböző v-re izomorfak – azaz v választásától eltekintve azonosak. Végül definiálhatjuk a B méretének meghatározásánál használt N paramétert: N egyszerűen a *sorozat*(v) sorozat hossza (tetszőleges v-re).

B csúcsainak címkézésére több *sorozat*(v) sorozatot fogunk használni. Emlékeztetünk arra, hogy az értékkészlet elemei $0,1,\ldots,k$. Minden $v \in \{0,1,\ldots,k\}$ értékre legyen τ_v az a hibamentes 1-futam, amelyben minden folyamat kezdeti értéke egyenlő v-vel. Minden *sorozat*(v) $(1 \leq v \leq k)$ sorozatot arra használunk, hogy a τ_{v-1} hibamentes 1-futamhoz 1-futamok olyan sorozatát állítsuk elő, ame-



7.3.. ábra. Bermuda-háromszög címkézése k-futamnál.

lyeket a B-nek a v-edik dimenziós élei (a v-tengelyek) mentén lévő csúcsok *előzetes címkézésére* használhatunk. Ekkor a B x csúcsához rendelendő k-futamot annak a k darab 1-futamnak az „összefésülésével" kapjuk, amelyek x k-tengelyekre való vetületének előzetes címkéi.

7.1.3. példa. A Bermuda-háromszög címkézése k-futamokkal

Ennek az összefésülésnek a megértését segíti a 7.3. ábra, amelyen a $k = 2$ (és így $V = \{0,\ 1,\ 2\}$) és $n = 5$.

A diagram csak azokat a csúcsokat tartalmazza, amelyeket hibamentes k-futamokkal címkéztünk. Ehhez csak az ábrázolt csúcsokhoz tartozó bemeneti értékek vektorára van szükségünk. A B háromszög csúcsait címkéző futam minden sarokcsúcsban olyan hibamentes k-futam, melynek elemei azonosak és a bal alsó sarokcsúcsnál nullák, a jobb alsó sarokcsúcsnál egyesek és a felső csúcsnál kettesek. A vízszintes tengelyen lévő láncot *sorozat*(1) segítségével állítjuk elő, és a csupa nullából álló vektorból a csupa egyesből álló vektorhoz vezet, míg a függőleges tengelyen lévő láncot a *sorozat*(2) segítségével állítottuk elő és ez a lánc a csupa egyesből álló vektortól a csupa kettesből álló vektorhoz vezet.

Figyeljük meg a B-ben megjelenő bemenetek mintáját. A vízszintes tengelyen a folyamatok bemenetei egyenként 0-ról 1-re változnak, a P_1 folyamattal kezdve. A függőleges tengely mentén a folyamatok bemenetei egyenként 1-ről kettőre változnak, a P_1 folyamattól kezdve. B belsejében a változások mindkét irányban egyidejűleg mennek végbe. például tekintsük a nyíllal megjelölt 22100 csúcsot. A vízszintes és a függőleges tengelyen lévő vetületeit címkéző vektorok 11100 és 22111.

A 22100 vektor úgy változtatható 22111-re, hogy az utolsó két folyamat bemenetét 0-ról 1-re változtatjuk, miközben B-ben vízszintesen mozgunk. Hasonlóképpen a 11100 vektor úgy változtatható 22100-ra, hogy az első két folyamat bemenetét 1-ről 2-re változtatjuk, miközben B-ben függőlegesen mozgunk. A B csúcsait címkéző vektor minden esetben a $\{0, 1, 2\}$ halmaz elemeit tartalmazza, nem növekvő sorrendben.

Most megadjuk az összefésülés formális definícióját. Az 1-futamok $\sigma_1, \ldots, \sigma_k$ 1-futamokból álló sorozatának *összefésültje* a következőképpen definiált ρ k-futam.

1. A P_i folyamat bemeneti értéke ρ-ban v, ahol v az $\{1, \ldots, k\}$ értékek közül a legnagyobb olyan érték, amelyre P_i bemeneti értéke σ_v-ben v, vagy pedig nulla, ha ilyen érték nem létezik.
2. Az (i, j, k) hármas pontosan akkor van benne ρ-ban, ha minden σ_v-ben $(1 \leq v \leq k)$ benne van.
3. Az (i, t) párhoz ρ-ban rendelt jelek száma az összes σ_v-ben lévő (i, t) párhoz rendelt jelek számának összege.

Az első feltétellel kapcsolatban vizsgáljuk meg újra azt a módot, ahogyan az összefésülési műveletet alkalmazzuk. Minden σ_v-t úgy kapunk meg, hogy *sorozat*(v) bizonyos előtagját alkalmazzuk τ_{v-1}-re. Ennek a sorozatnak bizonyos pontján a P_i folyamat bemeneti értéke $v-1$-ről v-re változik. Ha ez már megtörtént σ_v-ben, akkor mondjuk azt, hogy a P_i folyamat „átment" a v-edik dimenzióba. Az első feltétel éppen a legnagyobb v-t választja (ha van ilyen), amelyre igaz, hogy P_i átment a v-edik dimenzióba.

A második feltétel azt mondja, hogy egy üzenet akkor és csak akkor hiányzik a ρ új futamból, ha hiányzik bármely összefésülendő σ_v futamból. A harmadik feltétel összeszámolja a jeleket. Nem nehéz belátni, hogy 1-futamok sorozatának összefésültje valójában k-futam.

Rakjuk össze az elemeket és definiáljuk B k-futamokkal való címkézését. Legyen $x_1, \ldots, x_k$ B tetszőleges csúcsa. Minden $v \in \{1, \ldots k\}$ értékre legyen σ_v az az 1-futam, amelyet úgy kapunk, hogy *sorozat*(v) első x_v műveletét alkalmazzuk τ_{v-1}-re. Akkor az x csúcsot címkéző k-futamot a $\sigma_1, \ldots, \sigma_k$ összefésülésével kapjuk. Megjegyezzük, hogy az összefésült futamban legfeljebb $rk \leq f$ jel van, és ezért legfeljebb f hiba. A bizonyítás hátralévő részére rögzítjük B k-futamokkal (és végrehajtási sorozatokkal) való címkézését.

Ezt a szakaszt azzal fejezzük be, hogy megmutatjuk azt a szoros kapcsolatot, amely B bármely T kis szimplexének csúcsait címkéző k-futamok között van. Legyenek $y_0, \ldots, y_k$ T csúcsai abban a sorrendben, amelyet a T-t előállító „végrehajtás" meghatároz (amint azt a Bermuda-háromszög definíciójában leírtuk). Legyenek $\rho_0, \ldots, \rho_k$ az ezeket a csúcsokat címkéző, megfelelő k-futamok.

Az első lemma azt mondja ki, hogy minden folyamat, amely ezen k-futamok valamelyikében hibás, legalább egy jellel rendelkezik ezen futamok mindegyikében.

7.4. lemma . *Ha a P_i folyamat hibás egy ρ_v $(0 \leq v \leq k)$ futamban, akkor a P_i folyamathoz minden ρ_v futamban tartozik jel.*

Bizonyításvázlat. Az állítás igaz, mivel a változások minden *sorozat*-ban fokozatosak, és mivel egy jel mozgása és egy hármas eltávolítása két külön lépésben valósul meg. A részletes bizonyítást meghagyjuk gyakorlatnak (lásd 7-4. gyakorlat). □

A második lemma korlátot tartalmaz a futamokban előforduló összes hiba számára.

7.5. lemma . *Legyen F_v ($v \in \{0, \ldots, k\}$) azoknak a folyamatoknak a halmaza, amelyek hibásak ρ_v-ben. Legyen $F = \cup_v F_v$. Ekkor $|F| \leq rk \leq f$.*

Bizonyítás. Meghagyjuk gyakorlatnak (lásd 7-5. gyakorlat). A bizonyítás felhasználja a 7.4. lemmát. □

Végül T csúcsainak folyamatindexekkel való címkézését vizsgáljuk meg. T egy *helyi folyamatcímkézése* az $i_0, \ldots, i_k$ indexű folyamatok hozzárendelése az $y_0, \ldots, y_k$ csúcsokhoz oly módon, hogy minden v-re teljesül az, hogy az i_v indexű folyamatnak nincs jele ρ_v-ben. A T csúcsait címkéző k-futamok fontos tulajdonsága, hogy ha létezik T helyi folyamatcímkézése, akkor T egyetlen végrehajtási sorozattal konzisztens.

7.6. lemma . *Legyen $i_0, \ldots, i_k$ a T szimplex egy helyi folyamatcímkézése. Akkor van olyan, legfeljebb f hibát tartalmazó ρ futam, hogy minden v-re teljesül, hogy P_{i_v} nem hibás ρ-ban, továbbá végrehajtás(ρ_v) és végrehajtás(ρ) megkülönböztethetetlenek a P_i folyamat számára.*

Bizonyításvázlat. ρ-t a következőképpen definiáljuk. A kezdeti értéket minden ρ-beli P_i folyamatra úgy definiáljuk, hogy legyen egyenlő P_i kezdeti értékével a ρ_v-k valamelyikében. Az (i, j, t) $(1 \leq t \leq r-1)$ hármast pontosan akkor vesszük bele ρ-ba, ha minden ρ_v-ben benne van. Hasonlóképpen az (i, j, r) hármast pontosan akkor vesszük bele, ahol a j fogadó az i_v indexű folyamatok mindegyikétől különbözik, ha minden ρ_v-ben benne van. Végül az olyan (i, j, r) hármast, ahol $j = i_v$ (egy speciális v-re) pontosan akkor vesszük bele, ha benne van ρ_v-ben (ugyanarra a v-re).

Meghagyjuk gyakorlatnak (lásd 7-6. gyakorlat) annak megmutatását, hogy ρ minden szükséges tulajdonsággal rendelkezik: azaz valóban futam, legfeljebb f hibát tartalmaz és az i_v indexű folyamat minden v-re hibátlan ρ-ban, valamint hogy *végrehajtás*(ρ_v) és *végrehajtás*(ρ) megkülönböztethetetlenek az i_v indexű folyamat számára. A bizonyítás a hibák számának korlátjához a 7.5. lemmát használja fel. □

***B* címkézése folyamatindexekkel.** Emlékeztetünk arra, hogy célunk folyamatindexek hozzárendelése B csúcsaihoz úgy, hogy minden T kis szimplexre legyen olyan végrehajtási sorozat, amely kompatibilis a T csúcsait címkéző végrehajtási sorozatokkal és folyamatokkal. A 7.6. lemma módszert ad ennek megoldására: B minden csúcsához kiválasztunk egy folyamatot úgy, hogy a megfelelő k-futamban nincs jel és a kis szimplexek csúcsaihoz választott folyamatok mind különbözők. Ekkor a 7.6. lemmából minden kis szimplexre adódik a szükséges kompatibilitási feltétel.

B globális folyamatcímkézését folyamatoknak B csúcsaihoz való olyan hozzárendeléseként definiáljuk, hogy a tetszőleges x csúcshoz rendelt folyamatnak nincs jele az x-et címkéző k-futamban, és minden T kis szimplexre fennáll, hogy a T csúcsaihoz rendelt folyamatok különbözők. B globális folyamatcímkézése B minden kis szimplexére nézve helyi folyamatcímkézést eredményez.

Most elkészítjük B globális folyamatcímkézését. (Mivel ez pusztán technikai feladat, az első olvasáskor átugorhatjuk, és közvetlenül a 7.10. lemmával folytathatjuk.) A konstrukciót azzal kezdjük, hogy a B csúcsát címkéző ρ k-futamhoz hozzákapcsolunk egy *élő*(ρ) folyamathalmazt, majd minden *élő*(ρ) halmazból választunk egy folyamatot. A *élő*(ρ) halmazok rendelkeznek a következő tulajdonságokkal.

1. Minden *élő*(ρ) halmaz pontosan $n - rk$ folyamatot tartalmaz. (Mivel feltettük, hogy $n \geq f + k + 1$ és $f \geq rk$, ez azt jelenti, hogy minden *élő*(ρ) halmaz tartalmaz legalább $k + 1$ folyamatot.)
2. A *élő*(ρ) halmazban lévő folyamatokat azok közül választjuk, amelyek ρ-ban nem rendelkeznek jellel.
3. Ha ρ és ρ' B-ben ugyanannak a kis szimplexnek két csúcsát címkéző k-futamok és ha a $P_i \in$ *élő*(ρ) $\cap$ *élő*(ρ'), akkor P_i rangja a két halmazban azonos[1]

Rögzítsük ρ egyik k-futamát. Ez a futam pontosan rk jelet tartalmaz; legyen a *jelek* a folyamatindexeknek egy multihalmaza, amely megadja az egyes folyamatokhoz rendelt jelek számát. A *jelek* multihalmazt „kisimítjuk”, hogy megkapjuk az új *új_jelek* multihalmazt, amelyben ugyanannyi jel van, de ebben minden folyamathoz legfeljebb egy jel van hozzárendelve. Ugyancsak teljesül, hogy minden olyan folyamatnak, amelynek volt jele a *jelek* multihalmazban, lesz jele a *új_jelek* multihalmazban is. A SIMÍT eljárás a következőképpen működik.

SIMÍT eljárás (vázlatosan)
új_jelek := *jelek*
while *új_jelek*-ben van többszörös elem **do**
  válasszunk egy ilyen elemet, például i-t
  if van olyan $j < i$, melyre *új_jelek*$(j) = 0$ **then** mozgassunk egy jelet P_i-től a legnagyobb ilyen P_j-hez
  else mozgassunk el egy jelet P_i-től a legkisebb olyan $j > i$ indexű folyamathoz, melyre *új_jelek* $= 0$

Legyen $l(\rho)$ az olyan P_i folyamatok indexeinek halmaza, melyekre *új_jelek*$(i) = 0$.

Az könnyen belátható, hogy az így definiált *élő* halmaz rendelkezik az előbb említett tulajdonságok közül az első kettővel. A harmadik tulajdonság belátásához rögzítsünk egy T kis szimplexet, melynek csúcsai a szimplexet előállító bejárás sorrendjében legyenek y_0, y_1, ..., y_k, és legyenek ρ_0, ρ_1, ..., ρ_k a megfelelő k-futamok, amelyek ezeket a csúcsokat címkézik. Először jegyezzük meg azt, hogy amikor sorrendben haladunk T csúcsain, akkor ha a P_i folyamat kapott egy jelet, akkor később a bejárás során végig rendelkezni fog jellel.

[1] Egy i elem rangja az L teljesen rendezett halmazban az i-nél nem nagyobb elemek száma.

7.7. lemma . *Legyen $v < v' < v''$. Ha a P_i folyamatnak nincs jele ρ_v-ben, de van jele $\rho_{v'}$-ben, akkor P_i-nek van jele $\rho_{v''}$-ben.*

Bizonyítás. Meghagyjuk gyakorlatnak (lásd 7-7. gyakorlat). □

Most már be tudjuk bizonyítani az *élő* halmazok harmadik tulajdonságát.

7.8. lemma . *Ha $P_i \in$ élő$(\rho_v) \cap$ élő(ρ_w), akkor P_i rangja élő(ρ_v)-ben és élő(ρ_w)-ben ugyanaz.*

Bizonyítás. Az általánosság megszorítása nélkül feltehetjük, hogy $v < w$. Mivel $P_i \in$ élő(ρ_v) és $P_i \in$ élő(ρ_w), P_i-nek nincs jele sem ρ_v-ben, sem ρ_w-ben. Ekkor a 7.7. lemmából következik, hogy a $\rho_0, \ldots, \rho_w$ futamok egyikében sincs P_i-nek jele.

Mivel jelek elhelyezése szomszédos k-futamokban legfeljebb egy jelnek egy folyamattól szomszédos folyamathoz való mozgatásával különbözik, és mivel P_i-nek ezen futamok egyikében sincs jele, következik, hogy az i-nél kisebb indexű folyamatoknál lévő összes jel száma ugyanaz, mondjuk s, a $\rho_0, \ldots, \rho_w$ futamok mindegyikére. Mivel $i \in$ élő(ρ_v), ezért a SIMÍT eljárás működéséből következik, hogy $s < i$. (Ha $s \geq i$, akkor az i-nél kisebb indexű folyamatoknál induló jelek a SIMIT eljárásban „túlcsordulnak", azaz a P_i folyamatnál végeznek). Ezért biztosítva van az, hogy P_i rangja *élő*ρ_v-ben és *élő*ρ_w-ben ugyanaz az $i - s$ érték. □

Készen állunk arra, hogy B csúcsait megcímkézzük folyamatindexekkel. Legyen $x = (x_1, \ldots, x_k)$ B tetszőleges csúcsa, és legyen ρ a csúcs k-futama; választunk egy folyamatindexet az élő(ρ) halmazból. Nevezetesen, legyen *sík*$(x) = \sum_{i=1}^{k} x_i \pmod{(k+1)}$; x-et azzal a folyamattal címkézzük, amelynek rangja élő(ρ)-ban *sík*(x). Ezt a választást B következő tulajdonsága indokolja.

7.9. lemma . *Ha x és y ugyanannak a kis szimplexnek különböző csúcsai, akkor sík$(x) \neq$ sík(y).*

Most megfogalmazzuk azt az állítást, melyre szükségünk van.

7.10. lemma . *B folyamatindexekkel való, most ismertetett címkézése globális folyamatcímkézés.*

Bizonyítás. Mivel minden x csúcs indexét a élő(ρ) halmazból választottuk, ahol ρ a csúcshoz kapcsolt k-futam, ehhez az indexhez ρ-ban nem tartozhat jel. Bármely T rögzített kis szimplexre következik a 7.8. és 7.9. lemmákból, hogy a választott indexek különbözők. □

Most összegezzük mindazt, amit az előállított címkézésekről tudunk.

7.11. lemma . *B k-futamokkal és folyamatokkal való, adott címkézései rendelkeznek a következő tulajdonsággal. Minden olyan T kis szimplexre, amely a $\rho_0, \ldots, \rho_k$ futamcímkékkel és $i_0, \ldots, i_k$ folyamatcímkékkel van ellátva, létezik olyan ρ futam legfeljebb f hibával, hogy minden v-re teljesül az, hogy i_v nem hibás ρ-ban és végrehajtás(ρ_v) és végrehajtás(ρ) megkülönböztethetetlenek az i_v folyamat számára.*

Bizonyítás. Az állítás a 7.6. és 7.10. lemmákból következik. □

Sperner-lemma. Majdnem készen vagyunk. Csak az van hátra, hogy megfogalmazzuk a Sperner-lemmát (a Bermuda-háromszögre vonatkozó speciális esetben), és alkalmazzuk az ellentmondás eléréséhez. Ez megadja a k-megegyezés eléréséhez szükséges menetek számára vonatkozó alsó korlátot. *B Sperner-színezése* hozzárendeli egy $k+1$ színt tartalmazó színhalmaz valamelyik elemét B minden csúcsához a következőképpen.

1. B $k+1$ sarokcsúcsának színei mind különbözőek.
2. Bármely, a B külső élén fekvő csúcs színe az adott él valamelyik végpontjának színével egyezik meg.
3. Általánosan, egy külső oldal (bármelyik dimenzióban) bármely belső pontjának színe B szomszédos sarokcsúcsai valamelyikének színe.

A Sperner-színezésnek van egy figyelemre méltó tulajdonsága: van legalább egy olyan kis szimplex, melynek $k+1$ csúcsa $k+1$ különböző színnel van kiszínezve.

7.12. lemma . ***(Sperner-lemma B-re)*** *B bármely Sperner-színezésére van B-ben legalább egy olyan kis szimplex, melynek $k+1$ csúcsa $k+1$ különböző színnel van kiszínezve.*

Emlékeztetünk arra, hogy feltételezésünk szerint A olyan algoritmus a k-megegyezés megoldására, amely képes f hiba eltűrésére, és legfeljebb $\lfloor \frac{f}{k} \rfloor$ menet után megáll. B C_A színezését a következőképpen definiáljuk. Ha egy x csúcs a ρ futammal és a P_i folyamattal van címkézve, akkor x-et arra a színre színezzük, amelyet a P_i folyamat választ az A algoritmus *végrehajtás*(ρ) végrehajtási sorozatában.

7.13. lemma . *Ha A egy f hibát tűrő k-megegyezési algoritmus, amely $\lfloor \frac{f}{k} \rfloor$ menet alatt befejezik, akkor C_A a B háromszög Sperner-színezése.*

Bizonyítás. A k-megegyezés érvényességi feltétele segítségével történhet. □

Most már kimondhatjuk a fő tételt.

7.14. tétel . *Ha $n \geq f+k+1$, akkor a k-megegyezés megoldására nincs olyan algoritmus, amely f hiba esetében biztosítja, hogy minden hibátlan folyamat legfeljebb $\lfloor \frac{f}{k} \rfloor$ menet alatt döntsön.*

Bizonyítás. A 7.13. lemmából következik, hogy C_A Sperner-színezés, ezért a 7.12. Sperner-lemmából következik, hogy van olyan T kis szimplex, melynek minden csúcsa különböző színű a C_A szerint.

Tegyük fel, T k-futam címkéi $\rho_0, \ldots, \rho_k$ és folyamatcímkéi $i_0, \ldots, i_k$. C_A definíciója szerint ez azt jelenti, hogy a $k+1$ különböző i_v indexű folyamat megfelelő *végrehajtás*(ρ_v) végrehajtási sorozataiban mind a $k+1$ különböző döntés előfordult. A 7.11. lemmából következik, hogy van olyan ρ futam legfeljebb f hibával, hogy minden v-re teljesül az, hogy P_{i_v} nem hibás ρ-ban, továbbá *végrehajtás*(ρ_v) és *végrehajtás*(ρ) megkülönböztethetetlenek az i_v indexű folyamat számára. Ez azonban azt jelenti, hogy ρ-ban az $i_0, \ldots, i_k$ indexű folyamatok (számuk $k+1$) $k+1$ különböző értéket választanak, amivel megsértik a k-megegyezési probléma megegyezési feltételét. □

7.2.. Közelítő megegyezés

Ebben az alfejezetben a *közelítő megegyezés* problémáját bizánci hibák előfordulása mellett vizsgáljuk. Ebben a problémában a folyamatok valós bemenetekkel kezdenek, és feltesszük, hogy végül valós kimeneteket választanak. Üzeneteikben valós értékeket küldhetnek. Ezúttal nem pontos értékben kell megegyezniük, mint a közönséges megegyezési problémánál, hanem egymástól legfeljebb egy kis pozitív ϵ-nal eltérő értékekben. Pontosabban a következő feltételeknek kell teljesülniük.

Megegyezés. Bármely két hibátlanul működő folyamat döntési értékei legfeljebb ϵ-nal térnek el egymástól.

Érvényesség. Bármely hibátlanul működő folyamat döntési értéke a hibátlanul működő folyamatok kezdeti értékei által meghatározott tartományba esik.

Befejezés. Minden hibátlanul működő folyamat végül döntést hoz.

Ez a probléma előfordul például az óraszinkronizáló algoritmusokban, ahol a folyamatok óraértékeket kezelnek, de ezek az értékek nem egyeznek meg pontosan. A gyakorlatban számos osztott hálózati algoritmus dolgozik közelítőleg szinkronizált órákkal, ahol rendszerint elegendő az óraértékek közelítő megegyezése.

Itt csak teljes gráfokban vizsgáljuk a közelítő megegyezési problémát. A probléma megoldásának egyik módja, hogy eljárásként alkalmazzuk a közönséges bizánci megegyezési algoritmust. Ehhez feltesszük, hogy $n > 3f$.

> **BizKözelMegegyezés (vázlatosan)**
>
> A folyamatok egy közönséges bizánci megegyezés algoritmust futtatnak, hogy minden folyamat számára egy értéket válasszanak. Ezek az algoritmusok párhuzamosan futnak. A P_i folyamat algoritmusa az első menetben üzenetet küld minden más folyamatnak, azután a folyamatok a kapott értékeket bemenő adatként használják fel egy bizánci megegyezést kereső algoritmusban. Amikor ezek az algoritmusok befejeznek, minden jó folyamat ugyanazokat a döntési értékeket választja minden folyamat számára. Minden folyamat a döntési értékek multihalmazából az $\lfloor \frac{n}{2} \rfloor$-edik legnagyobb értéket választja saját döntési értékeként.

A működés jobb megértéséhez megjegyezzük, hogy ha P_i nem hibás, akkor a bizánci megegyezés érvényességi feltétele biztosítja, hogy a hibamentes folyamatok által P_i számára választott érték P_i aktuális bemeneti értéke. Mivel $n > 3f$, ezért a multihalmaz középső értékének a hibamentes folyamatok kezdeti értékeinek tartományában kell lennie.

7.15. tétel . *Ha $n > 3f$, akkor a* BizKözelMegegyezés *algoritmus egy n-csúcsú teljes gráfra megoldja a közelítő megegyezés problémáját.*

Most egy másik algoritmust mutatunk be, amely nem alkalmazza a bizánci megegyezést. Főleg azért mutatjuk be ezt az algoritmust, mert könnyű kiterjeszteni az aszinkron hálózati modellre, amelyet majd a 21. fejezetben tárgyalunk. A bizánci megegyezés problémája az aszinkron hálózatokban nem oldható meg. A második megoldásnak olyan tulajdonsága is van, hogy esetenként kevesebb menet után befejeződik, mint amit a bizánci megegyezés igényel. Ez attól függ, milyen messze vannak egymástól a hibátlan folyamatok kezdeti értékei. Az algoritmus a fokozatos közelítésen alapul. Az egyszerűség kedvéért először az algoritmus be nem fejeződő változatát írjuk le, azután a befejezést külön elemezzük. Ez az algoritmus is felteszi, hogy $n > 3f$.

Szükségünk van néhány jelölésre és kifejezésre. Először, ha U egy valós számokból álló véges multihalmaz legfeljebb $2f$ elemmel, és $u_1, \ldots, u_k$ a U multihalmaz elemeinek nem csökkenő sorozata, akkor legyen *csökkentés*(U) annak eredménye, hogy U-ból eltávolítjuk az f legkisebb és f legnagyobb elemet, azaz legyen az $u_{f+1}, \ldots, u_{k-f}$ elemekből álló multihalmaz. Ha U egy valós számokból álló, véges, nem üres multihalmaz, és $u_1, \ldots, u_k$ a U multihalmaz elemeinek nem csökkenő sorozata, akkor legyen *kiválasztás*(U) annak eredménye, hogy U-ból kiválasztjuk a legkisebb elemét és azután minden f-edik elemet, azaz $u_1, u_{f+1}, u_{2f+1}, \ldots$. Végül, ha U egy valós számokból álló, véges, nem üres multihalmaz, akkor legyen *átlag*(U) az U multihalmaz elemeinek számtani közepe.

Azt mondjuk, hogy egy valós számokból álló, véges, nem üres multihalmaz *tartománya* a legkisebb intervallum, amely a multihalmaz minden elemét tartalmazza, és *szélessége* a tartomány hossza.

A második megoldás a következő.

KovergensKözelMegegyezés algoritmus (vázlatosan)

A P_i folyamat karbantart egy *érték* változót, amely az utolsó becslést tartalmazza. Kezdetben *érték*$_i$ a P_i folyamat kezdeti értékét tartalmazza. P_i minden menetben a következőket végzi el.

Először, elküldi saját *érték*-ét minden folyamatnak, beleértve saját magát is.[2] Ezután az adott menetben kapott összes értéket összegyűjti egy W multihalmazban; ha P_i nem kap értéket egy bizonyos másik folyamattól, akkor a multihalmazban egy előre megadott értéket rendel ahhoz a folyamathoz, és ezzel biztosítja, hogy $|W| = n$ teljesüljön.

Ezután P_i beállítja *érték*-et *átlag*(*választás*(*csökkentés*(W)))-re, azaz P_i eldobja W f legkisebb és f legnagyobb elemét. A megmaradó elemek közül P_i csak az i-edik elemet és attól számítva minden f-edik elemet választja. Végül az így kiválasztott elemek átlagát teszi az *érték*-be.

Azt állítjuk, hogy bármelyik menetre igaz, hogy a hibátlan folyamatok *érték*-ei a hibátlan folyamatoknak közvetlenül az adott menet előtti *érték*-ei által meghatározott tartományban vannak. Továbbá az is igaz minden menetre, hogy a hibátlan folyamatok *érték*-ei multihalmazának szélessége minden menetben lega-

[2] A saját magának való küldést rendszerint helyi átmenettel szimulálják.

lább $(\lfloor \frac{n-2f-1}{f} \rfloor + 1)$-ed részére csökken. Ha $n > 3f$, akkor ez az „osztótényező" egynél nagyobb.

7.16. lemma . *Tegyük fel, hogy pontosan a* KovergensKözelMegegyezés *egyik végrehajtásának r-edik menete után érték$_i$ = v, ahol P_i egy hibátlan folyamat. Akkor közvetlenül az r-edik menet előtt v a hibátlan folyamatok tartományában van.*

Bizonyítás. Ha W_i a P_i folyamat által az r-edik menetben összeállított multihalmaz, akkor W_i-nek legfeljebb f olyan eleme van, melyek hibás folyamatok által küldött értékek. Ekkor közvetlenül az r-edik menet előtt *csökkentés*(W_i) minden eleme a hibátlan folyamatok *érték*-einek a tartományában van. Ebből következik, hogy ugyanez teljesül *átlag*(*választás*(*csökkentés*(W_i)))-re, azaz a *érték$_i$* új értékére is. □

7.17. lemma . *Tegyük fel, hogy pontosan a* KovergensKözelMegegyezés *egyik végrehajtási sorozatának éppen az r-edik menete után lesz az érték$_i$ = v és érték$_{i'}$ = v′, ahol i és i′ két hibátlan folyamat indexe. Ekkor*

$$|v - v'| \leq \frac{d}{\lfloor \frac{n-2f-1}{f} \rfloor + 1},$$

ahol d a hibátlan folyamatokhoz közvetlenül az r-edik menet előtt tartozó érték-ek tartományának szélessége.

Bizonyítás. Legyen W_i, illetve W_i' a P_i és $P_{i'}$ folyamatok által az r-edik menetben összegyűjtött multihalmaz. Legyen S_i, illetve $S_{i'}$ a *kiválasztás*(*csökkentés*(W_i)), illetve a *kiválasztás*(*csökkentés*(W_i')) multihalmaz. Legyen $c = \lfloor \frac{n-2f-1}{f} \rfloor + 1$; megjegyezzük, hogy c pontosan az S_i-ben és az S_i'-ben lévő elemek száma. Jelöljük S_i elemeit $u_1, \ldots, u_c$-vel, S_i' elemeit pedig $u_1', \ldots, u_c'$-vel, mindkét esetben nem csökkenő sorrendben. Egy segédtétellel kezdünk, amely szerint a csökkentett multihalmazok mérete legfeljebb f elemmel különbözik.

7.18. segédtétel . $|$*csökkentés*(W_i) − *csökkentés*$(W_{i'})| \leq f$.

Bizonyítás. Mivel a hibátlanul működő folyamatok mindegyike egy-egy értéket ad mind W_i-hez, mind pedig $W_{i'}$-höz, ezért $|W_i - W_{i'}| \leq f$. Meg tudjuk mutatni, hogy egy-egy legkisebb elemet eltávolítása mindkét halmazból nem növeli a különbséghalmazban lévő elemek számát, és ugyanez érvényes egy-egy legnagyobb elem eltávolítására is. Ezt a két megállapítást f-szer alkalmazva megkapjuk a kívánt eredményt. □

A 7.18. segédtétel segítségével bizonyítjuk a következő segédtételt.

7.19. segédtétel . *Ha $1 \leq j \leq c-1$, akkor $u_j \leq u_{j+1}'$ és $u_j' \leq u_{j+1}$.*

Bizonyítás. Csak az első állítást bizonyítjuk; a második állítás bizonyítása hasonló. Kihasználjuk, hogy u_j a *csökkentés*(W_i) multihalmaz $((j-1)f+1)$-edik

legkisebb eleme, továbbá u'_{j+1} a *csökkentés*$(W_{i'})$ $(jf+1)$-edik legkisebb eleme. Mivel a 7.18. segédtétel szerint *csökkentés*$(W_{i'})$-nek legfeljebb f eleme nem eleme *csökkentés*(W_i)-nek, ezért $u_j \leq u'_{j+1}$. □

A kívánt korlát kiszámításával befejezzük a 7.17. lemma bizonyítását.

$$|v - v'| = |\mathit{átlag}(S_i) - \mathit{átlag}(S_{i'})| = \frac{1}{c}|(\sum_{j=1}^{c}(u_j - u'_j))| \leq$$

$$\leq \frac{1}{c}(\sum_{j=1}^{c}|u_j - u'_j|) = \frac{1}{c}(\sum_{j=1}^{c}(\max(u_j, u') - \min(u_j, u'_j))).$$

A 7.19. segédtétel szerint $\max(u_j, u'_j) \leq \min(u_{j+1}, u'_{j+1})$ teljesül minden olyan j-re, amelyre $1 \leq j \leq c-1$, ezért az utolsó kifejezés értéke legfeljebb

$$\frac{1}{c}(\sum_{j=1}^{c-1}(\min(u_{j+1}, u'_{j+1}) - \min(u_j, u'_j))) + \frac{1}{c}(\max(u_c, u'_c) - \min(u_c, u'_c)),$$

amiből összevonással adódik, hogy

$$\frac{1}{c}(\max(u_c, u'_c) - \min(u_1, u'_1)).$$

Mivel *csökkentés*(W_i) és *csökkentés*$(W_{i'})$ minden eleme a hibátlan folyamatok r-edik menete előtt *érték*-einek tartományában van, ezért az u_c, u'_c, u_1 és u'_1 értékek is ebben a tartományban vannak. Ezért az utolsó kifejezés legfeljebb $\frac{d}{c}$, és éppen ezt akartuk belátni. □

Befejeződés. A KonvergensKözelMegegyezés algoritmust befejeződő algoritmussá alakítjuk, azaz olyan algoritmussá, amelyben végül minden folyamat dönt. (Valójában végül minden folyamat megáll.) Nevezetesen, minden hibátlan folyamat az első menetben kapott értékek tartományát használja annak a menetszámnak a meghatározásához, amikor már bármely két hibátlan folyamat *érték*-ei legfeljebb ϵ-nal különbözhetnek. Ezt minden folyamat megteheti, mivel ismeri ϵ értékét és a garantált konvergenciasebességet, továbbá tudja, hogy azok az értékek, melyeket az első menetben kap, minden hibátlan folyamat kezdeti értékeit tartalmazzák. Különböző hibátlan folyamatok azonban különböző menetszámokat számolhatnak ki.

Minden P_i folyamat, amely eléri az általa kiszámított menetet, a saját aktuális *érték*-ét választja döntési értéknek. Miután ezt megtette, a P_i folyamat szórja a saját *érték*-ét egy különleges *leállás* címkével együtt, majd leáll. Ha bármely P_j folyamat P_i-től *leállás* címkével együtt kapja az *érték*-et, akkor ezt használja P_i üzeneteként az adott és minden későbbi menetre nézve (mindaddig, amíg a saját kiszámított menetszáma alapján P_j úgy nem dönt, hogy ő is leáll).

Bár a hibátlan folyamatok különböző menetszámokat számolhatnak, világos, hogy a legkisebb ilyen becslés is helyes. Ezért amikor a legelső hibátlan folyamat

leáll, *érték*-einek tartománya már elég kicsi. A későbbi menetek során a hibátlan folyamatok *érték*-einek tartománya sohasem nő, bár arra nincs garancia, hogy tovább csökken.

7.20. tétel . *Ha $n > 3f$, akkor a fenti befejezéssel kiegészített* KONVERGENSKÖZELMEGEGYEZÉS *egy n-csúcsú teljes gráfra megoldja a közelítő megegyezési problémát.*

Bonyolultságelemzés. Nincs az n, f, ϵ értékektől és a hibátlan folyamatok kezdeti értékeiből álló multihalmaz szélességétől függő felső korlát arra, hogy mennyi idő alatt döntenek a KONVERGENSKÖZELMEGEGYEZÉS algoritmusban a hibátlan folyamatok. Ennek oka, hogy a hibás folyamatok az első menetben tetszőleges értéket küldhetnek, ami azt okozhatja, hogy a hibátlan folyamatok akármilyen nagy menetszámot is kiszámolhatnak befejezésükhöz.

A gyakorlatokban elemezzük azokat a folyamatok számára és az összefüggőségre vonatkozó korlátokat, amelyeknek teljesülniük kell a közelítő megegyezési probléma megoldásához. A 21. fejezetben az aszinkron hálózati modellel kapcsolatban újra tárgyaljuk ezeket a problémákat.

7.3.. A véglegesítési probléma

Ebben a szinkron rendszerek osztott megegyezési problémáiról szóló utolsó alfejezetben az *osztott adatbázis véglegesítés* problémájának kulcsgondolatait mutatjuk be. Amint azt az 5.1. alfejezetben elemeztük, a probléma akkor merül fel, amikor egy adatbázis tranzakció végrehajtásában több folyamat vesz részt. A feldolgozás után minden folyamat kialakítja kezdeti „véleményét" arról, hogy a tranzakciót *véglegesíteni* kell (azaz eredményeit véglegessé és más folyamatok számára hozzáférhetővé tenni), vagy *elvetni* (azaz eredményeit eldobni). Egy folyamat általában a véglegesítést részesíti előnyben, ha a tranzakcióval kapcsolatos számításait sikeresen befejezte, egyébként pedig inkább az eldobást választja. Feltesszük, hogy a folyamatok kommunikálnak, és végül megegyeznek a *véglegesítés* vagy *eldobás* kimenetben. Ha lehetséges, akkor az eredmény *véglegesítés*.

Léteznek olyan erre a problémára megoldások a gyakorlatban előforduló osztott hálózatok estén is, amelyekben folyamat- és vonalhibák együtt is előfordulhatnak. Az 5. fejezet eredményeiből azonban az következik, hogy ha nincs felső korlát a vonalhibák számára, akkor nincs megoldás. Ezért az üzenetvesztésekre bizonyos korlátokat kell feltételeznünk.

7.3.1.. A feladat

A véglegesítési problémának azt az egyszerűsített változatát tekintjük, amelyben a hálózatban nincs üzenetvesztés, csak folyamathiba. Ha ennek a fejezetnek az algoritmusait működő hálózatokban akarjuk megvalósítani, akkor más mechanizmusokat is kell alkalmazni kell, például ismételt átvitelt az elveszett üzenetek pótlására. Tetszőleges számú folyamat megállási hibát megengedünk.

Feltesszük, hogy a bemeneti tartomány $\{0,1\}$, ahol az 1 a *véglegesítésnek*, 0 pedig az *elvetésnek* felel meg. Figyelmünket arra az esetre korlátozzuk, amelyben a hálózat gráfja teljes. A helyességi feltételek a következők.

Megegyezés. Bármely két folyamatnak ugyanazt az értéket kell választania.

Érvényesség.

1. Ha bármely folyamat 0 értékkel kezd, akkor 0 az egyetlen lehetséges döntési érték.
2. Ha minden folyamat az 1 értékkel kezd és nincs hibás folyamat, akkor az 1 az egyetlen lehetséges döntési érték.

Befejezés. Ennek két változata van. A *gyenge befejezési feltétel* (GyBF) azt mondja, hogy ha nincs hiba, akkor végül minden folyamat dönt. Az *erős befejezési feltétel* (EBF) (amelyet *nemblokkolási feltételnek* is neveznek) azt mondja, hogy végül minden hibátlan folyamat dönt.

Az erős befejezési feltételt kielégítő véglegesítő algoritmusokat időnként *nem blokkoló véglegesítő algoritmusoknak* is nevezzük, míg a gyenge befejezési feltételt kielégítő, de az erős befejezési feltételt nem kielégítő véglegesítő algoritmusokat időnként *blokkoló algoritmusoknak* nevezzük.

Megjegyezzük, hogy az a megegyezési feltételünk, hogy *bármely két folyamat ugyanazt az értéket választja*. Ezek szerint egy hibás folyamatnak sem engedjük meg, hogy a többi folyamattól eltérő értéket válasszon. Ezt azért követeljük meg, mert a véglegesítő protokollok gyakorlati alkalmazásaiban egy folyamat hibássá válhat, majd később újra helyesen működhet. Tegyük fel például, hogy a P_i folyamat hibássá válása előtt a *véglegesítés* döntést választja, később pedig más folyamatok az *eldobást* választják. Ha a P_i folyamat helyreállítódik és fenntartja a *véglegesítést*, ellentmondás jön létre.

A probléma formális megfogalmazása hasonló ahhoz a két esethez, amelyeket korábban már vizsgáltunk: az összehangolt támadáshoz az 5.1. alfejezetben és a megegyezési problémához megállási hibák esetében a 6.1. alfejezetben. A véglegesítési probléma és az összehangolt támadás problémája között a legfontosabb különbség az, hogy itt folyamathibát vizsgálunk, és nem vonalhibát; és különbség van az érvényességi feltételben is. A véglegesítési probléma és a megállási megegyezési probléma közötti fontos különbségek: elsősorban az eltérő érvényességi feltétel, másodsorban pedig a gyengébb befejezési feltétel. A 6.7. alfejezetnek a megállási megegyezési problémára vonatkozó eredményeiből következik, hogy az erős befejezési feltétel esetében a véglegesítési probléma megoldásához legalább $n-1$ menetre van szükség. (Megjegyezzük, hogy a 6.33. tétel bizonyítása a véglegesítési érvényességi feltétel esetében is helyes.)

Az alfejezet hátralévő részében két, a gyakorlatban használt véglegesítő algoritmust ismertetünk (azzal az egyszerűsítéssel, hogy csak folyamathibákat engedünk meg). Az első, a *kétfázisú véglegesítés*, blokkoló algoritmus, míg a második, a *háromfázisú véglegesítés*, nem blokkoló. Azután megadunk egy egyszerű alsó korlátot a probléma megoldásához szükséges üzenetek számára abban az esetben, ha csak a gyenge befejezési feltétel teljesülését követeljük meg.

7.3.2.. Kétfázisú véglegesítés

A gyakorlatban legismertebb véglegesítő algoritmus a *kétfázisú véglegesítés*; ez az egyszerű algoritmus garantálja a gyenge befejezést.

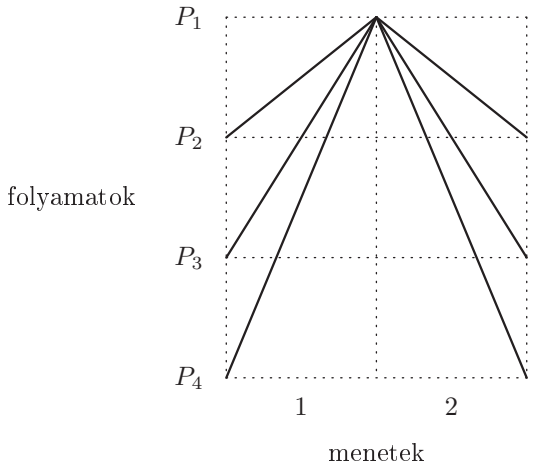
A KétfázisúVéglegesítés algoritmus

Ez az algoritmus feltételez egy kitüntetett folyamatot, például P_1-et.

1. menet. P_1 kivételével minden folyamat elküldi kezdeti értékét P_1-hez, és minden olyan folyamat, melynek kezdeti értéke 0, a 0 értéket választja. P_1 ezeket az értékeket és saját kezdeti értékét egy vektorba gyűjti össze. Ha a vektor minden eleme 1, akkor P_1 az 1 értéket választja. Egyébként – azaz, ha van a vektorban olyan elem, amely 0, vagy bizonyos elem hiányzik (mivel a megfelelő folyamattól nem érkezett üzenet) – P_1 a nulla értéket választja.

2. menet. P_1 döntését szórja a többi folyamatnak. Bármely, P_1-től különböző folyamat, amely a második menetben üzenetet kap és az első menetben még nem döntött, azt az értéket választja, amelyet a második menetbeli üzenetben kap.

Lásd a 7.4. ábrát a KétfázisúVéglegesítés hibamentes meneteiben alkalmazott kommunikációs minták illusztrálására.[3]



7.4.. ábra. Kommunikációs minta a KétfázisúVéglegesítés algoritmusban.

[3]Menet fogalmunk nem pontosan egyezik meg a kétfázisú protokollokban szokásos fázisok fogalmával. Rendszerint az elején még egy további menetet alkalmaznak, amelyben P_1 a *véglegesítés* vagy *eldobás* értékeket igényli a többi folyamattól. Ekkor az első fázis ebből a többlet menetből és a mi első menetünkből áll. Egyszerűsített modellünknek és problémaleírásunknak köszönhetően nincs szükségünk erre a többlet menetre.

7.21. tétel . *A* KétfázisúVéglegesítés *algoritmus a gyenge befejezési feltétellel megoldja a véglegesítési problémát.*

Bizonyítás. A megegyezést, érvényességet és befejezést egyaránt könnyű belátni. □

A KétfázisúVéglegesítés azonban nem elégíti ki az EBF-t, azaz blokkoló algoritmus. Ez azért igaz, mert ha P_1 hibázik, mielőtt a második menetben elkezdené a szórást, akkor azok a hibátlan folyamatok, melyek kezdeti értéke 1, sohasem fognak dönteni. A gyakorlatban ha P_1 hibázik, akkor a többi folyamat bizonyos *befejezési protokollt* hajt végre és bizonyos esetekben sikerül döntést hozniuk. Például ha P_1 hibássá válik, de valamely másik folyamat, például P_i, már az első menetben a 0 értéket választotta, akkor P_1 értesítheti a többi jó folyamatot arról, hogy ő a 0 értéket választotta, és akkor azok biztonságosan választhatják a 0 értéket. A befejezési protokoll azonban nem minden esetben sikeres. Például tegyük fel azt, hogy P_1 kivételével minden folyamat az 1 bemeneti értékkel kezd, de P_1 hibássá válik, mielőtt bármilyen üzenet küldene. Ekkor egyetlen további folyamat sem értesül P_1 1-es kezdeti értékéről, és így az érvényességi feltétel szerint egyetlen folyamat sem választhatja az 1 értéket. Másrészt, egyetlen folyamat sem választhatja a nullát, mivel bármelyik folyamat mondhatja azt, hogy előfordulhat az, hogy P_1 hibássá válása előtt már az 1 értéket választotta, és az ellentmondás megsértené a megegyezési feltételt.

Bonyolultsági elemzés. A KétfázisúVéglegesítés algoritmus csak két menetet igényel. Emlékeztetünk arra, hogy a 6.33. tétel $(f+1)$-es alsó korlátot ad a megállási megegyezéshez szükséges menetek számára. A KétfázisúMegegyezés algoritmusra vonatkozó alsó korlát ennek nem mond ellent, mivel a KétfázisúMegegyezés csak a gyenge befejezési feltételnek tesz eleget. Ha a kommunikációs bonyolultságot a bármely végrehajtásban legrosszabb esetben küldött nem-*null* üzenetek számával mérjük, akkor $2n-2$ értéket kapunk; például egy hibamentes végrehajtásban ennyi az elküldött üzenetek száma.

7.3.3.. Háromfázisú véglegesítés

A HáromfázisúVéglegesítés a KétfázisúVéglegesítés olyan javított változata, amely garantálja az erős befejezést.

Ennek alapja az, hogy P_1 csak akkor választja az 1 értéket, ha minden olyan folyamat, amely még nem hibázott, „kész” az 1 értéket választani. Egy további menetet igényel, hogy P_1 erről megbizonyosodjon. Először az algoritmus első három menetét írjuk le és elemezzük. Az algoritmus további részét, amely a nem blokkoló tulajdonságot biztosítja, később ismertetjük.

HáromfázisúVéglegesítés algoritmus, első három menet (vázlatosan)

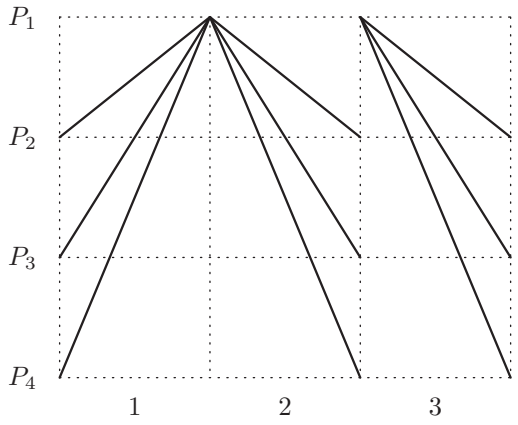
Első menet. P_1 kivételével minden folyamat elküldi kezdeti értékét P_1-nek, és minden olyan folyamat, melynek kezdeti értéke 0, a 0 értéket választja. A P_1 folyamat ezeket a kapott értékeket és a saját kezdeti értékét egy

vektorba gyűjti össze. Ha ennek a vektornak minden eleme 1, akkor P_1 `kész` állapotba kerül, de még nem választ. Egyébként – azaz ha ennek a vektornak van 0 eleme vagy üres eleme (mert bizonyos üzenet nem érkezett meg) – P_1 a 0 értéket választja.

Második menet. Ha a P_1 folyamat a 0 értéket választja, akkor szétküldi a `dönt`(0) üzenetet. Ha nem, akkor P_1 a `kész` üzenetet szórja. Minden folyamat, amely `kész` üzenetet kap, `kész` állapotba kerül. Ha P_1 még nem választott, akkor az 1 értéket választja.

Harmadik menet. Ha P_1 az 1 értéket választotta, akkor szétküldi a `dönt`(1) üzenetet. Minden folyamat, amely `dönt`(1) üzenetet kap, az 1 értéket választja.

A 7.5. ábra a HáromfázisúVéglegesítés hibamentes futamaiban előforduló kommunikációs mintát ábrázolja.[4]



7.5.. ábra. Kommunikációs minta a HáromFázisúVéglegesítés algoritmusban.

Mielőtt a megfelelő befejezést biztosító protokollt ismertetnénk, elemezzük az első három menet után kialakuló helyzetet. Minden folyamat (hibás és hibátlan) állapotait négy – egymást kizáró és minden esetet magában foglaló – osztályba soroljuk:

1. 0-*döntés*: azok az állapotok, amelyekben a folyamat a 0 értéket választja;
2. 1--*döntés*: azok az állapotok, amelyekben a folyamat az 1 értéket választja;
3. *kész*: azok az állapotok, amelyekben a folyamat még nem választott, de *kész*;

[4] Az itt alkalmazott menetek nem felelnek meg pontosan a háromfázisú véglegesítési protokollok szokásos fázisainak. Rendszerint hozzáadnak még egy igénylő menetet az elejéhez és valamilyen nyugtázást is használnak.

4. *bizonytalan*: azok az állapotok, amelyekben a folyamat még nem választott és nem *kész*.

A HÁROMFÁZISÚVÉGLEGESÍTÉS kulcstulajdonságait fogalmazza meg a következő lemma: olyan állapotkombinációkat ír le, amelyek együtt nem fordulhatnak elő.

7.22. lemma . *A* HÁROMFÁZISÚVÉGLEGESÍTÉS *három menete után teljesülnek a következők.*

1. *Ha valamelyik folyamat kész vagy 1-döntés állapotban van, akkor minden folyamat kezdeti értéke 1.*
2. *Ha valamelyik folyamat 0-döntés állapotban van, akkor nincs folyamat 1-döntés állapotban és nincs hibátlan folyamat a kész állapotban.*
3. *Ha valamely folyamat 1-döntés állapotban van, akkor nincs folyamat 0-döntés állapotban és nincs hibátlan folyamat bizonytalan állapotban.*

Bizonyítás. Nyilvánvaló. A bizonyítás egyetlen meggondolandó része a harmadik feltétel belátása. Ehhez megjegyezzük, hogy P_1 csak a második menet végén dönthet az 1 mellett, miután a *kész* üzeneteket szétküldte. Ez azt jelenti, hogy a második menet végén P_1 megtudja, minden más folyamat vagy már megkapta és feldolgozta a *kész* üzenetet, és ezáltal *kész* állapotba került, vagy hibássá vált. (A modell szinkronizáltsága ebben az esetben fontos.) □

Most bebizonyíthatjuk, hogy a bennünket érdeklő feltételek többsége már a harmadik menet után teljesül.

7.23. lemma . *A* HÁROMFÁZISÚVÉGLEGESÍTÉS *három menete után teljesülnek a következők.*

1. *A megegyezési feltétel teljesül.*
2. *Az érvényességi feltétel teljesül.*
3. *Ha a P_1 folyamat nem hibázott, akkor minden hibátlan folyamat döntött.*

Bizonyítás. A megegyezés feltétele a 7.22. lemmából következik, és az érvényességi feltételnek azt a részét biztosítja, amely szerint ha valamely folyamat a 0 értékkel kezd, akkor 0 az egyetlen lehetséges döntési érték. Az érvényességi feltétel másik része egyszerűen ellenőrizhető.

Végül ha P_1 nem hibás, akkor minden hibátlan folyamat döntött. Ez azért igaz, mert bármit tesz a többi folyamat, P_1 mindenképpen dönteni fog, és ha P_1 döntött, döntését azonnal szétküldi a többi folyamatnak, amelyek ugyanúgy döntenek. □

Ez a három menet azonban még nem elég a nem blokkoló véglegesítési probléma megoldásához, mivel nem garantálják az erős befejezést. Ha P_1 nem hibás, akkor a 7.23. lemma szerint minden hibamentes folyamat dönt. Ha azonban P_1 hibássá válik, akkor előfordulhat, hogy a többi folyamat bizonytalan állapotban marad. Ennek elkerülése érdekében a többi folyamatnak az első három menet után végre kell hajtania a *befejezési protokollt*. A pontos részletek többféleképpen is megfogalmazhatók; az alábbiakban egy lehetséges változatot ismertetünk.

HÁROMFÁZISÚVÉGLEGESÍTÉS befejezési protokoll

4. menet. Minden folyamat (amely még nem hibázott) elküldi jelenlegi állapotát (ami 0-*döntés*, 1-*döntés, kész* vagy *bizonytalan*) P_2-nek. P_2 ezeket az állapotértékeket és a saját állapotát összegyűjti egy vektorba. Nem biztos, hogy a vektor minden eleme ki van töltve – P_2 figyelmen kívül hagyja azokat az elemeket, amelyek nincsenek kitöltve. Ha a vektor csak 0-*döntés* tartalmaz 0-*döntés* értéket, és P_2 még nem döntött, akkor P_2 a 0 értéket választja. Ha a vektor tartalmaz legalább egy 1-*döntés* értéket, és P_2 még nem döntött, akkor akkor P_2 az 1 értéket választja. Ha a vektor minden kitöltött eleme *bizonytalan*, akkor P_2 a 0 értéket választja. Egyébként – azaz a vektorban csak *bizonytalan* és *kész* értékek vannak, és van legalább egy *kész*, akkor P_2 *kész* állapotba kerül, de még nem dönt.

5. menet. Ebben és a következő menetben P_2 ahhoz hasonlóan viselkedik, ahogy P_1 a 2. és 3. menetben. Ha P_2 már döntött, akkor egy `dönt` üzenetben szétküldi a döntését. Ha nem dönt, akkor a *kész* állapotát szórja. Ha bármely folyamat, amely még nem döntött, `dönt`(0) vagy `dönt`(1) üzenetet kap, akkor az üzenetnek megfelelően a 0, illetve 1 értéket választja. Ha egy folyamat *kész* értéket kap, állapota *kész* lesz. Ha P_2 még nem döntött, akkor az 1 értéket választja.

6. menet. Ha P_2 az 1 értéket választotta, akkor szétküldi a `dönt`(1) értéket. Ha olyan folyamat kap `dönt`(1) üzenetet, amely még nem döntött, az az 1 értéket választja.

A 6. menet után a protokoll hasonló 3-3 menettel folytatódik, melyeket rendre a P_3, $P_4, \ldots,$ P_n koordinálnak.

7.24. tétel . *A befejezési protokollt is tartalmazó teljes* HÁROMFÁZISÚVÉGLEGESÍTÉS *nem blokkoló véglegesítő algoritmus.*

Bizonyításvázlat. Először azt mutatjuk meg, hogy a 7.22. lemmában felsorolt 3 tulajdonság nem csak három menet után teljesül(ahogy a lemmában állítottunk), hanem a teljes HÁROMFÁZISÚVÉGLEGESÍTÉS algoritmus *bármelyik* menete után. A bizonyítás a menetek száma szerinti indukcióval történik.

Ezután a megegyezés és az érvényességi feltétel egy része – az, hogy ha valamelyik folyamat a 0 értékkel kezd, akkor 0 az egyetlen lehetséges döntési értéke – a korábbiakhoz hasonlóan következik a 7.22. lemma kiterjesztéséből. Az érvényességi feltétel másik része azért igaz, mert ha nincsenek hibák, akkor minden folyamat dönt az első három menet során.

Az erős befejezési feltétel van még hátra. Ha minden folyamat hibás, akkor a tulajdonság üres halmazra vonatkozik, ezért biztos igaz. Egyébként tegyük fel, hogy P_i hibátlan. Ekkor azalatt, míg P_i a koordináló folyamat, minden hibátlan folyamat dönteni fog. □

Bonyolultsági elemzés. A HÁROMFÁZISÚVÉGLEGESÍTÉS itt bemutatott változata $3n$ menetet igényel. Ez még akkor is lényegesen magasabb a 6. fejezetben (ott megállási hibák fordulhattak elő) tanulmányozott megegyezési algoritmusoknál

általában elérhető, körülbelül n menetes felső korlátoknál, ha minden folyamat meghibásodhat. Természetesen a megállási hibákat tűrő megegyezési algoritmusok más érvényességi feltételt elégítenek ki, de kis módosítással a véglegesítési érvényességi feltételt is ki tudják elégíteni. Akkor miért jobbak a gyakorlatban a HÁROMFÁZISÚVÉGLEGESÍTÉS-hez hasonló algoritmusok?

A fő ok az, hogy a HÁROMFÁZISÚVÉGLEGESÍTÉS átalakítható úgy, hogy a hibamentes esetben alacsony legyen bonyolultsága. Ha egyetlen folyamat sem hibás, akkor 3 menet alatt minden folyamat dönt. Ekkor egy egyszerű protokoll segítségével a folyamatok megtudhatják, hogy már minden folyamat döntött, és a befejezési protokoll további részében már nem vesznek részt. Ezzel a kiegészítéssel az egész algoritmus szükséges meneteinek száma egy kis konstans, az üzenetek száma pedig $O(n)$.

7.3.4.. Alsó korlát az üzenetek számára

Ezt a fejezetet (és az első részt) azzal zárjuk, hogy megvizsgáljuk a véglegesítési probléma megoldásához szükséges üzenetek számát. Emlékezzünk arra, hogy a KÉTFÁZISÚVÉGLEGESÍTÉS algoritmus a hibamentes esetben $2n-2$ üzenetet használ. A HÁROMFÁZISÚVÉGLEGESÍTÉS valamivel többet, de még mindig $O(n)$ üzenetet igényel, ha az algoritmust úgy módosítjuk, hogy hamar megálljon. Ebben a szakaszban megmutatjuk, hogy $2n-2$ üzenetnél kevesebb még akkor sem elég a hibamentes esetben, ha megelégszünk egy blokkoló algoritmussal.

7.25. tétel . *A véglegesítési problémát megoldó algoritmusok még a gyenge befejezési feltétel esetében is legalább $2n-2$ üzenetet használnak az olyan végrehajtási sorozatokban, amelyekben minden bemenet* 1.

A fejezet hátralévő részében rögzítünk egy A véglegesítő algoritmust, és legyen α_1 A-nak az a hibamentes végrehajtása, amelyben minden bemenet 1. Célunk annak megmutatása, hogy α_1 legalább $2n-2$ üzenetet tartalmaz.

Ismét felhasználjuk a *kommunikációs mintának* a 6.7. alfejezetben megadott definícióját. Ezúttal arra használjuk a kommunikációs mintát, hogy leírja a hibamentes végrehajtási sorozatban elküldött üzenetek halmazát. (A korábbiaktól eltérően nem tételezzük fel, hogy minden folyamat minden menetben minden másik folyamatnak küld üzenetet). Az A algoritmus bármely α hibamentes végrehajtási sorozatából a nyilvánvaló módon készítjük a *minta*(α) kommunikációs mintát.

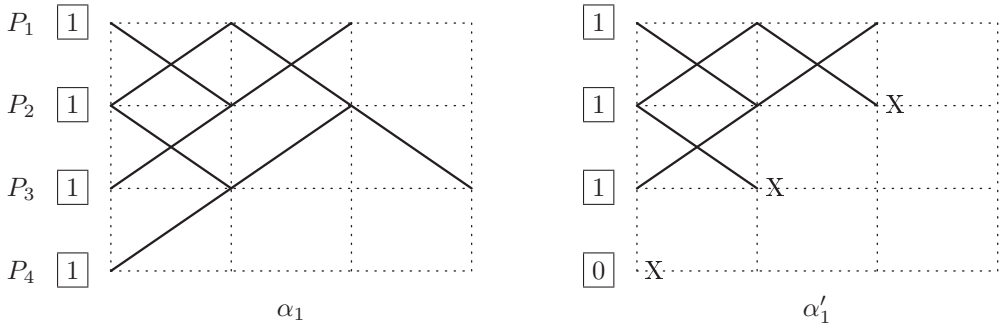
A különböző folyamatok között különböző időpontokban végbemenő információáramlás leírására felhasználjuk a γ kommunikációs mintának az 5.2.2. szakaszban definiált $\leq_\gamma$ rendezését is. Azt mondjuk, hogy a P_i folyamat a γ kommunikációs mintában hat a P_j folyamatra, ha van olyan k, amelyre $(i,0) \leq_\gamma (j,k)$) teljesül. A következő lemma tartalmazza az alsó korlát alapgondolatát.

7.26. lemma . *Bármely P_i és P_j folyamatra fennáll, hogy P_i hat P_j-re a minta(α_1)-ben.*

7.3.1. példa. Alsó korlát a véglegesítésre

A 7.26. lemma bizonyítása előtt bemutatunk egy példát, amely alátámasztja, hogy a lemma helyes. Tegyük fel, hogy α_1 (A-nak az a hibamentes végrehajtási sorozata, amelyben minden bemenet 1) pontosan azokat az üzeneteket tartalmazza, amelyeket a 7.6. ábra bal oldalán megjelöltünk.

Az érvényességi és a gyenge befejezési feltétel szerint α_1-ben végül minden folyamatnak 1-et kell választania. Vegyük észre, hogy α_1-ben P_4 nem hat P_1-re; vizsgáljuk meg, milyen problémákat okoz ez. Tekintsünk egy másik, α_1' végrehajtási sorozatot, amely csak annyiban tér el α-tól, hogy P_4 bemenete 0, és minden folyamat hibássá válik közvetlenül azután, hogy először hatott rá P_4. Az α_1' végrehajtási sorozatot a 7.6. ábra jobb oldali része mutatja; a hibákat X jelzi. Közvetlenül megmutatható, hogy $\alpha_1 \overset{1}{\sim} \alpha_1'$, amiből következik, hogy P_1 α_1-ben ugyancsak 1-et választ. Ez azonban megsérti az α_1-re vonatkozó érvényességi feltételt, ami ellenmondás.



7.6.. ábra. Az α_1 és α_1' végrehajtási sorozatban küldött üzenetek.

A 7.26. lemma ugyanazzal a gondolatmenettel bizonyítható, mint amit a 7.3.1. példában alkalmaztunk.

7.26. lemma bizonyítása. Az érvényesség és a gyenge befejezési feltételek szerint az α_1 mintában végül minden folyamatnak 1-et kell választania. Tegyük fel, hogy a lemma állítása hamis, és rögzítsünk két folyamatot, például P_i-t és P_j-t, amelyekre igaz, hogy P_i nem hat P_j-re $minta(\alpha_1)$-ben. Ekkor i és j definíciója szerint $i \neq j$. Állítsuk elő α_1'-et úgy, hogy α_1-ben P_i bemenetét 0-ra változtatjuk, és minden folyamat meghibásodik közvetlenül azután, hogy P_i először hatott rá. Ekkor $\alpha_1 \overset{j}{\sim} \alpha_1'$, ezért P_j α_1'-ben ugyancsak 1-et választ. Ez megsérti az érvényességi feltételt, ami ellentmondás. □

A 7.25. tétel bizonyításának befejezéséhez egyszerűen azt kell megmutatnunk, hogy abból a feltételből, hogy minden folyamat hat minden másik folyamatra, következik, hogy összesen legalább $2n-2$ üzenet van. Felhasználjuk a következő, kommunikációs mintákra vonatkozó lemmát.

7.27. lemma . *Legyen γ tetszőleges kommunikációs minta. Ha γ-ban egy $m \geq 1$ folyamatot tartalmazó halmaz minden eleme hat a rendszernek mind az n folyamatára, akkor γ-ban legalább $n+m-2$ üzenet (hármas) van.*

Bizonyítás. m szerinti indukcióval.

Az indukció alapja: $m = 1$. Legyen P_i az a folyamat, amelyről feltettük, hogy mind az n folyamatra hat. Mivel P_i mind az n folyamatra hat, γ szükségképpen tartalmaz üzenetet mind az $n-1$, P_i-től különböző folyamathoz. Ez legalább $n-1$ üzenet, és éppen erre volt szükségünk.

Indukciós lépés. Feltesszük, hogy a lemma teljesül m-re, és megmutatjuk, hogy $(m+1)$-re is teljesül. Legyen I olyan halmaz, amely $m+1$ olyan folyamatot tartalmaz, amelyek mind az n folyamatra hatnak. Az általánosság megszorítása nélkül feltehetjük, hogy az első menetben legalább egy I-beli folyamat üzenetet küld valamelyik folyamatnak. Ugyanis ha ez nem igaz, akkor eltávolíthatjuk mindazokat a kezdeti meneteket, amelyekben I-beli folyamat nem küld üzenetet; a megmaradó kommunikációs mintában még mindig teljesül, hogy I minden folyamata hat mind az n folyamatra. Legyen P_i egy olyan I-beli folyamat, amely γ-ban az első menetben küld üzenetet.

Tekintsük azt a γ' kommunikációs mintát, amely annyiban különbözik γ-tól, hogy eltávolítottunk belőle egy olyan üzenetet, amelyet az első menetben P_i küldött. Akkor az $I \setminus \{P_i\}$ halmaz minden eleme hat γ'-ben mind az n folyamatra. Az indukciós feltevés szerint γ'-ben legalább $n+m-2$ üzenet van. Ezért γ legalább $n+m-1 = n+(m+1)-2$ üzenetet tartalmaz, és éppen ezt akartuk belátni. □

Most már befejezhetjük a 7.25. tétel bizonyítását.

7.25. tétel bizonyítása. A 7.26. lemma szerint tetszőlegesen választott P_i és P_j folyamatokra igaz, hogy P_i hat P_j-re a $minta(\alpha_1)$-ben. Ezért a 7.27. lemmából következik, hogy $minta(\alpha_1)$-ben legalább $2n-2$ üzenet van. □

7.4.. Megjegyzések a fejezethez

A szakirodalomban a k-megegyezési problémát k-halmaz megegyezési problémának hívják. A problémát először Chaudhuri [73] vetette fel, mint a korábban már alaposan elemzett egyszerű megegyezési probléma természetes általánosítását. A MinTerjed algoritmus Chaudhuri, Herlihy, Lynch és Tuttle [75] cikkéből származik és egy olyan algoritmuson alapul, melyet eredetileg Chaudhuri [73] javasolt. A k-megegyezésre vonatkozó alsó korlát bizonyításának gondolatmenete a [75,76,77] cikkekből származik. Az alsó korlát bizonyításának algebrai topológiai háttere megtalálható Spanier algebrai topológiáról szóló klasszikus könyvében [266]. A Sperner-lemmát eredetileg Sperner [267] igazolta és később Spanier [266] elemezte.

A közelítő megegyezésre vonatkozó rész Dolev, Lynch, Pinter, Stark és Weihl cikkéből [98] származik. Ezzel a problémával kapcsolatosak Fekete, [110,111] valamint Attiya, Lynch és Shavit [24] cikkei is. A véglegesítési problémáról, valamint a KétfázisúVéglegesítés és a HáromfázisúElfogadás algoritmusok-

ról szóló anyag Bernstein, Hadzilacos és Goodman adatbázisok elméletével foglalkozó könyvéből [50] származik. Ez a könyv a miénknél részletesebben tárgyalja a protokollok gyakorlati megvalósítását, például a hibás folyamatok helyreállításával is foglalkozik. A véglegesítéshez szükséges üzenetek számára vonatkozó alsó korlát Dwork és Skeen eredménye [106].

7.5.. Gyakorlatok

7-1. Ha a k-megegyezés MinTerjed algoritmusa $\lfloor \frac{f}{k} \rfloor + 1$ helyett csak $\lfloor \frac{f}{k} \rfloor$ menetet fut, akkor legfeljebb hány döntést hozhatnak a hibátlanul működő folyamatok?

7-2. Adjunk jó alsó korlátot a 7.14. tétel bizonyításában szereplő *sorozat*(v) sorozat hosszára. Ehhez szükség van a sorozat explicit leírására.

7-3. Bizonyítsuk be, hogy az 1-futamok sorozatának összefésültje valóban k-futam. Ez magában foglalja annak belátását, hogy a futam definíciójában megkövetelt feltételek, valamint a jelekre vonatkozó feltételek is teljesülnek.

7-4. Bizonyítsuk be a 7.4. lemmát.

7-5. Bizonyítsuk be a 7.5. lemmát.

7-6. Bizonyítsuk be a 7.6. lemmát.

7-7. Bizonyítsuk be a 7.7. lemmát.

7-8. Legyen $n = 5$, $k = f = 2$ és $r = 1$.

(a) Adjuk meg részletesen az ezekhez a paraméterekhez tartozó Bermuda-háromszöget és annak címkézését k-futamokkal és folyamatindexekkel.

(b) Tekintsük azt az egyszerű A algoritmust, amely a következőképpen működik: minden folyamat egyszer cseréli az értékeket, és minden folyamat a legkisebb értéket választja azok közül, amelyeket kapott. Írjuk le a C_A Sperner-színezést.

(c) Meg tudunk adni az A algoritmushoz egy olyan kis szimplexet, amelyben három különböző döntési érték van?

7-9. Rögzítsünk tetszőleges n, f és ϵ értéket, tetszőleges $w \in \mathbb{R}_0^+$-t és $r \in \mathbb{N}$-et úgy, hogy $n > 3f$ teljesüljön. Írjuk le a KonvergensKözelMegegyezés algoritmusnak egy, az adott n, f és ϵ értékeknek megfelelő végrehajtási sorozatát, amelyben a hibátlan folyamatok kezdeti értékeit tartalmazó multihalmaz szélessége legfeljebb w, és amelyben a befejezés r menetnél többet vesz igénybe.

7-10. *Kutatási kérdés.* Módosítsuk a KonvergensKözelMegegyezés algoritmust úgy, hogy az az idő, amíg minden folyamat dönt, korlátozva van n, f, ϵ, valamint a hibátlan folyamatok kezdeti értékeit tartalmazó multihalmaz w szélességének valamilyen függvényével.

7-11. Tegyük fel, hogy a KonvergensKözelMegegyezés algoritmusban a folyamatok az *átlag*(*kiválasztás*(*csökkentés*(W))) helyett a következő értékek egyikét számítják ki:

(a) *átlag*(*választás*(W));
(b) *átlag*(*csökkentés*(W));
(c) *átlag*(W).

Így is megoldja az algoritmus a közelítő megegyezési problémát? Ha igen, miért igen, ha nem, miért nem?

7-12. Bizonyítsuk be, hogy a közelítő megegyezési probléma akkor és csak akkor oldható meg egy f bizánci hibát tűrő G hálózati gráfban, ha a következő két feltétel mindegyike teljesül:

(a) $n > 3f$;
(b) *összefügg*(G) $> 2f$.

7-13. Tervezzünk egy közelítő megegyezési algoritmust a megállási hibák esetére.

(a) Próbáljuk minimalizálni a megoldáshoz szükséges folyamatok és a hibák számának arányát.
(b) Próbáljuk minimalizálni a megoldáshoz szükséges menetek számát.

7-14. Fogalmazzuk meg a közelítő megegyezési probléma olyan változatát, amely rögzített r számú menetet használ, és amelyben ϵ nincs előre meghatározva. Minden folyamat egy valós értékkel indul, mint korábban. r menet után a folyamatoknak fel kell venniük a végső értéküket. Az érvényességi feltétel ugyanaz, mint korábban. A cél most a legjobb lehetséges megegyezés elérése. A megegyezés jóságát a hibátlan folyamatok végső értékeinek szélessége és a hibátlan folyamatok kezdeti értékeinek szélessége hányadosára vonatkozó felső korláttal jellemezzük.

(a) Milyen arányt ér el a KonvergensKözelMegegyezés ebben a modellben?
(b) Adjunk alsó korlátot az elérhető arányra, n, f és r függvényében. (*Útmutatás.* Alkalmazzuk a láncérvelés gondolatait ahhoz hasonlóan, ahogy a 6.33. tétel bizonyításában tettük. A kapott alsó és a felső korlátok valószínűleg nem egyeznek meg.)

7-15. Írjuk le a teljes HáromfázisúVéglegesítés algoritmus kódját (beleértve a befejezési protokollt is).

7-16. Bizonyítsuk be részletesen, hogy a 7.22. lemma a HáromfázisúVéglegesítés tetszőleges számú menetére érvényes.

7-17. Írjuk le részletesen a HáromfázisúVéglegesítés algoritmus olyan módosítását, amely lehetővé teszi, hogy a folyamatok hibamentes esetben gyorsan döntsenek és leálljanak. Az algoritmus hibamentes esetben kevés menetet alkalmazzon és $O(n)$ üzenetet. Bizonyítsuk be az algoritmus helyességét.

7-18. Tervezzünk olyan algoritmust a 6. fejezetben szereplő megállási megegyezési algoritmus stílusában, amely a véglegesítési problémát az erős befejezési feltétellel megoldja. Igyekezzünk minimalizálni a menetek számát.

7-19. *Kutatási kérdés.* 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 a kommunikációs bonyolultság?

7-20. Adjuk meg a 7.26. lemma részletes bizonyítását. Hol hibás a bizonyítás, ha α_1' létrehozásakor nem követeljük meg, hogy minden folyamat azonnal hibásodjon meg, miután P_i először hat rá, csak P_i kezdeti értékét változtatjuk meg 1-ről 0-ra?

7-21. Tervezzünk egy nem blokkoló véglegesítő algoritmust, amely a lehető legkevesebb üzenet használ hibamentes futamok esetében. Be tudjuk bizonyítani, hogy ez az üzenetszám optimális?