4. fejezet

Általános szinkron hálózatok algoritmusai

A 3. fejezetben nagyon egyszerű szinkron hálózatokban – egyirányú és kétirányú gyűrűkben – oldottuk meg a vezetőválasztás leegyszerűsített problémáit. Ebben a fejezetben a szinkron hálózatok egy bővebb osztályában egy nagyobb problémakörrel foglalkozunk. Nevezetesen, olyan algoritmusokról lesz szó, amelyek tetszőleges gráfokból vagy irányított gráfokból épített hálózatokban *vezető folyamatot* választanak, *szélességi keresést (SZK)* végeznek, *legrövidebb utakat*, *minimális feszítőfát (MFF)*, és *maximális független halmazt (MFH)* keresnek.

A vezetőválasztás feladata akkor vetődik fel, amikor egy hálózati számítás „vezérlését" kell rábízni egy folyamatra. A szélességi keresést, a legrövidebb utak és a minimális feszítőfa keresését az motiválja, hogy megfelelő kommunikációs szerkezeteket tudjunk építeni egy hatékony üzenetváltás biztosítása érdekében. A maximális független halmaz keresésének igénye a hálózati erőforrások hozzárendelésének feladatából adódik. (Később, a 15. fejezetben, az aszinkron hálózatok kapcsán újra találkozunk ezen feladatok és algoritmusok többségével.)

Ebben a fejezetben egy tetszőleges, erősen összefüggő, n darab csúcsot tartalmazó $G = (V, E)$ irányított hálózati gráffal dolgozunk majd. (Néha arra az esetre szorítkozunk, amikor az élek kétirányúak, azaz ilyenkor a gráf lényegében irányítatlan.) Most is feltesszük – mint ahogy azt a szinkron hálózatoknál megszokhattuk –, hogy a folyamatok csak a gráf irányított élei mentén érintkezhetnek egymással. A gráf csúcsaihoz az $1, \ldots, n$ indexeket rendeljük, de a gyűrűkkel ellentétben ez az indexelés egyáltalán nem utal a csúcsoknak a gráfban való elhelyezkedésére. A folyamatok nem ismerik sem saját, sem szomszédaik indexét, de szomszédaikra helyi nevekkel hivatkozhatnak. Feltesszük továbbá, hogy ha a P_i folyamatnak a P_j folyamat egyszerre kimenő és a bejövő szomszédja is, akkor ezt az egybeesést P_i felismeri.

4.1.. Vezetőválasztás általános hálózatokban

Idézzük fel először a vezetőválasztás problémáját, de azt most egy tetszőleges, erősen összefüggő irányított gráfban fogalmazzuk meg.

4.1.1.. A feladat

Tegyük fel, hogy minden folyamat rendelkezik egy egyedi azonosítóval, amit az azonosítók teljesen rendezett halmazából választunk ki; minden egyes folyamat azonosítója különbözik a hálózat más folyamatának azonosítójától, és semmiféle megkötés sincs arra nézve, hogy egy folyamat melyik azonosítóval rendelkezzen. Mint azt a 3. fejezetben is

megköveteltük, végül pontosan egy folyamat választhatja magát vezetőnek azáltal, hogy az állapotának a *státus* összetevőjét *vezető* értékre állítja. Akárcsak a 3. fejezetben, itt is több változata van a feladatnak.

1. Megkövetelhetjük, hogy az összes nemvezető folyamat *nem_vezető* értékre állítsa be a *státus*-át.
2. A folyamatok ismerhetik a csúcsok számát (n) és a gráf átmérőjét (*átm*), vagy ezen értékeknek egy felső korlátját.

4.1.2.. Egy egyszerű terjedő algoritmus

Most egy olyan egyszerű algoritmust mutatunk be, amelynek segítségével mind a vezető, mind a nemvezető folyamatok azonosíthatják magukat. Ennek az algoritmusnak előfeltétele, hogy a folyamatok ismerjék az *átm* értékét. Az algoritmus működése során a maximális egyedi azonosító terjed el a hálózatban, ezért nevezzük MaxTerjed algoritmusnak.

MaxTerjed algoritmus (vázlatosan)

Minden folyamat azt a maximális egyedi azonosítót tartja nyilván, amely a végrehajtás adott pillanatáig eljutott hozzá (kezdetben ez a saját egyedi azonosítója). Mindegyik lépésben minden egyes folyamat továbbítja ezt a maximális egyedi azonosítót a kimenő szomszédainak. Ha *átm* darab menet után egy folyamat a saját egyedi azonosítójával azonos értékű maximális egyedi azonosítót tárol, vezetőnek választja magát; egyébként nemvezető lesz.

A P_i folyamat kódja tehát a következő.

MaxTerjed algoritmus (formálisan)

Az üzenetábécé az egyedi azonosítók halmaza.

állapotok$_i$ összetevői:
u egy azonosító, kezdeti értéke a P_i folyamat egyedi azonosítója
max_azon egy azonosító, kezdeti értéke a P_i folyamat egyedi azonosítója
státus $\in$ {*ismeretlen, vezető, nem_vezető*}, kezdetben az értéke *ismeretlen*
menetek egy egész szám, kezdetben az értéke 0

üzenetek$_i$:
if *menetek* $<$ *átm* **then**
  send *max_azon* az összes $j \in$ *ki_szom* folyamatnak

átmenet$_i$:
menetek := *menetek* + 1
legyen U a *be_szom*-beli folyamatoktól érkező egyedi azonosítók halmaza
max_azon := max({*max_azon*} $\cup\, U$)
if *menetek* = *átm* **then**
  if *max_azon* = u
    then *státus* := *vezető*
    else *státus* := *nem_vezető*

Könnyű belátni, hogy a MaxTerjed algoritmus a maximális egyedi azonosítójú folyamatot választja vezetőnek. Jelöljük a maximális egyedi azonosítójú folyamat indexét i_{max}-szal, az egyedi azonosítóját u_{max}-szal. Be fogjuk bizonyítani az alábbi tételt.

4.1. tétel. . *A* MaxTerjed *algoritmus átm darab menetben a* $P_{i_{max}}$ *folyamatot vezetőnek, a többi folyamatot nemvezetőnek jelöli ki.*

Bizonyítás. Elég belátni a következő állítást.

4.1.1. állítás. *átm darab menet után státus$_{i_{max}}$ = vezető, és minden $j \neq i_{max}$ indexre a státus$_j$ = nem_vezet.*

A 4.1.1. állítás bizonyításának kulcsa az, hogy r menet után a maximális egyedi azonosító minden olyan folyamathoz eljut, amelynek az i_{max}-adik folyamattól a G gráfbeli irányított utak mentén vett távolsága legfeljebb r. Ezt fogalmazza meg az alábbi invariáns.

4.1.2. állítás. *Tetszőleges $0 \leq r \leq$ átm egész szám és tetszőleges P_j folyamat esetén teljesül, hogy ha a P_j folyamat $P_{i_{max}}$-tól vett távolsága legfeljebb r, akkor max-azon$_j = u_{max}$ fennáll az r-edik menet után.*

A gráf átmérőjének meghatározása miatt a 4.1.2. állításból következik, hogy a maximális egyedi azonosító az *átm*-adik menet végére minden folyamathoz eljut. A 4.1.2. állítás bizonyításához hasznos segítséget adnak a következő segéd-invariánsok.

4.1.3. állítás. *Tetszőleges r egész szám és tetszőleges P_j folyamat esetén r menet után menetek$_j = r$.*

4.1.4. állítás. *Tetszőleges r egész szám és tetszőleges P_j folyamat esetén r menet után max_azon$_j \leq u_{max}$.*

A 4.1.2., a 4.1.3. állításokból, és $r = átm - 1$ esetén a 4.1.4. állításból, valamint annak megfontolásából, hogy mi történik az *átm*-adik menetben, következik a 4.1.1. állítás, és ennek megfelelően a 4.1. tétel is. □

A MaxTerjed algoritmus a 3.3. alfejezetben tárgyalt LCR algoritmusnak egyfajta általánosítása, hiszen az LCR algoritmus szintén a maximális értéket terjesztette el a (gyűrűs) hálózatban. Meg kell azonban jegyeznünk, hogy az LCR algoritmus nem követelte meg az átmérőnek, mint speciális hálózati tulajdonságnak az ismeretét. Az LCR algoritmusban akkor vált egy folyamat vezetővé, amikor a saját egyedi azonosítóját kapta meg üzenetként, nem pedig egy meghatározott számú menet után, mint ahogy azt a MaxTerjed algoritmusban láttuk. Az LCR stratégiája gyűrűs hálózatokban megfelelő volt, de nem működik általános irányított gráfokban.

Bonyolultságelemzés.. Könnyű belátni, hogy *átm* darab menet összideje szükséges ahhoz, hogy a vezető folyamatot kiválasszuk (és a többi folyamat megtudja, hogy ő nem vezető). Az üzenetek száma *átm* $\cdot |E|$, ahol az $|E|$ az irányított gráf irányított éleinek száma, ugyanis minden menetben minden irányított él mentén el kell küldeni egy üzenetet.

Az átmérő felső korlátja.. Megjegyezzük, hogy az algoritmus akkor is helyesen működik, ha a folyamatok az átmérő helyett annak csak egy d felső korlátját ismerik. A futási idő ekkor nő, mivel az *átm* helyett a d-től függ.

4.1.3.. A kommunikációs bonyolultság csökkentése

Az az egyszerű optimalizálás[1], amelyet most alkalmazni fogunk, számos esetben képes a kommunikációs bonyolultságot javítani, habár a legrosszabb eset nagyságrendjét nem

[1] Az „optimalizálás" nem igazán megfelelő szó erre. A „javítás" jobb lenne, de a szakirodalomban az „optimalizálás" használata a megszokott.

csökkenti. Nevezetesen arról van szó, hogy a folyamatok csak akkor küldik el a saját maximális egyedi azonosítójuk értékét, amikor annak először birtokába jutnak, és nem minden menetben. A MaxTerjed algoritmus ezen módosítását OptMaxTerjed algoritmusnak hívjuk, a kódja pedig a következő.

OptMaxTerjed algoritmus (formálisan)

állapotok$_i$ új összetevője:
új_info logikai értékű változó, kezdetben *igaz*

üzenetek$_i$:
if *menetek* $<$ *átm* és *új_info* $=$ *igaz* **then**
  send *max_azon* az összes $j \in$ *ki_szom* folyamatnak

átmenet$_i$:
menetek $:=$ *menetek* $+ 1$
legyen U a *be_szom*-beli folyamatoktól érkező egyedi azonosítók halmaza
if $max(U) > max\_azon$
  then *új_info* $:=$ *igaz*
  else *új_info* $:=$ *hamis*
max_azon $:= \max(\{max\_azon\} \cup U)$
if *menetek* $=$ *átm* **then**
  if *max_azon* $= u$
    then *státus* $:=$ *vezető*
    else *státus* $:=$ *nem_vezető*

Érezhető, hogy ez a módosítás helyes algoritmust eredményez. De hogyan bizonyíthatjuk be ezt formálisan? Az egyik módja ennek az, hogy a MaxTerjed algoritmus bizonyításánál alkalmazott utat követjük, csak más invariáns állításokra támaszkodunk. Ez azonban sok olyan munkát is magában foglal, amit a korábbi bizonyításokban már egyszer elvégeztünk. Ahelyett, hogy mindezt újra végigcsinálnánk, kínálkozik egy másik lehetőség is. Egy olyan bizonyítást mutatunk, amely az OptMaxTerjed algoritmus és a MaxTerjed algoritmus formális kapcsolatára támaszkodik. Ez egy egyszerű példája az osztott algoritmusok helyességét ellenőrző úgynevezett *szimulációs* módszernek.

4.2. tétel. . *Az* OptMaxTerjed *algoritmus átm darab menetben az* i_{max}*-adik folyamatot vezetőnek, minden más folyamatot nemvezetőnek jelöl ki.*

Bizonyítás. Elég a 4.1.1. állításhoz hasonlóan bebizonyítani az alábbit.

4.1.5. állítás. *átm darab menet után* $státus_{i_{max}} =$ *vezető, és minden* $j \neq i_{max}$ *indexre* $státus_j =$ *nem_vezető.*

Kezdjük egy olyan invariáns bizonyításával, amely azt mondja ki, hogy egy folyamat *új_info* jelzője mindig *igaz* értékre áll be, valahányszor van olyan új információ, amit a folyamatnak a következő menetben tovább kell küldenie. Még pontosabban, ha a P_i folyamatnak valamelyik kivezető szomszédja a P_i folyamat maximális egyedi azonosítójánál kisebb azonosítót ismer, a P_i folyamat *új_info* jelzőjét *igaz*-ra kell állítani.

4.1.6. állítás. *Minden olyan i, j indexű folyamatra, ahol $j \in ki\_szom_i$, r $(0 \leq r \leq átm)$ menet után fennáll, hogy ha $max\_azon_j < max\_azon_i$, akkor $új\_info_i = igaz$.*

A 4.1.6. állítás r szerinti indukcióval igazolható. Az állítás $r = 0$ esetben teljesül, mert kezdetben mindegyik *új_info* jelző *igaz*. Az indukciós lépésben veszünk egy tetszőleges olyan P_i és P_j folyamatot, ahol $j \in ki\_szom_i$. Ha max_azon_i értéke nő az r-edik menetben, az $új\_info_i$-t mindenképpen *igaz*-ra kell állítani; ekkor tehát teljesül az állítás. Ha max_azon_i értéke nem nő, akkor két eset képzelhető el. Az egyik esetben már az r-edik menet előtt fennállt a $max\_azon_j \geq max\_azon_i$. Mivel a max_azon_j érték soha nem csökken, a $max\_azon_j \geq max\_azon_i$ igaz az r-edik menet után is. A másik esetben az r-edik menet előtt $max\_azon_j < max\_azon_i$ teljesül. Az indukciós feltevés miatt ilyenkor $új\_info_i = igaz$, amelynek következtében az r-edik menetben a max_azon_i új információ eljut a P_i folyamattól a P_j folyamathoz, így $max\_azon_j \geq max\_azon_i$ megvalósul. Egyik esetben sem kell tehát az $új\_info_i$-t *igaz*-ra állítani.

Most ahhoz, hogy az OptMaxTerjed helyességét belássuk, gondolatban futtassuk egymás mellett ezt és a MaxTerjed algoritmust, úgy, hogy a kezdőérték-beállítás mindkét esetben azonos legyen. A bizonyítás lelke az az invariáns állításként megfogalmazott *szimulációs kapcsolat*, amely azt fejezi ki, hogy ugyanannyi menet után mindkét algoritmus ugyanabba az állapotba kerül.

4.1.7. állítás. *Tetszőleges r-re, ahol $0 \leq r \leq átm$, mindkét algoritmus r-edik menet utáni állapotában az u, a max_azon, a státus és a menetek értékei megegyeznek.*

A 4.1.7. szimulációs állítás bizonyítása az r szerinti indukcióval történik, ugyanúgy, mintha csak egyetlen algoritmus esetében kellene egy állítást igazolni. Az indukciós lépés érdekes része az, amikor rámutatunk arra, hogy egy folyamatnak a max_azon értéke mindkét algoritmus működése során minden menetben megegyezik.

Vegyünk egy tetszőleges P_i és P_j folyamatot, amelyekre $j \in ki\_szom_i$. Ha az r-edik menet előtt az $új\_info_i$=*igaz*, akkor a P_i folyamat ugyanazt az információt küldi el j-ediknek az OptMaxTerjed algoritmusban, mint amit a MaxTerjed algoritmusban. Másrészről, ha az r-edik menet előtt az $új\_info_i$ = *hamis*, az OptMaxTerjed algoritmusban a P_i folyamat semmit sem küld a P_j-nek, ugyanakkor a MaxTerjed algoritmusban a P_i folyamat elküldi a P_j-nek a max_azon_i-t. Ilyenkor azonban a 4.1.6. állítás következtében az r-edik menet előtt $max\_azon_j \geq max\_azon_i$ áll fenn, így az üzenetnek nincs hatása a MaxTerjed algoritmusban. Ebből következik, hogy a P_i folyamat ugyanolyan hatással van a max_azon_j-re mindkét algoritmusban. Mivel ez minden i és j indexre igaz, bármely folyamatnak a max_azon értéke a két algoritmus működése során minden menetben megegyezik.

Hátra van még a 4.1.5. állítás, amely viszont következik a 4.1.7. és 4.1.1 állításokból. □

Ez a módszer, amit az OptMaxTerjed algoritmus helyességének bizonyítására alkalmaztunk, gyakran használható egy osztott algoritmus „optimalizált" változata helyességbizonyítására. Ilyenkor először az algoritmus egy kevésbé hatékony, egyszerűbb változatának helyességét bizonyítjuk, majd a hatékonyabb, de bonyolultabb változat helyességét az egyszerűbb változattal való formális kapcsolata alapján mutatjuk meg. Ez a kapcsolat szinkron hálózatokra megfogalmazott algoritmusoknál a fenti formában – mindkét algoritmus ugyanazon menetszám utáni állapotára megfogalmazott invariáns állítással – írható fel általánosan.

Másik javítás.. A MaxTerjed algoritmus üzeneteinek száma kisebb mértékben még tovább is csökkenthető. Nevezetesen, ha P_i folyamat egy új maximumot fogad attól a P_j folyamattól, amely egyszerre bejövő és kimenő szomszédja is, azaz amellyel kétirányú kommunikációt folytathat, akkor a P_i-nek nem kell a P_j irányába üzenetet küldenie a következő menetben.

A vezetőválasztás olyan egyedi azonosítókkal ellátott irányított gráfbeli hálózatban is megvalósítható, amelynek sem átmérőjét, sem csúcsainak számát nem ismerjük. Azt javasoljuk az Olvasónak, hogy álljon meg itt, és próbáljon elkészíteni egy ilyen algoritmust. Erre az egyik lehetőség az, hogy olyan kisegítő protokollt vezet be, amely megengedi mindegyik folyamatnak, hogy a hálózat átmérőjét kiszámolja. Felhasználhatja az ebben a fejezetben később található ötleteket is.

4.2.. Szélességi keresés

Most egy olyan szélességi keresést (SZK) mutatunk be, amely egy kitüntetett kezdőcsúcsú, erősen összefüggő irányított gráfra épül. Pontosabban, azt vizsgáljuk meg, hogyan lehet irányított gráfokban *szélességi kereső fát* (SZK fa) felépíteni. Az ilyen fák létrehozását az motiválja, hogy a kommunikáció számára alkalmas szerkezetet találjunk. Az SZK fa minimalizálja a kitüntetett kezdőcsúcsnak megfeleltetett folyamattól a többi folyamat irányába tartó maximális üzenetközvetítési időket (feltételezve, hogy minden egyes él mentén ugyanannyi ideig halad egy-egy üzenet).

Az SZK feladat és annak megoldásai egyszerűbbek azokban az esetekben, ahol a szomszédos csúcsok kétirányú kommunikációval rendelkeznek, azaz ahol a hálózati gráf irányítatlan. Később ezeket az egyszerűsítéseket is be fogjuk mutatni.

4.2.1.. A feladat

A $G = (V, E)$ irányított gráf egy *irányított feszítőfája* olyan gyökeres fa, amely kizárólag a szülőcsúcsból a gyerekcsúcs felé irányuló E-beli irányított élekből áll, és a G összes csúcsát tartalmazza. A G-nek egy i indexű gyökércsúcsú irányított feszítőfája akkor SZK fa, ha az i indexű gyökércsúcstól d távolságra levő minden egyes csúcs a fa d-edik szintjén helyezkedik el (azaz a fában i indexű csúcstól d távolságra). Minden erősen összefüggő irányított gráfnak van egy SZK fája.

Az SZK feladatnál feltételezzük, hogy a hálózat erősen összefüggő, és hogy van egy kitüntetett i_0 indexű kezdőcsúcs. Az algoritmus célja, hogy felderítse a hálózati gráf i_0 gyökerű SZK fájának szerkezetét. Ezt osztott módon tartjuk nyilván, úgy, hogy mindegyik P_{i_0}-tól különböző folyamat rendelkezik majd egy *szülő* összetevővel, amely kijelöli azt a csúcsot, amelyik az ő fabeli szülője.

Mint rendszerint, a folyamatok most is csak az irányított élek mentén küldhetnek egymásnak üzeneteket. A folyamatokhoz egyedi azonosítókat rendelünk, és a folyamatok nem rendelkeznek semmiféle ismerettel a hálózat méretéről vagy átmérőjéről.

4.2.2.. Egy alapvető szélességi kereső algoritmus

Az itt tárgyalt úgynevezett SzinkSZK algoritmus alapötlete megegyezik a szokásos szekvenciális szélességi keresés algoritmusáéval.

SzinkSZK algoritmus

A végrehajtás egy adott pillanatában a folyamatoknak egy bizonyos része „megjelölt”, kezdetben egyedül csak az i_0 indexű. Az i_0-adik folyamat egy `keres`

üzenetet küld az 1. menetben az összes kimenő szomszédjának. Bármelyik menetben, ha egy megjelöletlen folyamat fogad egy `keres` üzenetet, megjelöli magát és szülőnek választ egy folyamatot azok közül, amelyektől a `keres` üzenetet kapta. Közvetlenül azután, hogy a folyamat megjelölt lett, a következő menetben tovább küldi a `keres` üzenetet a kimenő szomszédjainak.

Könnyű megmutatni, hogy a SzinkSZK algoritmus egy SZK fát állít elő. Ahhoz, hogy ezt formálisan is igazoljuk, be kell bizonyítani azt az állítást, amely szerint minden olyan csúcs, amelyik az i_0-tól d ($1 \leq d \leq r$) távolságra van, r menet után már rendelkezik szülő mutatóval; továbbá minden ilyen mutató egy olyan csúcsra mutat, amelynek az i_0-tól vett távolsága $d-1$. Ez az invariáns a szokásos módon, a menetek száma szerinti indukcióval látható be.

Bonyolultságelemzés.. A futási időt meghatározó menetek száma legfeljebb *átm* darab lehet. (A valóságban ezt finomíthatjuk egy kicsit az i_0 indexű csúcsnak a többi csúcstól vett távolságainak maximumára.) Az üzenetek száma éppen $|E|$, hiszen egy `keres` üzenet pontosan egyszer halad át minden irányított élen.

A kommunikációs bonyolultság csökkentése.. Akárcsak a MaxTerjed algoritmusnál, némileg itt is csökkenthetjük az üzenetek számát. Egy újonnan megjelölt folyamatnak ugyanis nem szükséges a `keres` üzenetet azon folyamatokhoz elküldenie, amelyektől már kapott ilyen üzenetet.

Üzenetszórás.. A SzinkSZK-t könnyű úgy kiegészíteni, hogy egy üzenet szétküldését valósítsa meg. Ha egy folyamatnak van egy olyan m üzenete, amelyet közölni akar a hálózat összes folyamatával, magát kezdőcsúcsnak választva végrehajtja a SzinkSZK algoritmust, úgy, hogy az 1. menetben elküldött `keres` üzenet mellé csatolja az m üzenetet is. A többi folyamat ugyancsak hozzákapcsolja az általa elküldött üzenethez az m üzenetet. Mivel a fa az összes csúcsot tartalmazza, az m üzenet minden folyamathoz eljut.

gyerek mutató.. Az SZK probléma egy fontos változata az, amikor megköveteljük, hogy a folyamatok ne csak a fabeli szülőjüket, hanem a gyerekeiket is ismerjék meg. Ebben az esetben arra van szükség, hogy amikor egy folyamat egy `keres` üzenetet fogad, akkor egy `szülő` vagy `nem_szülő` üzenettel feleljen a küldő folyamatnak, hogy ezzel közölje, őt választotta-e szülőjének.

Ha mindegyik szomszédos pár közt kétirányú kommunikációt engedünk meg, azaz a hálózati gráf irányítatlan, akkor ennek a többletkommunikációnak a megvalósítása nem okoz – egy kis költség növekedésen kívül más – problémát. Mivel azonban olyan szomszédos párokat is megengedünk, amelyek között csak egyirányú kapcsolat van, a `szülő` vagy `nem_szülő` üzenetek közül néhányat közvetett úton kell elküldenünk. Ezt megtehetjük például úgy, hogy újra végrehajtjuk a SzinkSZK algoritmust, és a korábban látott módon hozzácsatoljuk a `szülő` vagy `nem_szülő` üzenetet. Ahhoz, hogy egy ilyen üzenetet a megfelelő fogadó folyamat felismerjen, a `keres`-hez csatolt üzenetnek tartalmaznia kell a fogadó egyedi azonosítóját (és egy helyi nevet, amely alapján a fogadó felismeri a küldőt). Megjegyezzük, hogy a SzinkSZK algoritmus ezen végrehajtásainak többsége párhuzamosan is futtatható. Ahhoz, hogy a formális modellünkhöz – amely szerint menetenként legfeljebb egy üzenet küldhető egy kapcsolaton keresztül – illeszkedjünk, több üzenet egyidejű küldése esetén azokat egyetlen üzenetbe kell belefoglalnunk.

A nem kizárólag kétirányú kommunikációt folytató irányított gráfoknál hasznos lehet a gyerekből a szülőbe vezető legrövidebb utat is kiszámoltatni a folyamatokkal a szülő és gyerek mutatókon kívül. Ilyen információ például a SzinkSZK algoritmus újbóli végrehajtásával állítható elő.

Bonyolultságelemzés.. Ha a gráf irányítatlan, az SZK fa meghatározásának futási ideje, beleértve a gyerek mutatók kiszámolását is, $O(átm)$, a kommunikáció bonyolultsága pedig $O(|E|)$.

Még ha valamely szomszédos pár között egyirányú kapcsolat van is, a fának a gyerek mutatókkal együtt való meghatározása $O(átm)$ ideig tart csak, mert a további SZK-kat párhuzamosan is végre lehet hajtani. Ebben az esetben az üzenetek teljes száma $O(átm|E|)$, mert legfeljebb $|E|$ üzenetet lehet menetenként küldeni, a menetek száma pedig $O(átm)$. De mivel egy üzenet akár $|E|$ darab egyidejű SZK végrehajtásáról is tartalmazhat információt, egy üzenetben $|E|b$ bit lehet, ahol a b egy egyszerű egyedi azonosító ábrázolásához szükséges bitek számát jelöli. Ez vezet a kommunikáció bitjeinek teljes $O(átm|E|^2b)$ számához. A bitek teljes számára egy kisebb korlát is kapható, ha észrevesszük, hogy a (legfeljebb $|E|$) darab egyidejű SZK végrehajtása legfeljebb $|E|$ olyan üzenetet használ, amelyek legfeljebb b bitből állnak. Így a kommunikációs bitek teljes száma $O(|E|^2b)$.

Befejeződés.. Hogyan ismerheti fel a P_{i_0} folyamat, hogy a fa építése befejeződött? Ha minden egyes `keres` üzenetre `szülő` vagy `nem_szülő` üzenet a válasz, akkor azután már, hogy egy folyamat minden `keres` üzenetére választ kapott, ismerni fogja az összes fabeli gyerekét, és tudja, hogy azok már mind megjelölődtek. Ezeket az SZK fa leveleitől induló „befejezés-jelzéseket" össze kell gyűjtenünk a kezdőcsúcsban: minden egyes folyamat küld a fabeli szülőjének egy befejezést jelző üzenetet, azután, hogy (a) mindenkitől, akinek a `keres` üzenetet elküldte, választ kapott (így tudja, hogy kik a gyerekei, és hogy azok meg vannak jelölve), és (b) minden gyerekétől megkapta a befejezést jelző üzenetet. Az ilyen típusú eljárást *„üzenetgyűjtés"*-nek nevezzük.

Ha a gráf irányítatlan, az SZK fa előállításának ideje, beleértve a gyerek mutatók beállítását és a befejezést jelző üzeneteknek a kezdőcsúcshoz való visszajuttatását is, $O(átm)$; a kommunikációs bonyolultság pedig csak $O(|E|)$. Ha csak egyirányú kommunikációt engedünk meg, a teljes futási idő, beleszámítva a befejezést jelző üzenetek küldését is, $O(átm^2)$. Ennek a négyzetes viselkedésnek az az oka, hogy befejezés-jelzéseket szintenként egymás után kell beállítani. Az üzenetek teljes száma $O(átm^2|E|)$ és a kommunikációs bitek teljes száma legfeljebb $O(|E|^2b)$.

4.2.3.. Alkalmazások

A szélességi keresés egy alapvető építőeleme az osztott algoritmusoknak. Most néhány példát mutatunk arra, hogyan használhatjuk fel a SzinkSZK algoritmust különféle feladatok megoldására.

Üzenetszórás.. Mint azt korábban már említettük, egy üzenet szétküldését egy SZK fa létesítésével valósíthatjuk meg. Ennek egy másik módja az, hogy először egy gyerek mutatókkal ellátott SZK fát hozzunk létre, úgy, mint azt fent leírtuk, és azután ezt a fát használjuk az üzenetek továbbítására. Csak az üzenetet kell végig terjeszteni a szülőktől azok gyerekeihez. Ez lehetővé teszi, hogy az SZK fa előállítására fordított munkát újra és újra felhasználhassuk, hiszen ugyanabban a fában több üzenet is elküldhető. Az SZK fát egyszer kell csak létrehozni, az egyes üzenetek közvetítésére felhasznált további idő mindössze $O(átm)$, a továbbított üzenetek száma pedig csak $O(n)$.

Globális számítás.. Az SZK fáknak egy másik alkalmazása a hálózaton keresztül történő információgyűjtés, vagy még általánosabban, egy osztott bemenetekre támaszkodó függvény értékének kiszámítása. Vegyük például azt a problémát, ahol minden folyamatnak egy nemnegatív egész bemenő értéke van, és meg akarjuk találni a hálózat bemeneteinek összegét. Ezt egy SZK fa felhasználásával könnyedén (és hatékonyan) megtehetjük, az alábbi módon. A levelektől elindulva összegyűjtjük az eredményeket az

alábbi „üzenetgyűjtő" eljárással. Mindegyik levél elküldi az értékét a szülőjének; mindegyik szülő vár, amíg minden gyerekétől megkapja az értékeket, hozzáadja azokat saját bemenő értékéhez, és elküldi az összeget a saját szülőjéhez. Az SZK fa gyökerében kiszámolt összeg a válasz.

Feltételezve, hogy az SZK fát már korábban létrehoztuk és hogy a fa minden élén kétirányú kommunikációt engedünk meg, ez a séma $O(\mathit{átm})$ idő alatt $O(n)$ üzenetet továbbít. Ugyanez a séma használható sok más függvény, például egész számok maximumának vagy minimumának kiszámolására is. (Az ilyen függvényeknek asszociatívnak és kommutatívnak kell lenniük.)

vezetőválasztás.. A SzinkSZK-t felhasználva olyan algoritmust is tervezhetünk, amely egy egyedi azonosítókat tartalmazó hálózatban, ahol a folyamatok nem ismerik sem az n, sem az *átm* értékét, vezető folyamatot választ. Minden folyamat párhuzamosan elindít egy szélességi keresést. A folyamatok az így felépített fa segítségével az előbb bemutatott globális számítási eljárás keretében határozzák meg a maximális egyedi azonosítót. Az a folyamat, amelyik a maximális egyedi azonosítóval rendelkezik, vezetőnek nyilvánítja magát, a többi pedig nemvezetőnek. Ha a gráf irányítatlan, a futási idő $O(\mathit{átm})$, az üzenetek száma pedig $O(\mathit{átm}|E|)$, aminek oka ismét az, hogy legfeljebb $|E|$ üzenet küldhető az *átm* darab menet mindegyikében. A bitek száma legfeljebb $O(n|E|b)$, ahol b az egyedi azonosító ábrázolásához felhasznált bitek maximális száma.

Az átmérő kiszámítása.. A hálózat átmérője úgy számolható ki, hogy minden folyamat párhuzamosan elindít egy szélességi keresést. A P_i folyamat az így előállított fát arra használja, hogy meghatározza a $\mathit{max\_táv}_i$ értéket, amely a P_i és a hálózat valamelyik másik folyamata közötti távolság legnagyobb értéke. Majd a folyamat újra felhasználja ezt a fát egy olyan globális számításban, amely előállítja a *max_ táv* értékek maximumát. Ha a gráf irányítatlan, a futási idő $O(\mathit{átm})$, az üzenetek száma $O(\mathit{átm}|E|)$, a bitek száma legfeljebb $O(n|E|b)$. Az így kiszámolt átmérőt felhasználhatjuk például a vezetőválasztó MaxTerjed algoritmusban.

4.3.. Legrövidebb utak

Vizsgáljuk meg most az SZK probléma egy általánosítását. Vegyünk ismét egy olyan erősen összefüggő irányított gráfot, ahol a szomszédok között csak egyirányú kapcsolat van. Tegyük fel, hogy mindegyik $e = (i, j)$ irányított él rendelkezik egy nemnegatív súllyal, amelyet *súly(e)*-vel vagy $\mathit{súly}_{i,j}$-vel jelölünk. A feladat az, hogy legrövidebb utat találjunk az irányított gráf egy kitüntetett i_0 indexű kezdőcsúcsából az irányított gráf másik csúcsába. Legrövidebb úton a minimális összsúlyú utat értjük.[2] Az i_0 indexű kezdőcsúcsból az összes többi csúcsba vezető legrövidebb utak együttese egy olyan részfát alkot az irányított gráfban, amelynek minden éle szülőtől gyerek felé irányul.

Egy ilyen fa előállítását – akárcsak a szélességi keresésnél – az indokolja, hogy megfelelő kommunikációs szerkezetet adjunk az üzenetszóráshoz. A súlyok az élek bejárásának költségét, például a kommunikáció idejét vagy árát fejezik ki. A legrövidebb utak fája minimalizálja a kitüntetett folyamatnak a hálózat bármely másik folyamatával történő kommunikációjának költségét.

Feltesszük, hogy kezdetben minden folyamat ismeri mindegyik hozzá kapcsolódó élnek a súlyát, és ezt speciális *súly* változókban tároljuk az él végpontjait adó mindkét folyamatnál. Ugyancsak feltesszük, hogy az egyes folyamatok ismerik az irányított gráf csúcsainak n számát. Megköveteljük azt is, hogy mindegyik folyamat egy részleges legrövidebb utak fájában is meg tudja határozni a szülőjét, és annak az i_0-tól való távolságát (azaz az odavezető legrövidebb út teljes költségét) is.

[2] A súly és a távolság fogalma – a hagyománynak megfelelően – szerencsétlen módon keveredik.

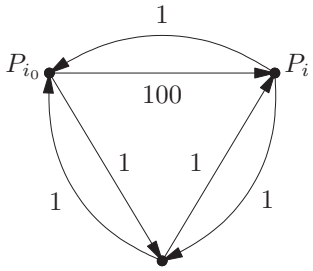
Ha mindegyik él súlya azonos, az SZK fa is egy legrövidebb utak fája. Ebben az esetben az egyszerű SzinkSZK egy nyilvánvaló módosításával elérhető, hogy előállítsuk a távolságokat és a szülő mutatókat.

A különböző súlyú élek esete már érdekesebb. Ezt a problémát például megoldhatjuk a Bellman-Ford legrövidebb utak szekvenciális algoritmusának egy osztott változatával.

BellmanFord algoritmus
Minden P_i folyamat nyilvántartja a *táv* összetevőjében az adott pillanatig megismert P_{i_0}-tól vett legrövidebb távolságot és az ahhoz tartozó *szülő*-t, azaz azt a bejövő szomszédos folyamatot, amely a P_i-t közvetlenül megelőzi a *táv* távolságú úton. Kezdetben $táv_{i_0} = 0$, minden $i \neq i_0$ folyamatra $táv_i = \infty$, a *szülő* összetevő értéke pedig nem ismert. Az egyes menetekben mindegyik folyamat elküldi a *táv* értékét az összes kimenő szomszédjának. Azután „egy enyhítő lépésben" egy P_i folyamat megváltoztatja a *táv* értékét annak korábbi értéke és az összes a $táv_j + súly_{j,i}$ érték (ahol j index egy bejövő szomszédot jelöl) közül a legkisebbre. Ha a *táv* változik, a *szülő* összetevőt is frissíteni kell. Az $n-1$-edik menet után a *táv* tartalmazni fogja a legrövidebb távolságot, a *szülő* összetevő pedig a legrövidebb utak fájában szereplő szülőt.

Könnyű belátni, hogy $n-1$ menet után a *táv* értékek a helyes távolságokat adják. A BellmanFord algoritmus helyességét például r szerinti indukcióval igazolhatjuk. Belátható ugyanis, hogy az r-edik menet után minden P_i folyamat *táv* és *szülő* összetevője megfelel egy olyan legrövidebb útnak, amely P_{i_0}-ból P_i-be vezet és legfeljebb r darab élt tartalmaz. (Ha nincs ilyen út, a $táv=\infty$ és a *szülő* nem meghatározott.) A részletek kidolgozását meghagyjuk gyakorlatnak (lásd 4-14. gyakorlat.

Bonyolultságelemzés.. A BellmanFord algoritmus futási ideje $n-1$, az üzenetek száma $(n-1)|E|$.



4.1.. ábra. A legrövidebb utak csak 2 menet után ismerhetők fel, pedig az $átm = 1$.

4.3.1. példa. A BellmanFord futási ideje
A SzinkSZK algoritmus futási idejét alapul véve azt gyaníthatnánk, hogy a BellmanFord algoritmus futási ideje valójában *átm*. A 4.1. ábrán látható példa azt mutatja, hogy ez nincs így. Ebben a példában ugyanis 2 menet kell ahhoz, hogy a P_{i_0}-ból P_i-be vezető út helyes távolsága, a 2 beállítódjon, mivel az út, amely mentén ez a távolság kialakul, két élből áll. Az átmérő ellenben csak 1.

A BellmanFord algoritmus az n helyett annak felső korlátjával is dolgozhat. Ha nem ismerünk ilyen korlátot, a 4.2. alfejezetben bemutatott módszert használhatjuk.

4.4.. Minimális feszítőfa

Most egy élsúlyozott irányított gráfban fogunk minimális (súlyú) feszítőfát (MFF) keresni. Egy ilyen fát az üzenetszórásnál használhatunk fel. Egy minimális súlyú feszítőfa minimalizálja annak a kommunikációnak a teljes költségét, amelyet egy tetszőleges forrásfolyamat a hálózat összes többi folyamatával folytat.

4.4.1.. A feladat

Egy $G = (V, E)$ irányítatlan gráf egy *feszítőerdeje* olyan erdő (azaz körmentes, de nem feltétlenül összefüggő gráf), amely kizárólag E-beli élekből áll és minden G-beli csúcsot tartalmaz. A G irányítatlan gráf egy *feszítőfája* G egy összefüggő feszítőerdeje***. Ha az irányítatlan E-beli élekhez súlyokat rendelünk, akkor G bármely részgráfjának (feszítőfájának vagy feszítő erdejének) súlyán a benne levő élek súlyának összegét értjük.

Emlékezzünk arra, hogy modellünkben az irányítatlan gráfokat olyan irányított gráfoknak tekintjük, ahol a szomszédos csúcsokat kétirányú élek kötik össze. A 4.3. alfejezethez hasonlóan feltesszük, hogy mindegyik $e = (i, j)$ irányított él rendelkezik egy nemnegatív valós értékű súllyal, $súly(e) = súly_{i,j}$, és tegyük fel azt is, hogy minden i és j indexű csúcsra $súly_{i,j} = súly_{j,i}$. Feltesszük továbbá, hogy kezdetben minden folyamat ismeri mindegyik hozzá kapcsolódó élnek a súlyát, és ezt speciális *súly* változókban tároljuk az él végpontjait adó mindkét folyamatnál. Feltesszük, hogy a folyamatok rendelkeznek egyedi azonosítókkal, és ismert a csúcsok n száma is. A feladat az, hogy találjunk egy minimális súlyú (irányítatlan) feszítőfát a teljes hálózatra; konkrétabban, minden folyamatnak el kell tudnia dönteni, hogy a bevezető élei közül melyik tartozik a minimális feszítőfához, és melyik nem.

4.4.2.. Elméleti háttér

Minden ismert MFF algoritmus, a szekvenciálisak éppúgy, mint a párhuzamosak, ugyanarra a most bemutatandó ötletre épülnek. A minimális feszítőfák felépítésének alapmódszere az n darab, egyetlen csúcsból álló triviális feszítőerdőből indul ki, majd újra és újra összeköti az erdő két összetevőjét egy megfelelő éllel egészen addig, amíg egy feszítőfát nem hoz létre. Ahhoz, hogy a végén minimális feszítőfát kapjunk fontos, hogy mindig azt az élt válasszuk ki az összekötéshez, amely egy összetevő kivezető élei közül a minimális súlyú. Ennek a kiválasztásnak az igazolását a következő lemma adja.

4.3. lemma. . *Legyen $G = (V, E)$ egy élsúlyozott irányítatlan gráf, és legyen $(V_i, E_i) : 1 \leq i \leq k$ a G egy feszítőerdeje, ahol $k > 1$. Rögzítsünk egy tetszőleges i-t $(1 \leq i \leq k)$. Legyen e az*

$$\{e' : e'\text{-nek pontosan egyik végpontja van } V_i\text{-ben}\}$$

halmaz legkisebb súlyú éle.

Ekkor van a G-nek egy olyan feszítőfája, amely a $\bigcup_j E_j$ mellett magában foglalja az e-t is, továbbá ez a fa minimális súlyú azon fák között, amelyek az $\bigcup_j E_j$-t tartalmazzák.

Bizonyítás. Bizonyítsunk indirekt módon. Tegyük fel, hogy létezik olyan T feszítőfa, amely tartalmazza $\bigcup_j E_j$-t, de nem tartalmazza e-t, ugyanakkor szigorúan kisebb súlyú, mint bármelyik más $\bigcup_j E_j$-t és az e-t tartalmazó feszítőfa. Jelöljük T'-vel azt a gráfot, amelyet úgy kapunk, hogy hozzávesszük T-hez az e-t. Világos, hogy T' tartalmaz egy olyan kört, amelynek van olyan V_i-ből kivezető e' éle, amely az e-től különbözik.

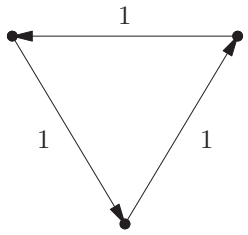
Az e választása miatt *súly*$(e') \geq$ *súly*(e). Készítsük most el a T'' gráfot, úgy, hogy a T'-ből töröljük az e'-t. Ekkor a T'' a G-nek egy olyan feszítőfája, amely tartalmazza $\bigcup_j E_j$-t és az e-t, de a súlya nem nagyobb, mint a T súlya. Ez azonban ellentmond az indirekt feltevésnek. □

A 4.3. lemma igazolja az alábbi MFF-t építő általános módszer helyességét.

Általános módszer az MFF-hez
Induljunk ki az n darab, egyetlen csúcsot tartalmazó triviális feszítőerdőből. Ezután ciklikusan hajtsuk végre a következőt: válasszunk az erdőből egy tetszőleges C összetevőt, és egy minimális súlyú, C összetevőből kivezető e élt. Vonjuk össze a C összetevőt, az e élt, és az e él végpontján levő másik összetevőt egy új összetevőbe. Akkor álljunk le, ha már csak egyetlen összetevőnk maradt.

A 4.3. lemma felhasználható annak induktív bizonyításához, hogy a fenti eljárás minden lépésében a nyilvántartott erdő egy részgráfja valamelyik MFF-nek. Több jól ismert szekvenciális MFF algoritmus is ennek az általános stratégiának az egyedi esete. Például a *Prim-Dijkstra algoritmus* kiválaszt a kiinduló (egyetlen csúcsból álló) összetevők közül egyet, és ezt bővíti újra és újra az abból kiinduló legkisebb súlyú éllel, amíg egy teljes feszítőfát nem kap. Másik példa a *Kruskal algoritmus*, amely minden lépésben az aktuális feszítőerdő összetevőit összekötő legkisebb súlyú élt választja ki, ennek segítségével egyesít két összetevőt egészen addig, amíg egyetlen összetevő nem marad, amely már a végleges feszítőfa lesz.

Ahhoz, hogy ezt az általános stratégiát osztott módon alkalmazzuk, az lenne jó, ha az erdőt párhuzamosan egyszerre több éllel is bővíthetnénk. Azaz az összetevők mindegyike egymástól függetlenül meghatározhatná a saját minimális súlyú kimenő élét, amelyeket aztán hozzáadhatnánk az erdőhöz, és ez így egyidejűleg több összetevőpár egyesítését foglalná magába. A 4.3. lemma azonban nem garantálja az ilyen párhuzamos stratégia helyességét. Sőt általában ez a stratégia nem is helyes.



4.2.. ábra. Minimális súlyú kivezető élek párhuzamos kiválasztása kört eredményez.

4.4.1. példa. Kör keletkezése egy párhuzamos MFF algoritmusban
Vegyük a 4.2. példa gráfját. A csúcsok a feszítőfa összetevőit ábrázolják. A három 1 súlyú él az összes kimenő él. Ha az összetevők kiválasztanák a nyilak által megjelenített legkisebb súlyú kimenő élüket, kör keletkezne.

Abban a különleges esetben elkerülhetők a körök, amikor minden élnek eltérő súlya van. Ez az alábbi lemma miatt van így.

4.4. lemma. . *Ha egy G gráf minden éle különböző súllyal rendelkezik, akkor G-nek pontosan egy MFF-je van.*

Bizonyítás. A bizonyítás a 4.3. lemma bizonyításához hasonlóan történik. Feltesszük, hogy van két különböző minimális súlyú feszítőfa, T és T', és legyen e az a minimális súlyú él, amely a két fa közül csak az egyikben jelenik meg. Tegyük fel (az általánosság megsértése nélkül), hogy $e \in T$. Ekkor az e-nek T'-höz adásával egy T'' kört tartalmazó gráfot kapunk, és ebben a körben legalább egy olyan e' él is van, amely nem szerepel a T-ben. Mivel az élsúlyok mind különböznek és az e' a két fa közül csak az egyikben szerepel, az e választása miatt biztosan fennáll, hogy $súly(e') > súly(e)$. Az e'-nek T''-ből való eltávolítása egy olyan feszítőfát eredményez, amely kisebb súlyú, mint a T', és ez ellentmond az indirekt feltevésnek. □

Most vizsgáljuk felül az általános stratégiát abban az esetben, amikor a gráf különböző élsúlyokat tartalmaz, és így a 4.4. lemma szerint egyetlen kizárólagos MFF-je van. Ilyenkor a feszítőfa építése során az erdő bármelyik összetevőjének pontosan egy minimális súlyú kimenő éle van (amelyet a kiejthetetlen MSKÉ-vel rövidítünk). A 4.3. lemmából következik, hogy ha egy olyan erdővel van dolgunk, amelynek minden éle a kizárólagos MFF-ben van, az összes összetevő MSKÉ-je is a kizárólagos MFF-hez tartozik. Így mindet azonnal hozzáadhatjuk annak a veszélye nélkül, hogy kört hoznánk létre.

4.4.3.. Algoritmus

Most a fent leírt általános módszert követve bemutatunk egy tetszőleges élsúlyozott irányítatlan gráfban MFF-et építő osztott algoritmust. Mivel az összetevőket párhuzamosan egyesíthetjük, feltesszük, hogy az élek különböző súlyúak; majd ennek a pontnak a végén megmutatjuk azt is, hogyan lehet ezt a feltételt elhagyni. Az algoritmust SzinkGHS-nek hívjuk, mert ez a Gallager, Humblet, és Spira által kifejlesztett aszinkron algoritmusra épül. (Majd a 15.5. alfejezetben fogjuk bemutatni az egyszerűen GHS-nek nevezett aszinkron algoritmust.)

SzinkGHS algoritmus

Az algoritmus "szintről szintre" építi a feszítőerdőnek (az MFF részfáiként megjelenő) összetevőit. Minden k-ra, a k szint összetevőinek legalább 2^k csúcsa van. Az egyes összetevők minden szinten rendelkeznek egy kitüntetett vezető csúccsal. A folyamatok minden szintet rögzített számú, $O(n)$ menetben befejődnek.

Az algoritmus egyetlen csúcsból álló, éleket nem tartalmazó 0 szintű összetevőkkel indul. Tegyük fel, hogy a k szintű összetevőket (a vezető csúcsaikkal együtt) már meghatároztuk. Részletesebben, tegyük fel, hogy minden folyamat ismeri saját összetevője vezetőjének egyedi azonosítóját; és ezek az egyedi azonosítók az egyes összetevőket is azonosítják. Minden folyamat tudja, hogy a hozzá kapcsolódó élek közül melyik tartozik a folyamatot tartalmazó összetevőbe.

A $k+1$ szintű összetevőket úgy kapjuk meg, hogy minden k szintű összetevő megkeresi a saját MSKÉ-jét. A vezető folyamat a 4.2. alfejezetben leírt üzenetszóró stratégia alapján indítja el ezt a keresést. Minden folyamat kiválasztja azt a legkisebb súlyú élt, amelyik kivezet a folyamatot tartalmazó összetevőből (ha van egyáltalán ilyen); ezt úgy érik el, hogy egy `teszt` üzenetet küldenek minden nem az összetevőhöz tartozó él mentén, hogy megtudják, annak másik végpontja ugyanabban az összetevőben található-e. (Ez az összetevők azonosítóinak összehasonlításával eldönthető.) Ezután a folyamatok az üzenetgyűjtés elvén továbbítják az egyes csúcsokból kivezető minimális súlyú élről származó információt, úgy, hogy összevetik annak súlyát a hozzájuk érkező többi élsúllyal, és mindig a minimális

súlyút küldik tovább a vezető folyamat felé. A vezető folyamat által előállított minimum a szóban forgó összetevő MSKÉ-je.

Amikor minden k szintű összetevő megtalálta a saját MSKÉ-jét, az összetevőket ezen MSKÉ-k mentén egyesítjük, ezáltal megkapjuk a $k+1$ szintű összetevőket. Ez magában foglalja, hogy minden k szintű összetevő vezető folyamata kapcsolatba lép az MSKÉ végpontjához tartozó folyamattal, hogy az az új összetevőhöz tartozóként jelölje meg az élt.

Ezután a $k+1$ szintű összetevőkben új vezető folyamatot kell választani a következő módon: megmutatható, hogy a k szintű öszetevők minden $k+1$ szintű összetevőbe egyesítendő csoportjában egyetlen olyan e él van, amely közös MSKÉ-je a csoportba tartozó két k szintű összetevőnek. (Ezt később látjuk be.) Legyen az új vezető folyamat ennek az e élnek a nagyobb egyedi azonosítójú végpontja. Megjegyezzük, hogy ez az új vezető folyamat a helyi információkra támaszkodva azonosíthatja magát.

Végül az új vezető folyamat üzenetszórás útján elterjeszti az egyedi azonosítóját az új összetevőben.

Néhány szint után a feszítőerdő a hálózat összes csúcsát tartalmazó egyetlen összetevőből áll majd. Ezután már nem lehet MSKÉ-t találni, mert egyik folyamatnak sincs már az összetevőből kivezető éle. Amikor a vezető folyamat megtudja ezt, szétsugároz egy üzenetet, amely elmondja, hogy az algoritmus befejeződött.

Az algoritmus kulcsa az, hogy a k szintű összetevők egyesítendő csoportjában egyetlen olyan (irányítatlan) él található, amely a végpontjait tartalmazó összetevőknek közös MSKÉ-je. Ahhoz, hogy lássuk, miért van ez így, bevezetjük az *összetevők irányított gráfját*, a G'-t, amely csúcsai azok a k szintű összetevők, amelyeket egy $k+1$ szintű összetevő előállításához használunk fel, élei pedig az MSKÉ-k. G' egy gyengén összefüggő irányított gráf, amelyben minden csúcsnak pontosan egy kivezető éle van. (Egy irányított gráf *gyengén összefüggő*, ha az irányítatlan változata, amelyet úgy kapunk, hogy figyelmen kívül hagyjuk az éleinek irányítását, összefüggő.) Felhasználhatjuk az alábbi tulajdonságot.

4.5. lemma. . *Legyen G egy gyengén összefüggő irányított gráf, amelyben minden csúcsnak pontosan egy kivezető éle van. Ekkor a G pontosan egy kört tartalmaz.*

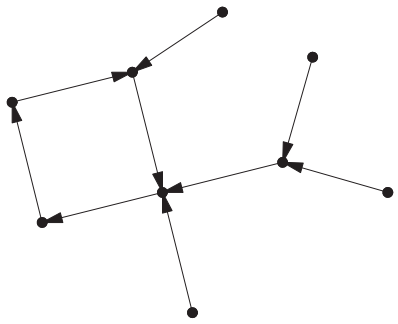
Bizonyítás. A bizonyítást meghagyjuk gyakorlatnak (lásd 4-16. gyakorlat). □

4.4.2. példa. Csúcsonként egy kivezető élt tartalmazó gráf

A 4.3. ábra egy olyan gráfot mutat, amelynek minden csúcsa pontosan egy kivezető élt tartalmaz.

Alkalmazzuk a 4.5. lemmát az összetevők irányított G' gráfjára, hogy megkapjuk az összetevők egyetlen körét. A G' előállítása miatt a kör egymást követő éleinek súlya nem növekszik; ezért az ilyen körnek a hossza nem lehet 2-nél nagyobb. Így az egyetlen kör pontosan 2 hosszú. Ez azonban egy élnek felel meg, amely mindkét szomszédos összetevőnek a közös MSKÉ-je.

A SzinkGHS algoritmus kritikus pontja a szintek szinkronizálása. Ehhez azt kell biztosítanunk, hogy amikor a P_i folyamat megpróbálja meghatározni, hogy a P_j folyamattal közös összetevőben vannak-e, akkor mindkét folyamat már az aktuális összetevő-azonosítóval rendelkezzen. Ha a P_j folyamat azonosítója különbözik a P_i azonosítójától, biztosak szeretnénk lenni abban, hogy P_i-edik és P_j-edik tényleg különböző összetevőben



4.3.. ábra. Egy gráf, amelyben minden egyes csúcs pontosan egy kivezető élt tartalmaz. Megfigyelhetjük az egyetlen kört is.

van, és nem csak arról van szó, hogy még nem kapták meg a vezetőjüktől az összetevőjük azonosítóját. Ahhoz, hogy a szinteket szinkronizáltan hajtsuk végre, a folyamatok az egyes szinteken előre meghatározott számú menetet engednek meg. Ahhoz, hogy a menetek minden számítása biztosan befejeződjön, ez a szám legyen $O(n)$; megjegyezzük, hogy a $O(átm)$ nem mindig elegendő, és kizárólag emiatt kell a csúcsoknak ismerni az n-t. (A 15.5. alfejezetben, amikor az aszinkron hálózatokban újra megvizsgáljuk ezt az algoritmust, az összetevők szinkronizálására ettől eltérő módszert fogunk használni.)

Bonyolultságelemzés.. Figyeljük meg először, hogy minden k szintű összetevő legalább 2^k csúcsot tartalmaz. Ezt indukcióval lehet belátni, úgy, hogy észrevesszük, egy összetevőt minden szinten ugyanazon szint legalább egy másik összetevőjével egyesítünk. Így a szintek száma legfeljebb $\log\ n$. Mivel mindegyik szint $O(n)$ ideig tart, ebből az következik, hogy a SzinkGSH futási ideje $O(n\ \log\ n)$. A kommunikáció bonyolultsága $O((n + |E|) \cdot \log\ n)$, mert mindegyik szinten $O(n)$ üzenetet kell a fa összes élén végig küldeni, és $O(|E|)$ további üzenet szükséges az adott csúcsból kivezető legkisebb súlyú élek megtalálásához.

Üzenetek számának csökkentése.. Az üzenetek számát az adott csúcsból kivezető minimális súlyú élek óvatosabb stratégiájú keresésével $O(n\ \log\ n + |E|)$-re csökkenthetjük. Ez a javítás a futási idő növekedéséhez vezet, bár annak nagyságrendje nem nő. Az ötlet az alábbi:

Minden egyes folyamat jelölje meg a hozzá tartozó élt az „elutasítva” címkével, amikor kiderül, hogy ugyanazon összetevőhöz tartozó csúcshoz vezet; így nincs szükség arra, hogy ezeket később ismét megvizsgáljuk. Az egyes szinteken csak a többi élt teszteljük egyenként a súlyuk szerint növekedő sorrendben, amíg az első olyat meg nem találjuk, amely kivezet az összetevőből (vagy amíg az élek el nem fogynak).

E javítás mellett a fa élein küldött üzenetek száma továbbra is $O(n\ \log\ n)$. Nézzük most meg az adott csúcsokból kivezető minimális súlyú élek megtalálásának amortizációs elemzését. Minden él legfeljebb egyszer válik teszteltté és elutasítottá, ez összesen $O(|E|)$. Egy adott csúcsból kivezető, már tesztelt minimális súlyú élt, amely nem a szóban forgó összetevő MSKÉ-je, újra lehet tesztelni, de minden szint minden csúcsánál legfeljebb egy ilyen vizsgálatra kerülhet sor, ami $O(n\ \log\ n)$. A teljes kommunikációs bonyolultság így $O(n\ \log\ n + |E|)$.

A most vázolt módszernek van egy másik előnye. Mivel az egyes csúcsok megjelölik mind a hozzájuk tartozó MFF-be kerülő éleket, mind azokat, amelyek nincsenek a fában, nincs szükség arra, hogy a végső fázisban a vezető folyamat jelezze mindenkinek az

algoritmus befejezését. Mindegyik csúcs egyszerűen tartalmazza a hozzá tartozó élekkel kapcsolatos információt.

Nem feltétlenül különböző élsúlyok.. Vizsgáljuk most meg az MFF problémát egy olyan gráfban, ahol az élsúlyok nem feltétlenül különbözők. Ebben az esetben egy kis módosítással alkalmazhatjuk a SzinkGHS algoritmust. Vegyük először észre, hogy a SzikGHS algoritmus a súlyokat csak a $<, >, =$ összehasonlításoknak veti alá.

Tetszőleges élsúlyok esetén könnyen származtathatunk a csúcsok egyedi azonosítójából „élazonosítókat". Egy (i, j) él azonosítója a $(súly_{i,j}, v, v')$ hármas lesz, ahol v és v' az i és j indexű csúcsok egyedi azonosítói, és $v < v'$. (Így (i, j) és (j, i) ugyanazzal az azonosítóval bír.) A hármasok lexikografikus rendezése teljes rendezést határoz meg az élazonosítók között.

Mivel a SzinkGHS algoritmus csak összehasonlításra használja a súlyokat, így azt a valódi élsúlyok helyett az élazonosítókkal is lefuttathatjuk; ez ugyanazt a végrehajtást eredményezi, mint ha a SzinkGHS algoritmust az ugyanúgy rendezett egyedi élsúlyok halmazára futtatnánk le. Ennek eredményeként egy fát kapunk. Egy gyakorlatra hagyjuk annak az igazolását, hogy ez a fa valóban az eredeti gráf egy MFF-je (lásd 4-18. gyakorlat).

Vezető folyamat választása.. Ha egyszer egy irányítatlan gráfra épülő hálózatban egy MFF-t (vagy bármilyen feszítőfát) már megismertünk, könnyű egy egyedi vezető folyamatot választani a rendelkezésre álló egyedi azonosítók segítségével. Nevezetesen a feszítőfa levelei egy üzenetet indítanak el a fa útjain a gyökér felé; mindegyik belső csúcs legalább az egyik szomszédjára vár, mielőtt üzenetet küldene a többi szomszédjához. Ha egy csúcs az összes szomszédjától anélkül kap üzenetet, hogy üzenetet küldött volna, vezetőnek nevezi ki magát. Ha két szomszédos csúcs ugyanabban a menetben egymástól kap üzenetet, akkor az egyikük, mondjuk a nagyobb egyedi azonosítójú, nyilvánítja magát vezetőnek. A vezetőfolyamat-választás teljes többlet futási ideje (az MFF felépítés után) csak $O(n)$, és ehhez $O(n)$ üzenet küldése szükséges.

Összesítve ezt az MFF bonyolultságelemzésével, azt látjuk, hogy egy élsúlyozott irányítatlan gráffal indulva, amelyben a csúcsok ismerik az n-t (de az *átm* értékét nem), $O(n \log n)$ idő alatt lehet vezetőt választani $O(n \log n + |E|)$ üzenet küldése mellett.

4.5.. Maximális független halmaz

Az utolsó probléma, amivel ebben a fejezetben foglalkozunk, az irányítatlan gráfok csúcsai közül történő *maximális független halmaznak (MFH)* a kiválasztása. Csúcsoknak egy halmazát akkor nevezzük *független halmaznak*, ha nem tartalmaz szomszédos csúcsokat; egy független halmazt *maximálisnak* mondunk, ha nem lehet újabb csúcs hozzáadásával nagyobb független halmazt készíteni belőle. Megjegyezzük, hogy az irányítatlan gráfoknak számos különböző maximális független halmazuk lehet. Nincs szükségünk a lehető legnagyobb maximális független halmazra, bármelyik megteszi.

Az MFH feladat aktualitását az a probléma adja, hogyan foglaljuk le osztott erőforrásainkat a hálózatbeli folyamatok számára. A G gráf szomszédai képviselik azokat a folyamatokat, amelyek nem tudják egyidőben végrehajtani valamelyik osztott erőforrást igénylő tevékenységüket (ez lehet például adatbázis elérése vagy rádiós műsorszórás). Ki szeretnénk választani azon folyamatok halmazát, amelyek egyidejű végrehajtása megengedett; ahhoz, hogy az ütközéseket elkerüljük, ezeket a folyamatokat a G egy független halmazába gyűjtjük össze. Nyilván nem kívánatos kizárni ebből egyik folyamatot sem, amelynek egyik szomszédja sem aktív; ezért a kiválasztott halmaznak maximálisnak kellene lennie.

4.5.1.. A feladat

Legyen $G = (V, E)$ egy irányítatlan gráf. A csúcsok egy $I \subseteq V$ halmazát *függetlennek* mondjuk, ha minden $i, j \in I$-re $(i, j) \notin E$. Egy független I halmaz *maximális*, ha bármely olyan I' halmaz, amelynek valódi része az I, nem független. A cél, hogy meghatározzuk G egy maximális független halmazát. Minden folyamatnak, amelynek indexe az I-ben van, végül *győztessé* kell válnia, és mindegyik I-n kívüli indexű folyamatnak *vesztessé*.

Feltesszük, hogy a csúcsok számát, az n-t, az összes folyamat ismeri. (Használhatnánk helyette az n egy felső korlátját is.) A csúcsok most nem rendelkeznek egyedi azonosítókkal.

4.5.2.. Véletlenített algoritmus

Könnyű megmutatni, hogy azon gráfokban, ahol a csúcsokhoz rendelt folyamatok determinisztikusak, az MFH problémát nem lehet megoldani. Ennek igazolása a 3.1. tétel mintájára történik. Ebben az alfejezetben egy olyan egyszerű megoldást mutatunk, amely a véletlenszerűséget használja fel ahhoz, hogy leküzdje ezt a determinisztikus rendszerekben rejlő akadályt. Legyünk azonban korrektek, és ne felejtsük el megemlíteni azt sem, hogy van (nulla) valószínűsége annak, hogy a véletlenített algoritmus nem fog befejeződni. Ezt az algoritmust feltalálója, Luby után, LubyMFH algoritmusnak nevezzük.

A LubyMFH azon az iterációs sémán alapul, amely az adott G gráfból egy tetszőleges nemüres független halmazt választ ki, majd ennek a halmaznak a csúcsait és azok minden szomszédját eltávolítja a gráfból, és ezt a folyamatot ciklikusan ismételi. Ha W egy gráf csúcsainak egy részhalmaza, akkor a $szom(W)$ a W-beli csúcsok szomszédainak halmazát jelöli.

> Legyen a *gráf* a *csúcsok* és az *élek* mezők alkotta rekord,
> amelyet az eredeti G gráf alapján töltünk fel. Legyen I egy csúcsokat
> tartalmazó halmaz, amely kezdetben üres.

while *gráf.csúcsok* $\neq \emptyset$ **do**
  kiválaszt egy független nemüres $I' \subseteq$ *gráf* halmazt a *gráf*-ban
  $I := I \cup I'$
  gráf := a *gráf.csúcsok* $- I' -$ *gráf_szom(I')* által indukált *gráf*-beli részgráf [a]
end while

[a] Egy G gráf csúcsainak egy W részhalmaza által indukált részgráfján azt a gráfot értjük, amelynek csúcshalmaza a W, élei pedig a G gráf W-beli csúcsait összekötő élek.

Könnyű igazolni, hogy ez a séma egy maximális független halmazt állít elő. Ez a halmaz független, mert a kiválasztott I' halmaz minden szakaszban független, és ténylegesen kivesszük a maradék gráfból az összes olyan csúcs szomszédját, amely az I halmazba esik. Ez a halmaz maximális, hiszen csak olyan csúcsokat zárunk ki a további vizsgálatokból, amelyek az I halmazba került csúcsok szomszédjai.

Ennek az általános sémának egy osztott hálózatban történő megvalósításánál az a kulcskérdés, hogyan választható ki az egyes iterációkban az I' halmaz. Ehhez használjuk a véletlenített módszert. Minden szakaszban minden P_i folyamat kiválaszt egy egyenletes eloszlású véletlen *érték*$_i$ egész számot az $\{1, ..., n^4\}$ készletből. Az n^4 használatát az indokolja, hogy elég nagy ahhoz, hogy a gráf összes folyamata nagy valószínűséggel

különböző számot válasszon. (Ebben a könyvben nem térünk ki e valószínűség kiszámolására, helyette felhívjuk a figyelmet Luby cikkére.) Miután a folyamatok kiválasztják ezeket az értékeket, az I'-be, definíció szerint, azok az i indexű csúcsok kerülnek, amelyek helyi győztesek, azaz a i csúcs minden j indexű szomszédos csúcsára fennáll, hogy $érték_i > érték_j$. Ez nyilvánvalóan egy független halmazhoz vezet, mivel két szomszédos csúcs egyszerre nem győzheti le egymást.

Ebben a megvalósításban előfordulhat, hogy a véletlen választások olyan szerencsétlenül alakulnak, hogy az I' halmaz néhány szakaszon keresztül üres lesz; ezek a szakaszok tehát haszontalanok, hiszen semmit sem csinálnak. Feltéve, hogy az algoritmus nem ér el egy olyan ponthoz, amelytől kezdve csupa ilyen haszontalan szakaszokat hajt végre, egyszerűen figyelmen kívül hagyjuk a haszontalan szakaszokat és kikötjük, hogy a LubyMFH algoritmus helyesen kövesse az általános sémát. Az elemzésbe azonban bele fogjuk számolni a haszontalan szakaszokat is. Az algoritmus az alábbi.

LubyMFH algoritmus (vázlatosan)

Az algoritmus működése szakaszokra bontható, minden szakasz három menetből áll.

1. menet: Egy szakasz első menetében a folyamatok kiválasztanak maguknak egy saját *érték*-et, és elküldik azt minden szomszédjuknak. Az 1. menet végére, amikor az összes *érték* célba ér, a (szomszédjaiknál nagyobb értéket választó) győztesek – azaz az I'-beli folyamatok – ismerni fogják magukat.

2. menet: A második menetben a győztesek megismerik a szomszédjaikat. A 2. menet végére a vesztesek – azaz azok a folyamatok, amelyeknek van I'-beli szomszédjuk – meghatározzák magukat.

3. menet: A harmadik menetben mindegyik vesztes megismeri a szomszédjait. Aztán minden érintett folyamat – győztesek, vesztesek, és a vesztesek szomszédjai – törli a megfelelő csúcsot és az ahhoz tartozó éleket a gráfból. Pontosabban, a győztesek és vesztesek a további szakaszokban már nem vesznek részt, és a vesztesek szomszédjai törölnek minden olyan élt, amely a most törölt csúcsokhoz vezet.

Most a fejezetben megszokott módon formalizáljuk az algoritmust. Mint azt a 2.7. alfejezetben már leírtuk, minden folyamat egy speciális $véletlen_i$ véletlen függvényt használ, amelyet minden menetben az $üzenetek_i$ és a $átmenet_i$ függvények végrehajtását megelőzően futtat le. Itt a *random* az $\{1, \ldots, n^4\}$-ból egyenletes eloszlású, véletlenül kiválasztott értéket jelöli.

LubyMFH algoritmus (formálisan)

állapotok$_i$:
menet$\in \{1, 2, 3\}$, kezdetben 1
érték $\in \{1, \ldots, n^4\}$, kezdetben tetszőleges
éber egy logikai változó, kezdetben *igaz*
marad_szom csúcsok egy halmaza,
kezdetben az eredeti G gráfbeli összes szomszédja
státus $\in \{$*ismeretlen, győztes, vesztes*$\}$, kezdetben *ismeretlen*

véletlen$_i$:
if *éber* és $menet = 1$ **then** *érték* := *random*

üzenetek$_i$:
if *éber* **then**
  case
    $menet = 1$:
      send *érték* a *marad_szom* minden csúcsához
    $menet = 2$:
      if *státus* = *győztes* **then**
        send *győztes* a *marad_szom* minden csúcsához
    $menet = 3$:
      if *státus* = *vesztes* **then**
        send *vesztes* a *marad_szom* minden csúcsához
  endcase

Az alábbi kódban a 3 azonos a 0-val modulo 3.

átmenet$_i$:
if *éber* **then**
  case
    $menet = 1$:
      if *érték*$> v$ minden bejövő v értékre **then** *státus*:=*győztes*
    $menet = 2$:
      if egy *győztes* üzenet érkezett **then** *státus* := *vesztes*
        send *győztes* a *marad_szom* minden csúcsához
    $menet = 3$:
      if *státus* $\in$ {*győztes, vesztes*} **then**
        éber := *hamis*
        marad_szom :=
          marad_szom $-$ {j : j-től egy *vesztes* üzenet érkezett}
  endcase
$menet := (menet + 1)$ modulo 3

Megjegyezzük, hogy a LubyMFH algoritmus akkor is helyesen működik, ha egy-két szakaszban néhány szomszédos folyamat ugyanazon véletlen értékeket választja.

4.5.3.. Elemzés*

Azt már korábban megvitattuk, hogy a LubyMFH algoritmus, feltéve, hogy nem akad el a haszontalan szakaszok újra és újra történő végrehajtásával, egy MFH-t állít elő. Azt állítjuk, hogy az algoritmus egy valószínűséggel nem akad el. Pontosabban, az algoritmus egy tetszőleges szakaszában a megmaradt gráfból eltávolított élek számának várható értéke legalább akkora, mint a megmaradt élek számának egy állandó hányada. Ez maga után vonja azt is, hogy egy szakaszban az élek legalább egy rögzített hányadának eltávolítására egy állandó valószínűséggel kerül sor. Az is következik, hogy a befejeződésig végrehajtott menetek számának várható értéke $O(\log\ n)$, továbbá, hogy az algoritmus egy valószínűséggel befejeződik.

A LubyMFH algoritmus teljes elemzése Luby eredeti cikkében található. Most bizonyítás nélkül mutatjuk be a legfontosabb lemmákat, és jelezzük, hogyan lehet azokat a szükséges eredmények bizonyításához felhasználni. A következő három lemmához rögzítünk egy $G = (V, E)$ gráfot, amelynek tetszőleges $i \in V$ csúcsára határozzuk meg az alábbi összeget, ahol $d(j)$ a G gráf j-edik csúcsának fokszáma.

$$sum(i) = \sum_{j \in szom_i} \frac{1}{d(j)}$$

4.6. lemma. . *Legyen az I' a* LubyMFH *algoritmus egyik szakaszában szereplő halmaz. Ekkor a szakasz előtt közvetlenül a gráf minden i indexű csúcsára fennáll, hogy*

$$Pr[i \in szom(I')] \geq \frac{1}{4} \min\left(\frac{sum(i)}{2}, 1\right).$$

A gráfból eltávolított élek számának várható értékére a 4.6. lemmát felhasználva kapunk korlátot.

4.7. lemma. . *A* LubyMFH *algoritmus egyetlen szakaszában a G gráfból eltávolított élek számának várható értéke legalább $\frac{|E|}{8}$.*

Bizonyítás. Az algoritmus biztosan elhagy minden olyan élt, amelynek legalább egyik végpontja $szom(I')$-beli. Ebből az következik, hogy az elhagyott élek számának várható értéke legalább

$$\frac{1}{2} \sum_{i \in V} d(i) \cdot Pr[i \in szom(I')].$$

Ez azért van így, mert minden i indexű csúcs bizonyos valószínűséggel rendelkezik egy I'-beli szomszéddal; ha ez az eset fennáll, akkor a i indexű csúcsot eltávolítjuk, ami maga után vonja az összes, $d(i)$ darab hozzá tartozó él törlését is. Az $\frac{1}{2}$ szorzó azt ellensúlyozza, hogy a törölt éleket kétszer számoljuk, hiszen minden élnek két olyan végpontja van, amely a törlését előidézheti.

Most a 4.6. lemmából származó korlátot használjuk fel, és azt kapjuk, hogy az elhagyott élek számának várható értéke legalább

$$\frac{1}{8} \sum_{i \in V} d(i) \cdot \min\left(\frac{sum(i)}{2}, 1\right).$$

Két részre bontva ezt az összeget aszerint, hogy a $sum(i)$ értéke mely i-kre lesz kisebb, mint 2 (és ennek megfelelően a minimalizálandó kifejezés milyen értéket vesz fel), azt kapjuk, hogy

$$\frac{1}{8}\left(\frac{1}{2} \sum_{i:sum(i)<2} d(i) \cdot sum(i) + \sum_{i:sum(i)\geq 2} d(i)\right).$$

Most ide beírjuk a $sum(i)$ definícióját, a $d(i)$-t pedig egy magától értetődő összeg alakjában fejezzük ki.

$$\frac{1}{8}\left(\frac{1}{2} \sum_{i:sum(i)<2} \sum_{j \in szom_i} \frac{d(i)}{d(j)} + \sum_{i:sum(i)<2} \sum_{j \in szom_i} 1\right).$$

Vegyük észre, hogy minden (i, j) irányítatlan él a zárójelen belüli kifejezés két (irányonként egy-egy) tagjában jelenik meg, és ezeknek a tagoknak az összege minden esetben nagyobb, mint 1. Így a teljes kifejezés legalább $\frac{|E|}{8}$.

□

A 4.7. lemmából következik az alábbi állítás:

4.8. lemma. . *A* LubyMFH *algoritmus egyetlen szakaszában a G-ből eltávolított élek száma legalább $\frac{1}{16}$ valószínűséggel legalább $\frac{|E|}{16}$.*

A 4.7. és 4.8. lemmákat flhasználva az alábbi tételhez jutunk:

4.9. tétel. . *A* LubyMFH *algoritmus egy valószínűséggel befejeződik. Továbbá, a befejeződésig végrehajtott menetek száma $O(\log\ n)$.*

Véletlenített algoritmusok.. Véletlenített algoritmusokkal gyakran találkozhatunk az osztott algoritmusok között. Ezek fő szerepe a szimmetria megtörésében van. Például a vezetőválasztás és az MFH feladatokat nem lehet determinisztikus folyamatok általános gráfjában egyedi azonosítók nélkül megoldani, mivel a szimmetriát lehetetlen megtörni, véletlenített algoritmussal viszont megoldhatók. Ráadásul, ha a folyamatok rendelkeznek is egyedi azonosítókkal, a véletlenített algoritmus gyorsabban szünteti meg a szimmetriát.

A véletlenített algoritmusoknak azonban van egy hibájuk, nevezetesen az, hogy működésük helyességét és/vagy befejeződését csak nagy valószínűséggel tudják garantálni, de nem bizonyosan. Az ilyen algoritmusok tervezésében fontos meggyőződni arról, hogy az algoritmus kritikus tulajdonságait feltétlenül kell-e garantálnunk, vagy csak adott valószínűség mellett. Például a LubyMFH algoritmus bármelyik végrehajtása garantáltan egy független halmazt állít elő, tekintet nélkül a véletlen választások kimenetelére. A befejeződés azonban a szerencsés véletlen választásoktól függ. Lehetséges az is (nulla valószínűségű), hogy mindegyik folyamat újra és újra ugyanazt az értéket választja, így az algoritmus megakad. Az, hogy ezek a jelenségek komoly akadályát képezik-e az algoritmus alkalmazásának, a megoldandó problémától függ.

4.6.. Megjegyzések a fejezethez

A MaxTerjed algoritmust és az OptMaxTerjed algoritmust széles körben ismerik. Teljes szinkron hálózatokban Afek és Gafni [6] találta meg a vezetőfolyamat-választás bonyolultsági korlátjait. A SzinkSZK algoritmus, melyet megtalálhatunk például a [83]-ban, a jól ismert szekvenciális szélességi keresésre épül. A Bellman-Ford algoritmus egy osztott változata a Bellman és Ford által egymástól függetlenül felfedezett szekvenciális algoritmusnak [43, 125].

A SzinkGHS algoritmus egy szinkronizált (és ezért lényegesen egyszerűbb) változata a jól ismert aszinkron MFF algoritmusnak, amelyet Gallager, Humblet, és Spira tervezett. A LubyMFH algoritmus és annak elemzése Luby cikkében [200] található.

A [271]-ben példát találunk egy véletlenített algoritmus (nulla valószínűség mellett bekövetkező) olyan végrehajtására, amelynél a folyamatok rendre ugyanazt az értékválasztást végzik.

4.7.. Gyakorlatok

4-1. Dolgozzuk ki a MaxTerjed algoritmus helyességbizonyításának részleteit.

4-2. Könnyű belátni, hogy a MaxTerjed algoritmusban használt üzenetek *átm*$|E|$ száma $O(n^3)$. Állítsunk elő irányított gráfoknak egy osztályát, amelyben az *átm*$|E|$ szorzat $\Omega(n^3)$, vagy mutassuk meg, hogy nincs az irányított gráfoknak ilyen osztálya.

4-3. Az OptMaxTerjed algoritmus által elküldött üzenetek számára adjunk a $O(n^3)$-nál kisebb felső korlátot vagy mutassunk egy olyan irányítottgráf-osztályt és megfelelő egyedi azonosítókat, amelyben az üzenetek száma $\Omega(n^3)$.

4-4. Vegyük az OptMaxTerjed algoritmusnak a 4.1.3. szakasz végén azt a „tovább optimalizált" változatát, amely megakadályozza a folyamatokat abban, hogy újra elküldjék a *max_azon* információt azokhoz a folyamatokhoz, amelyektől előzőleg megkapták.

(a) Adjuk meg ennek az algoritmusnak a kódját ugyanabban a stílusban, amellyel ebben a fejezetben a többi kódot is megadtuk.
(b) Bizonyítsuk be ennek az algoritmusnak a helyességét az OptMaxTerjed algoritmus mintájára, felhasználva ugyanazt a fajta egyszerűsítő stratégiát, amit az OptMaxTerjed algoritmus helyességbizonyításánál (azaz a 4.2. tételben) alkalmaztunk.

4-5.
(a) Írjuk meg a SzinkSZK algoritmus kódját.
(b) Bizonyítsuk be invariáns állítások használatával ennek az algoritmusnak a helyességét.
(c) Ismételjük meg az (a) és (b) részeket a gyerek mutatókat használó SzinkSZK algoritmusra.
(d) Ismételjük meg az (a) és (b) részeket a gyerek mutatókat és befejezés-jelzéseket használó SzinkSZK algoritmusra.

4-6. Vegyük a SzinkSZK algoritmusnak azt a 4.2.2. szakaszban vázolt optimalizált változatát, amely megakadályozza a folyamatokat abban, hogy elküldjék a `keres` üzenetet azoknak a folyamatoknak, amelyektől előzőleg már kaptak ilyen üzenetet.
(a) Adjuk meg ennek az algoritmusnak a kódját.
(b) Bizonyítsuk be ennek az algoritmusnak a helyességét a SzinkSZK algoritmus mintájára, felhasználva ugyanazt a fajta egyszerűsítő stratégiát, amit az OptMaxTerjed algoritmus helyességbizonyításánál (azaz a 4.2. tételben) alkalmaztunk.

4-7. Részletesen írjunk le egy algoritmust a SzinkSZK algoritmus kiegészítéseként, amely nemcsak a gyerek mutatókat állítja elő, hanem tájékoztat a gyerekekből a szülőkbe vezető legrövidebb útról is. Ezt az információt meg kellene osztani ezen utak mentén, úgy, hogy minden folyamat ismerje az út mentén elhelyezkedő következő folyamatot. Elemezzük a futási időt és a kommunikációs bonyolultságot.

4-8. Részletesen írjunk le egy algoritmust a SzinkSZK algoritmus kiegészítéseként, amely megengedi, hogy a forráscsúcs egy üzenetet küldjön minden más csúcsnak, és értesüljön arról, ha minden csúcs megkapta az üzenetet. Ez az algoritmus $O(|E|)$ üzenetet küldhet, és $O(\mathit{átm})$ futási időt használhat. Feltételezhetjük, hogy a hálózat irányítatlan.

4-9. Elemezzük a globális számítási séma, a vezetőválasztás és a 4.2. alfejezet végén szereplő átmérő számítási séma futási idejét és kommunikációs bonyolultságát, feltéve, hogy néhány szomszédos csúcs között kétirányú kommunikációt is megengedünk.

4-10. Tervezzünk egy minél hatékonyabb vezetőválasztó algoritmust egy olyan erősen összefüggő irányított hálózatban, amelyben a folyamatoknak van egyedi azonosítójuk.
(a) Tegyük fel, hogy a kommunikáció minden szomszédos csúcs közt kétirányú, azaz a hálózat irányítatlan.
(b) Ne alkalmazzuk az előző feltevést.

4-11. Adjunk egy minél hatékonyabb algoritmust a csúcsok teljes számának megállapítására egy olyan erősen összefüggő irányított hálózatban, amelyben a folyamatoknak van egyedi azonosítójuk.
(a) Tegyük fel, hogy a kommunikáció minden szomszédos csúcs közt kétirányú, azaz a hálózat irányítatlan.
(b) Ne alkalmazzuk az előző feltevést.

4-12. Adjunk minél hatékonyabb algoritmust az élek számának megállapítására egy olyan erősen összefüggő irányított hálózatban, amelyben a folyamatoknak van egyedi azonosítójuk.

(a) Tegyük fel, hogy a kommunikáció minden szomszédos csúcs közt kétirányú, azaz a hálózat irányítatlan.

(b) Ne alkalmazzuk az előző feltevést.

4-13. Adjunk minél hatékonyabb algoritmust minimális magasságú gyökeres feszítőfa előállítására. Feltételezhetjük, hogy a folyamatoknak van egyedi azonosítójuk, de nincs közöttük kitüntetett csúcs.

4-14.

(a) Adjuk meg a BellmanFord legrövidebb utak algoritmus kódját.

(b) Bizonyítsuk be invariáns állítások használatával ennek helyességét.

4-15. Adjuk meg a SzinkGHS algoritmus kódját.

4-16. Bizonyítsuk be a 4.5. lemmát.

4-17. Mutassuk meg, hogy a SzinkGHS algoritmusban $O(diam)$ menet nem mindig elég ahhoz, hogy a számítás összes szintjét befejezzük.

4-18. Mutassuk meg, hogy a SzinkGHS algoritmus azon változata (4.4. alfejezet vége), amely élazonosítókat használ az élsúlyok helyett, tényleg egy MFF-t állít elő.

4-19. *Kutatási kérdés.* Fejlesszünk ki a SzinkGHS algoritmusnál – a futási idő vagy a kommunikációs bonyolultság, vagy mindkettő szempontjából – jobb szinkronizált minimális feszítőfa-kereső algoritmust.

4-20. Adjuk meg a kódját annak a 4.4. alfejezet végén vázolt üzenetgyűjtő algoritmusnak, amely egy irányítatlan gráf tetszőleges feszítőfájában vezető folyamatot választ.

4-21. Adjunk minél jobb alsó és felső korlátot egy tetszőleges feszítőfa előállításának feladatára irányítatlan gráfban. Feltételezhetjük, hogy a csúcsok egyedi azonosítóval rendelkeznek és súlyokat nem használunk. Körültekintően állapítsuk meg, hogy a folyamatok milyen ismeretekkel rendelkezzenek a gráfról.

4-22. Vegyünk egy vonalhálózatot, azaz az $1, \ldots, n$ sorszámú folyamatoknak egy olyan lineáris alakzatát, ahol a folyamatok kétirányú kapcsolatban állnak szomszédjaikkal. Tegyük fel, hogy minden egyes P_i folyamat meg tudja különböztetni a bal oldalát a jobb oldalától, és ismeri azt is, hogy ő maga végpont-e vagy sem. Tegyük fel, hogy minden folyamat kezdetben egy nagyon nagy v_i egész értékkel rendelkezik, és azt, hogy az ilyen értékekből egy adott időpillanatban csak adott számút tarthatunk nyilván a memóriában. Tervezzük meg azt az ezen értékeket sorba rendező algoritmust, amelyben az egyes P_i folyamatok által előállított o_i kimeneti értékek összeszorzott halmaza megegyezik a v_i bemeneti értékek összeszorzott halmazával, és $o_1 \leq \ldots \leq o_n$. Próbáljuk meg előállítani mind az üzenetek, mind a menetek száma tekintetében a leghatékonyabb algoritmust. Állításainkat indokoljuk.

4-23. Bizonyítsuk be a 4.5. alfejezet feltételeit alapul véve, de véletlenszerűek helyett determinisztikus folyamatokra, hogy van olyan gráf, amelyben lehetetlen az MFH problémát megoldani. Keressük meg a lehető legnagyobb olyan osztályát a gráfoknak, amelyre ez fennáll.

4-24. Tételezzük fel, hogy a LubyMFH algoritmust egy n méretű gyűrűben hajtjuk végre. Becsüljük meg, mekkora valószínűséggel távolít el az algoritmus egy tetszőleges élt egy ismétlésben (iterációban).

Tárgymutató