6.

Gráfalgoritmusok

Kezdetben van a viszony.
Tekintsük a primitív népek nyelvét... E nyelv sejtmagjai – e nyelvtaniság előtti ősformák, melyek szétrobbanásából jön létre a szófajok sokfélesége, ezek általában valamely viszony egészét jelölik. Ahol mi azt mondjuk: „messze, távol", ott a zulu mondatszót mond, mely ennyit tesz: „Ott, ahol az ember ezt kiáltja: »Anyám, elvesztem!«."
MARTIN BUBER: Én és Te

6.1. Bevezetés

Algoritmikus problémák megoldása során gyakran van szükségünk dolgok közötti bináris kapcsolatok ábrázolására, kezelésére. Erre természetes modellt kínálnak a gráfok. A gráf csúcsai az objektumoknak, az élei pedig a köztük levő (pár-)kapcsolatoknak felelnek meg. A probléma jellegéből adódóan használhatunk irányított vagy irányítatlan, súlyozott vagy súlyozatlan élű gráfokat. A gráf népszerű modellező eszköz egyszerűsége, kezelhetősége és kifejező képességének tágassága miatt. A seregnyi alkalmazási terület között előkelő helyen említhető maga a számítástudomány.

Az egyik legjelentősebb adatmodellező irányzat a hálós megközelítés[1], ami a leírni kívánt jelenségkört irányított gráfokkal ábrázolja. A modell alapfogalmai a gráf csúcsai, az élek pedig a köztük meglevő viszonyokat, kölcsönhatásokat, kapcsolatokat jelölik. Gráf az egész világ – sugallják a hálós filozófia hívei. Gráfoknak tekinthetők a belső memóriás adatkezelés (olykor igen szövevényes) listaszerkezetei is, ahol az adatelemek közötti mutatók (elterjedt szakkifejezéssel: pointe-

[1] A régebbi adatbázis-rendszerek közül ezt a modellt használja pl. az IDMS, az újabbak közül pedig a dbVISTA.

rek) jelentik az éleket. Ez a terület a gráfalgoritmusok időrendben első komoly alkalmazási terepe. Ebben a környezetben gyakran van szükség olyan algoritmusokra, amelyek elérhetőséggel, összefüggőséggel kapcsolatos kérdéseket válaszolnak meg, vagy hatékonyan bejárják a szerkezetet. Utóbbi feladat egy változatával, nevezetesen a bináris fák bejárásaival (pre-, in-, postorder) már találkoztunk.

Hasznosságukon túl az igazán csiszolt elemi gráfalgoritmusok sajátosan érdekesek is. Ennek a titka talán abban van, hogy itt három ismeretkör szálai fonódnak egybe: gráfokkal kapcsolatos matematikai tények (pl. a feszítőfák tulajdonságai), nagy erejű adatszerkezetek (pl. a kupac vagy az itt bemutatásra kerülő UNIÓ-HOLVAN) és általános algoritmikus elvek (pl. a mohó módszer vagy az optimalitás elve). A fejezetben tárgyalt módszerek tehát a közvetlen hasznukon túl egyszersmind az adatszerkezetekkel kapcsolatos ismeretek alkalmazásaként is szolgálnak, és előkészítő anyagot adnak a későbbi, általános algoritmuselméleti részekhez.

A fejezetben a meglehetősen szerteágazó anyagból csak néhány fontos és egyszerű, az alkalmazásokban gyakran felmerülő feladat és gráfalgoritmus ismertetésére szorítkozhatunk. Ezek az alapvető módszerek számos más, összetettebb algoritmus részét képezik, vagy éppen vázául szolgálnak. Némelyek közülük olyan gondolatot rejtenek magukban, amelyek kiindulópontjai voltak/lehetnek más problémák megoldásának.

Itt a bevezetőben tisztázzuk azokat a fogalmakat, jelöléseket, amelyeket a későbbiekben használni fogunk, és bemutatjuk a gráfok legtöbbször használt számítógépes ábrázolási módjait. Utána a legrövidebb utak keresésének feladatát tanulmányozzuk. Másik központi témánkat a hatékony bejáró algoritmusok (mélységi és szélességi bejárás) jelentik. Ezt követően a legismertebb feszítőfa-algoritmusokat tekintjük át. A fejezet végén a kombinatorikus optimalizálás irányába mutató feladatokat veszünk szemügyre: párosítások keresésével és hálózati folyamokkal foglalkozunk.

6.1.1. Alapfogalmak, jelölések

Egy G gráf két halmazból áll: a *csúcsok* vagy *pontok* V halmazából, mely egy véges, nem üres halmaz; és az *élek* E halmazából, melynek elemei bizonyos V-beli párok. Ha ezekről a pontpárokról kikötjük, hogy rendezettek legyenek, akkor *irányított gráfról*, különben pedig irányítatlan gráfról, vagy egyszerűen csak *gráfról* beszélünk. Az irányított gráfokat tehát nem szimmetrikus, míg az irányítatlanokat szimmetrikus kapcsolatok leírására használhatjuk. Már itt megjegyezzük, hogy egy irányítatlan gráf felfogható speciális irányított gráfként, ha minden élét két – oda és vissza mutató – irányított éllel helyettesítjük. Ezt az átírást használva

több irányított gráfokra megfogalmazott feladat (és azt megoldó algoritmus) értelmezhető (használható) irányítatlan gráfokra is. Fordítva, az irányított gráfok is tekinthetők irányítatlannak úgy, hogy elfeledkezünk az élek irányításról. Ekkor nem teszünk különbséget az (u, v) és a (v, u) pár (él) között.

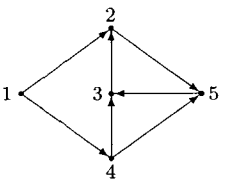
A H halmaz elemszámát $|H|$ jelöli. Az egyszerűség kedvéért az n és e betűk általában az adott gráf csúcs-, illetve élhalmazának elemszámát jelölik. Nevezetesen ha a $G = (V, E)$ gráfról beszélünk, akkor $n = |V|$ és $e = |E|$.

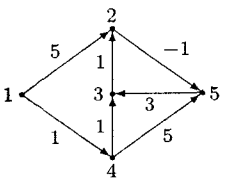
Az objektumok közötti kapcsolat sokszor jelenti út létezését vagy kommunikáció lehetőségét. Ilyenkor gyakran van szükség arra, hogy az élekhez *súlyokat* vagy *költségeket* tartsunk nyilván, melyek például időt, hosszúságot, pénzt és még sok mást ábrázolhatnak. Ezt általában egy valós értékű függvényként fogjuk fel, melynek értelmezési tartománya a gráf élhalmaza, és melyet c-vel, a *cost* angol szó kezdőbetűjével jelölünk.

Egy út – akár irányított, akár irányítatlan gráfban – csúcsok olyan $v_1, \ldots, v_k$ sorozata (ez lehet egyelemű is), melyre (v_i, v_{i+1}) minden i-re $(1 \leq i \leq k-1)$ éle a gráfnak. Egy utat *körnek* nevezünk, ha kezdő és végpontja megegyezik. Egy út, illetve egy kör *egyszerű*, ha minden csúcs, amin áthalad, különböző, kivéve a kör kezdő- és végpontját. Irányított gráfoknál a (v, w) él jelölésére használni fogjuk a $v \to w$ változatot is. Egy a v csúcsból a w-be menő irányított útra általában a $v \rightsquigarrow w$ jelöléssel utalunk.

Legyen $G = (V, E)$ egy – akár irányított, akár irányítatlan – gráf. Értelmezzük a $\sim$ relációt a V ponthalmazon a következőképpen: az $x, y \in V$ csúcsokra $x \sim y$, ha x és y között megy út (irányított gráfok esetén az irányítást nem figyelembe véve, azaz a $v \to w$ élen mehetünk w-ből v-be is). Könnyen belátható, hogy $\sim$ egy ekvivalenciareláció. A $\sim$ ekvivalenciaosztályait hívjuk a gráf *komponenseinek*. Egy gráf *összefüggő*, ha egyetlen komponensből áll. Egy gráfot *erdőnek* nevezünk, ha irányítatlan gráfként szemlélve nem tartalmaz kettőnél több pontból álló egyszerű kört (körmentesség). Ha egy erdőnek csak egyetlen komponense van, azaz összefüggő, akkor *fa*. Egy (irányított) gráf egy pontjának a *foka* a belőle kiinduló élek száma.

Példa: A következő ábrán egy irányított gráf és annak egy súlyozott élű változata látható. A gráf csúcshalmaza $V = \{1, 2, 3, 4, 5\}$, az élhalmaza pedig $E = \{(1,2), (1,4), (2,5), (3,2), (4,3), (4,5), (5,3)\}$. Tehát $n = 5$, és $e = 7$. A 4 csúcs foka 2.





A súlyozott változatnál az élekre írtuk azok költségeit. Így például a $2 \to 5$ él költsége $c(2,5) = -1$. Az 14325 pontsorozat egy $1 \rightsquigarrow 5$ irányított út; ez egyszerű út, mert a sorozatban nincs ismétlődés. Az 5325 sorozat pedig egy egyszerű irányított kör a gráfban.

A gráf összefüggő, hiszen ha az élek mentén a fordított irányban is léphetünk, akkor bárhonnan eljuthatunk bárhová. (Ha csak az élek irányát követve haladhatunk, akkor 1-be nem jutunk el egyetlen más csúcsból sem.)

Ha a gráfból töröljük a 2-nél nagyobb súlyú éleket, akkor a megmaradó G' gráf egy fa lesz. A G' csúcshalmaza V, az élei pedig $(1,4)$, $(4,3)$, $(3,2)$ és $(2,5)$. A G' gráf összefüggő és körmentes.

6.1.2. Gráfok ábrázolásai

A gráfok kényelmesen leírhatók számítógépes adatszerkezetekkel. Itt két széles körben használatos megadási módjukat ismertetjük, az adjacencia-mátrixot és az éllistát.

Adjacencia-mátrix

A $G = (V, E)$ gráf *adjacencia-mátrixa* (vagy *szomszédossági mátrixa*) a következő – a V elemeivel indexelt – n-szer n-es mátrix:

$$A[i,j] = \begin{cases} 0 & \text{ha } (i,j) \notin E, \\ 1 & \text{ha } (i,j) \in E. \end{cases}$$

Irányítatlan gráfok esetén a szomszédossági mátrix szimmetrikus lesz (azaz $A[i,j] = A[j,i]$ teljesül minden i, j csúcspárra). Ha az élekhez még költségeket is nyilván kell tartanunk, akkor az alábbi, árnyaltabb szerkezetű szomszédossági mátrixot fogjuk használni:

$$C[i,j] = \begin{cases} 0 & \text{ha } i = j, \\ c(i,j) & \text{ha } i \neq j \text{ és } (i,j) \text{ éle } G\text{-nek}, \\ * & \text{különben.} \end{cases}$$

Itt a $*$ szerepe csak annyi, hogy jelzi egy él nemlétét. Bizonyos esetekben más, az alkalmazás természetéhez jobban illeszkedő megkülönböztető jelet érdemes használni. Például később, a távolságok számításakor $*$ helyett a ∞ jelet használjuk, és „végtelenül nagy" súlyként értelmezzük. A főátlóba nullákat írtunk; ezt azért tettük, mert a súlyozott élű gráfokra vonatkozó feladatokban, amikkel foglalkozni fogunk, az $u \to u$ alakú ún. hurokélek érdektelenek.

Az adjacencia-mátrix természetes módon tekinthető kétdimenziós tömbnek. Ezzel kényelmes lehetőséget kínál gráfok számítógépes leírására.

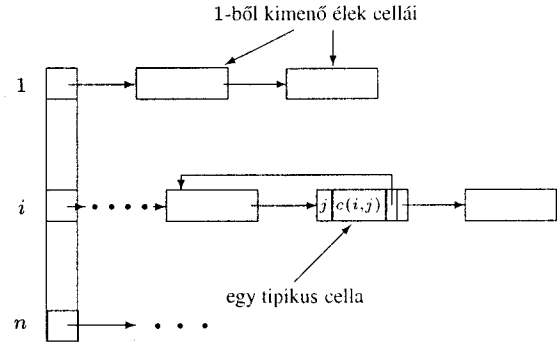
Az előző példa kétféle (egyszerű, illetve költségeket tartalmazó) adjacencia-mátrixa így fest:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & 5 & * & 1 & * \\ * & 0 & * & * & -1 \\ * & 1 & 0 & * & * \\ * & * & 1 & 0 & 5 \\ * & * & 3 & * & 0 \end{pmatrix}.$$

Az adjacencia-mátrixszal való leírás egyszerű, programozástechnikai szempontból tiszta megoldás. Hátránya viszont, hogy a mérete (n^2 tömbelem) teljesen független az élek számától. Ha G-nek viszonylag kevés éle van, akkor gyakran hasznosabb az éllistás megadás.

Éllistás megadás

Ennél az ábrázolási módnál a $G = (V, E)$ gráf minden csúcsához egy lista tartozik. Az $i \in V$ csúcs listájában tároljuk az i-ből kimenő éleket, és ha kell, ezek súlyát is. Az i listáján egy élnek a lista egy eleme (cellája) felel meg. A lista elemeit a szokásos módon, mutatókkal láncoljuk egymás után (esetleg kétszeresen, visszafelé menő mutatókat is alkalmazva). Alább egy ilyen szervezés vázlata látható:



A (függőlegesen rajzolt) n méretű $L[1:n]$ tömb $L[i]$ eleme az i csúcs éllistájának a feje. Az $L[i]$ tehát egy mutató az i-ből induló élek listájának első cellájára. Az (i, j) élnek megfelelő cella tartalmazza a j sorszámot, a $c(i, j)$ súlyt (ha van), egy mutatót a következő cellára, és esetleg még egyet az előzőre is.

Egy irányított gráf éllistás megadásában az n listán összesen e cella szerepel. Így az egész szerkezet elfér $n + e$ cellányi helyen. Irányítatlan gráfoknál a helyigény $n + 2e$ cella, hiszen az (i, j) él az i és a j csúcs listáján is megtalálható.

Ritka gráfok esetén, amikor e jóval kisebb, mint n^2, az éllistás megadás tömörebb az előzőnél. Az éllistás ábrázolás előnye még, hogy gyorsan végignézhetjük az egy csúcsból kiinduló összes élet. Az adjacencia-mátrixos megadás mellett viszont gyorsabban el tudjuk dönteni, hogy egy adott (i, j) pár éle-e G-nek.

6.2. A legrövidebb utak problémája (egy forrásból)

Legyen adott egy $G = (V, E)$ irányított gráf a $c(f)$, $f \in E$ élsúlyokkal. A csúcsok például jelölhetik egy többé-kevésbé hóval borított hegység különböző pontjait, az élek azt, hogy el lehet-e síelni egy sípályán az egyik pontból a másikba; az élsúlyok pedig azt, hogy mennyi idő kell ehhez. Felmerülhet a kérdés, hogy mekkora (itt időben mérve) a legrövidebb út *egy adott pontból egy másik adott pontba* vagy *egy adott pontból az összes többibe* vagy *bármely két pont között*. A meglepő az, hogy az első két probléma „ugyanolyan nehéz": az ismert algoritmusok, amelyek az elsőt megoldják, egyben a másodikat is megoldják. Tehát rögtön a második kérdéssel fogunk foglalkozni, a harmadik problémát pedig a következő részben tárgyaljuk.

Először fogalmazzuk meg pontosan a feladatot. A G gráf egy u-t v-vel összekötő (nem feltétlenül egyszerű) $u \rightsquigarrow v$ irányított útjának a *hossza* az úton szereplő élek súlyainak összege. *Legrövidebb* $u \rightsquigarrow v$ úton egy olyan $u \rightsquigarrow v$ utat értünk, amelynek a hossza minimális a G-beli $u \rightsquigarrow v$ utak között. Ezek után értelmezhetjük az u és v csúcsok (G-beli) $d(u, v)$ *távolságát*: ez 0, ha $u = v$; ∞, ha nincs $u \rightsquigarrow v$ út; egyébként pedig a legrövidebb $u \rightsquigarrow v$ út hossza. (Vigyázat, itt u és v nem felcserélhető: ha az egyik csúcs valamilyen távol van a másiktól, akkor nem biztos, hogy a másik is ugyanolyan távol van az egyiktől!) A meghatározás nem mindig értelmes. Ha például u és v egy olyan irányított körön vannak, amelynek az összsúlya negatív, akkor ezen körözve akármilyen kicsi (azaz nagy abszolút értékű negatív) úthosszat elérhetünk. Természetes kikötés tehát, hogy G ne tartalmazzon negatív összsúlyú irányított kört.

Jelöljünk ki a G gráfban egy $s \in V$ csúcsot forrásnak. Első célunk, hogy az $u \in V$ pontoknak az s-től való távolságát meghatározzuk. Tegyük fel továbbá, hogy a $c(f)$ élsúlyok nemnegatívak. Ekkor nyilván nincs az előbbi értelemben vett negatív kör.

A legrövidebb utak problémája (egy forrásból):

Adott egy $G = (V, E)$ irányított gráf, a $c : E \to \mathbb{R}^+$ nemnegatív értékű súlyfüggvény, és egy $s \in V$ csúcs (a forrás). Határozzuk meg minden $v \in V$-re a $d(s, v)$ távolságot.

6.2.1. Dijkstra módszere

A következőkben ismertetjük E. W. Dijkstra hatékony, ugyanakkor bűbájosan egyszerű algoritmusát az előbb megfogalmazott feladat megoldására. Egy a G csúcsaival indexelt $D[\,]$ tömböt használunk. A $D[v]$ értékre úgy gondolhatunk, hogy az minden időpillanatban az eljárás során addig megismert legrövidebb $s \rightsquigarrow v$ utak hosszát tartalmazza. A $D[v]$ mennyiség mindenkor felső közelítése lesz a keresett $d(s, v)$ távolságnak. A közelítést lépésről lépésre finomítva végül a kívánt értékeket kapjuk. Az algoritmus pontosabb megfogalmazásához először tegyük fel, hogy a G gráf az alábbi alakú C adjacencia-mátrixával adott:

$$C[v, w] = \begin{cases} 0 & \text{ha } v = w, \\ c(v, w) & \text{ha } v \neq w \text{ és } (v, w) \text{ éle } G\text{-nek}, \\ \infty & \text{különben.} \end{cases}$$

Kezdetben $D[v] := C[s, v]$ minden $v \in V$ csúcsra. Válasszuk ki ezután az s csúcs szomszédai közül a hozzá legközelebbit, vagyis egy olyan $x \in V \setminus \{s\}$ csúcsot, melyre $D[x]$ $(= C[s, x])$ minimális. Ekkor biztos, hogy az egyetlen (s, x) élből álló út egy legrövidebb $s \rightsquigarrow x$ út, hiszen bármerre másfele indulnánk el s-ből, már az első él miatt legalább ilyen hosszú utat kapnánk (az élsúlyok nemnegatívak!). Tehát x-et betehetjük (s mellé) a KÉSZ halmazba. A KÉSZ halmaz azokat a csúcsokat tartalmazza, amelyeknek s-től való távolságát már tudjuk. Ezek után módosítsuk a többi csúcs $D[w]$ értékét, ha az eddig ismertnél rövidebb úton el lehet érni oda x-en keresztül, azaz ha $D[x] + c(x, w) < D[w]$. Most újra válasszunk ki a $v \in V \setminus$ KÉSZ csúcsok közül egy olyat, amelyre $D[v]$ minimális. Ezen csúcs $D[\,]$-értéke már az s-től való távolságát tartalmazza az előzőhöz hasonló okok miatt (ezt később bebizonyítjuk). Majd megint a $D[\,]$-értékeket módosítjuk, és így tovább, míg minden csúcs be nem kerül a KÉSZ halmazba.

Az eljárás egy *mohó algoritmus*: a KÉSZ halmaz bővítésekor látszólag nem a teljes kép figyelembe vételével választ optimumot. Mégis ezek a helyinek, korlátozott érvényűnek tűnő döntések együtt elvezetnek a tényleges optimumokhoz. A módszert ezek után formálisan a következő programvázlat írja le:

Dijkstra algoritmusa adjacencia-mátrixszal

(1) KÉSZ := $\{s\}$
for minden $v \in V$ csúcsra **do**
$D[v] := C[s, v]$ (* a $d(s, v)$ távolság első közelítése *)
(2) **for** $i := 1$ **to** $n - 1$ **do begin**
Válasszunk olyan $x \in V \setminus$ KÉSZ csúcsot, melyre $D[x]$ minimális.
Tegyük x-et a KÉSZ-be.
for minden $w \in V \setminus$ KÉSZ csúcsra **do**
(3) $D[w] := \min\{D[w], D[x] + C[x, w]\}$ (* $d(s, w)$ új közelítése *)
end

A módszer helyességének igazolásához hasznos lesz a *különleges út* fogalma: egy $s \rightsquigarrow z$ irányított út különleges, ha a z végpontot kivéve minden pontja a KÉSZ halmazban van. A különleges úttal elérhető pontok éppen a KÉSZ-ből egyetlen éllel elérhető pontok.

Állítás: *A (2) ciklus minden iterációs lépése után érvényesek a következők:*

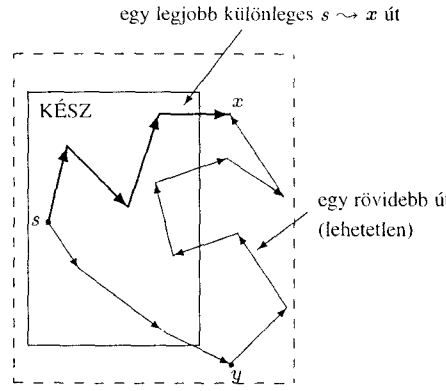
(a) KÉSZ pontjaira $D[v]$ a legrövidebb $s \rightsquigarrow v$ utak hossza.

(b) Ha $v \in$ KÉSZ, akkor van olyan $d(s, v)$ hosszúságú (más szóval legrövidebb) $s \rightsquigarrow v$ út is, amelynek minden pontja a KÉSZ halmazban van.

(c) Külső (vagyis $w \in V \setminus$ KÉSZ) pontokra $D[w]$ a legrövidebb különleges $s \rightsquigarrow w$ utak hossza.

Bizonyítás: Teljes indukciót alkalmazunk. A kezdőértékek beállítása után, amikor a (2) ciklus még egyszer sem futott le, mindhárom állítás nyilvánvalóan igaz. Tegyük fel, hogy igazak a j-edik iteráció után, azaz miután a (2) ciklus magja j-szer lefutott. Azt szeretnénk belátni, hogy igazak maradnak a $j + 1$-edik iteráció után is. Tegyük fel, hogy az algoritmus a $j + 1$. iterációs lépésben az x csúcsot választja a KÉSZ-be.

(a) Gondolkozzunk indirekte: mi van, ha $D[x]$ nem a $d(s, x)$ távolságot jelöli, azaz van ennél rövidebb $s \rightsquigarrow x$ út? Ez nem lehet különleges, elvégre a (c)-re vonatkozó indukciós feltevés szerint $D[x]$ az x-be vezető legrövidebb különleges utak hosszát tartalmazza. Ezen út „eleje" azonban különleges, mert a KÉSZ-ből indulva egy azon kívüli csúcsba jut el. Legyen y az út első olyan pontja, amely már nincs benne a KÉSZ halmazban. Ekkor az $s \rightsquigarrow y$ útdarab hossza is kisebb, mint $D[x]$. Az útdarab különleges, így a hossza nem kevesebb, mint $D[y]$. (Itt ismét használtuk a (c)-re vonatkozó indukciós feltevést.) Ezeket összevetve kiderül, hogy a j-edik iteráció után $D[y] < D[x]$, ami képtelenség, hiszen ekkor az algoritmusnak y-t kellett volna választania x helyett.



(b) Ezt is elég csupán az x csúcsra igazolni, hiszen a KÉSZ korábbi pontjaira az indukciós feltevés adja az állítást. Már beláttuk, hogy $d(s,x) = D[x]$. Az utóbbi érték egy különleges $s \rightsquigarrow x$ út hossza volt a $j+1$. iteráció előtt (itt a (c)-re vonatkozó indukciós feltevést használtuk); annak végeztével az út minden pontja KÉSZ-beli lesz.
(c) A (b) alapján $d(s,v) + C[v,w]$ a legrövidebb olyan $s \rightsquigarrow w$ különleges utak hossza, amelyek utolsó előtti pontja v. Emiatt az állítás egyenértékű a

$$D[w] = \min_{v \in \text{KÉSZ}} \{d(s,v) + C[v,w]\},$$

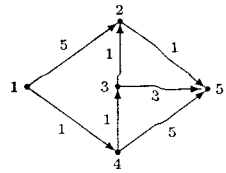
egyenlőséggel. Miért lesz ez igaz? A (c)-re vonatkozó indukciós feltevés alapján a $D[w]$ régi (a $j+1$. iteráció előtti) értéke

$$D[w] = \min_{v \in \text{KÉSZ} \setminus \{x\}} \{d(s,v) + C[v,w]\}.$$

Ezt – a már igazolt $d(s,x) = D[x]$ egyenlőséget is figyelembe véve – a (3) sorban a $d(s,x) + C[x,w]$ mennyiséggel vetjük össze. A $D[w]$-be tehát tényleg az új KÉSZ halmazra vett minimum kerül. □

Végül amikor KÉSZ már a teljes csúcshalmaz, $D[v] = d(s,v)$ teljesül minden $v \in V$ csúcsra az állítás (a) része szerint. Ezzel igazoltuk az algoritmus helyességét. Most vizsgáljuk meg az időigényét (n csúcs esetén). A kezdőértékek beállítása $O(n)$ elemi lépést vesz igénybe. A (2) ciklus belsejében a minimumkeresés és a (3) ciklus szintén $O(n)$-et, így mivel a (2) ciklus $n-1$-szer fut le, *az algoritmus összidőigénye* $O(n^2)$ *művelet.*

Példa: Nézzük a Dijkstra-algoritmus működését a következő rajzon látható G gráffal mint bemenettel; a forrás az $s := 1$ csúcs:



A D tömb helyzetét mutatjuk a (2) iteráció megkezdése előtt, majd pedig az első három iterációs menet után. Az utolsó, a negyedik menetben csak a KÉSZ halmaz változik, D nem. Vastag betűvel szedtük a KÉSZ-beli csúcsokhoz tartozó értékeket. Ezek a megfelelő legrövidebb utak hosszai. Érdemes megfigyelni a $D[5]$ alakulását. Látható, hogy mindhárom menetben csökkent, míg végül elérte a $d(1,5)$ értéket.

1	2	3	4	5
0	5	∞	1	∞

1	2	3	4	5
0	5	2	1	6

1	2	3	4	5
0	3	**2**	**1**	5

1	2	3	4	5
0	**3**	**2**	**1**	4

Dijkstra algoritmusa éllistával

Ha a G gráf ritka, vagyis kevés éle van, akkor az időigény csökkenthető, amennyiben a gráfot éllistával tároljuk. Az előző algoritmust a következőképpen módosítjuk: $V\backslash$KÉSZ csúcsait *kupacba* rendezve tartjuk a $D[\ ]$ érték szerint. A kezdeti kupacépítés $O(n)$ költséggel végezhető el. A (2) ciklus belsejében levő minimumkeresést egy $O(\log n)$ költségű MINTÖR művelettel oldjuk meg. A $D[\ ]$ értékek *újraszámolását és a kupac-tulajdonság helyreállítását (utóbbi csúcsonként egy* $O(\log n)$ költségű FOGYASZT) csak a választott csúcs szomszédaira kell elvégezni. Minden csúcsot pontosan egyszer választunk ki, és a szomszédok számának összege e. *Tehát itt az összidőigény* $O((n+e)\log n)$. Ez figyelemre méltó javítás a mátrixos megoldáshoz képest, ha a gráf ritka, azaz e sokkal kisebb, mint n^2.

Sűrű gráfokra még komolyabb gyorsítás érhető el alkalmas d-kupaccal. Ekkor az előző bekezdésben taglalt teendők költsége $O(n + nd\log_d n + e\log_d n)$ lesz. Itt az első tag a kupacépítés, a második a minimumok, a harmadik a fogyasztások lépésszáma. Tegyük fel, hogy G-nek sok éle van, mondjuk $n^{1,5} \leq e \leq n^2$ és legyen $d = \lceil e/n \rceil$. Ekkor $d \geq \sqrt{n}$, amiből $\log_d n \leq 2$. Ezt használva

$$O(n + nd\log_d n + e\log_d n) = O(n + nd + e) = O(n + n\cdot e/n + e) = O(e).$$

Az algoritmus *ebben az esetben a bemenet hosszával arányos lépésszám alatt* végez – más szóval: *lineáris idejű*. Ebből következően elmondhatjuk, hogy sűrű gráfokra ez az implementáció konstans szorzótól eltekintve optimális. A módszer eme

változata ékesszólóan példázza az adatszerkezetek alkalmazásaiban rejlő lehetőségeket. Az egyszerű kupacot (vagyis 2-kupacot) használó megoldáshoz képest azért nyertünk, mert sokkal több a fogyasztás, mint a minimumok törlése. Az előbbiek pedig olcsóbbak, ha nagy a kupac d elágazási tényezője.

Nyilvánvaló, hogy a Dijkstra-algoritmus irányítatlan gráfokra is működik, hiszen egy irányítatlan gráf felfogható olyan irányított gráfnak, melyben minden él mindkét irányba mutat. Megjegyzésképp annyit, hogy irányítatlan gráfok esetében az élsúlyokra vonatkozó kétféle kikötés ugyanazt jelenti (vagyis hogy ne legyen negatív összsúlyú kör, illetve hogy az élsúlyok nemnegatívak), hiszen egy negatív irányítatlan élből egy (kettő hosszúságú) negatív kört kapunk.

A legrövidebb utak nyomonkövetése

Sokszor persze nemcsak a legrövidebb utak hosszára, hanem magukra a legrövidebb utakra is kíváncsiak vagyunk. Mindkét algoritmust a futási idő nagyságrendjének megtartása mellett kibővíthetjük egy $P[\,]$ tömb karbantartásával, amely minden csúcshoz megadja egy az eddig ismert hozzá vezető legrövidebb úton az utolsó előtti csúcsot. Azért elég csupán az utolsó előtti csúcsot tárolni, mert ha egy v csúcsba vezető legrövidebb útból elhagyjuk az utolsó élet, akkor nyilván az utolsó előtti csúcsba vezető legrövidebb utat kapunk, aminek szintén ismerjük a cél előtti állomását, és így tovább. Visszafelé felgombolyíthatunk egy a v-be vivő legrövidebb utat.

Tehát kezdetben $P[v] := s$ minden $v \in V$-re. Ezután csak annyit kell hozzátenni az algoritmushoz, hogy a (3) ciklus belsejében, ha egy külső w csúcs $D[w]$ értékét megváltoztatjuk, akkor $P[w] := x$, hiszen ekkor találtunk egy rövidebb speciális utat w-be, melynek utolsó előtti állomása x. Könnyű meggondolni, hogy végül tényleg igaz lesz az, hogy $P[v]$ egy legrövidebb $s \rightsquigarrow v$ út utolsó előtti pontja; másként fogalmazva: $d(s,v) = d(s,P[v]) + C[P[v],v]$ és $P[v] \neq v$ érvényes lesz minden $v \in V \setminus \{s\}$-re.

Feladat: Mutassuk meg, hogy a $P[v] \to v$ élek egy fát alkotnak. A fa $P[v] \to v$ élének a súlya legyen $C[P[v],v]$. Ekkor a fában az s-től mért távolságok ugyanazok lesznek, mint a G-beli távolságok.

6.2.2. A Bellman—Ford-módszer

Olyan módszert ismertetünk az egy pontból induló legrövidebb utak (hosszának) meghatározására, amely akkor is működik, ha bizonyos élsúlyok negatívak. Csupán annyit teszünk fel, hogy *G nem tartalmaz negatív összhosszúságú irányított*

kört. Mint látni fogjuk, a nagyobb általánosságért idővel fizetünk. Az eljárás n^3-bel arányos számú műveletet igényel.

Feladat: Mutassuk meg, hogy ha a súlyozott élű G irányított gráfban nincs negatív összhosszúságú irányított kör, akkor bármely két pontja között van olyan legrövidebb út is, ami egyszerű (azaz nem tartalmaz ismétlődő csúcsot). Következésképpen van nem több, mint $n-1$ élből álló legrövidebb út is, ahol n a G csúcsainak száma.

Tegyük fel itt is, hogy a $G = (V, E)$ súlyozott élű irányított gráf a C adjacencia-mátrixával adott (ebben a diagonális elemek nullák, az i, j helyzetű elem a $c(i, j)$ élsúly, ha $i \to j$ éle G-nek, a többi elem pedig ∞). Az egyszerűség kedvéért tegyük még fel, hogy $V = \{1, 2, \ldots, n\}$ és $s = 1$. A szakirodalomban többnyire R. E. Bellmannak és L. R. Fordnak tulajdonított algoritmus fokozatos közelítéssel határozza meg az s-ből a többi csúcsba vivő legrövidebb utak hosszát.

A módszer tulajdonképpen egy $T[1 : n-1, 1 : n]$ táblázat (kétdimenziós tömb) sorról sorra haladó kitöltése. Azt szeretnénk elérni, hogy végül minden i, j-re ($1 \le i \le n-1, 1 \le j \le n$) teljesüljön, hogy

$$(*) \qquad T[i,j] = \begin{array}{l}\text{a legrövidebb olyan } 1 \rightsquigarrow j \text{ irányított utak hossza,}\\ \text{melyek legfeljebb } i \text{ élből állnak.}\end{array}$$

A feladatban foglalt állítás szerint ekkor $T[n-1, j]$ a legrövidebb $1 \rightsquigarrow j$ utak hosszát tartalmazza. A módszert úgy tekinthetjük, hogy a keresett minimális távolságokat $n-1$ menetben közelítjük. Az első menet, a $T[1, j]$ sor kitöltése kézenfekvő, hiszen nyilván $T[1, j] = C[1, j]$. Tegyük fel ezután, hogy az i-edik sort már kitöltöttük, azaz a $T[i, 1], T[i, 2], \ldots, T[i, n]$ értékekre $(*)$ igaz. Ekkor

$$(**) \qquad T[i+1, j] := \min\{T[i, j], \min_{k \neq j}\{T[i, k] + C[k, j]\}\}$$

adja az $i+1$. sor helyes értékeit. Ugyanis egy legfeljebb $i+1$ élből álló $\pi = 1 \rightsquigarrow j$ út kétféle lehet:

(a) Az útnak kevesebb, mint $i+1$ éle van. Ekkor ennek a hossza szerepel $T[i, j]$-ben.

(b) Az út éppen $i+1$ élből áll. Legyen l a π út utolsó előtti pontja. Ekkor a π út $1 \rightsquigarrow l$ szakasza i élből áll, és a π minimalitása miatt minimális hosszúságú a legfeljebb i élű $1 \rightsquigarrow l$ utak között. A hossza tehát a T már elkészült darabjára tett feltevésünk miatt $T[i, l]$. Eszerint a π hossza $T[i, l] + C[l, j]$.

A fejtegetés tanulsága, hogy a π hossza szerepel a $(**)$ jobboldalán azon mennyiségek között, amelyek minimumát vesszük; tehát egyenlő a minimummal.

A $(**)$ rekurzió fontos és hasznos vonása, hogy a $T[i+1, j]$ értéket adó minimális utak kezdődarabját, az $1 \rightsquigarrow l$ részt (illetve annak hosszát) nem kell hatalmas

szénakazalban keresnünk. Tudjuk ugyanis, hogy ez az útdarab is rendelkezik egy optimalitási tulajdonsággal: minimális hosszú a legfeljebb i élből álló $1 \rightsquigarrow l$ utak között. Optimalizálási feladatoknál gyakran találkozunk hasonló helyzettel, amikor is az optimumot adó megoldás – alkalmasan definiált – részeinek is rendelkezniük kell valamiféle optimalitási tulajdonsággal. Ez pedig rendszerint azért igaz, mert ha a részeken tudnánk javítani, akkor az javítást adna globális értelemben is. A jelenségre úgy szokás hivatkozni, hogy teljesül az *optimalitás elve*.

Nézzük ezután a módszer költségét! A T táblázat egy értékének a $(**)$ szerinti számítása $n-1$ összeadást és ugyanennyi összehasonlítást (minimumkeresés n elem közül) igényel. Mindez összesen $O(n^3)$ elemi műveletet jelent.

A T táblázat kitöltésével egyidőben gondoskodhatunk a minimális utak nyomonkövetéséről is. A Dijkstra eljárásánál alkalmazott megoldáshoz hasonlóan elegendő feljegyezni minden j csúcsra egy legrövidebb $1 \rightsquigarrow j$ út utolsó előtti pontját. Ezt egy $P[1 : n]$ tömb alkalmazásával oldhatjuk meg. A $P[j]$ értéke legyen az eljárás futása során az addig ismert legrövidebb $1 \rightsquigarrow j$ út utolsó előtti pontja (kezdetben $P[j] = 1$ minden j-re). A részletek kimunkálását az olvasóra hagyjuk.

Végezetül megemlítjük, hogy a módszer mintegy $3n$ tárcella alkalmazásával is megvalósítható – szemben a kb. n^2-tel, amit az itt leírt változat használ.

6.3. Floyd módszere az összes csúcspár közötti távolság meghatározására

Ebben a részben először azt a problémát tárgyaljuk, hogy miként lehet egy irányított gráfban az összes pontpár távolságát meghatározni, majd ennek a problémának egy speciális esetét, amikor csak arra vagyunk kíváncsiak, hogy mely pontok között vezet egyáltalán irányított út. Végül pedig az első algoritmus egy alkalmazását mutatjuk be.

Tehát nézzük először az általános kérdést! Nemnegatív élsúlyok esetén nyilvánvaló, hogy ha a Dijkstra-algoritmust minden csúcsra mint forrásra lefuttatjuk, akkor az összes rendezett pontpár távolságát megkapjuk. Van azonban egy ennél közvetlenebb módszer, amelynek előnye, hogy akkor is működik, ha van a gráfban negatív élsúly (természetesen negatív összsúlyú irányított kör nem lehet). Ez a módszer R. W. Floyd nevéhez fűződik.

A probléma

Adott egy $G = (V, E)$ irányított gráf, és egy $c : E \to \mathbb{R}$ súlyfüggvény úgy, hogy a gráfban nincs negatív összsúlyú irányított kör. Határozzuk meg az összes $v, w \in V$ rendezett pontpárra a $d(v, w)$ távolságot.

6.3.1. Floyd módszere

Tegyük fel továbbra is, hogy $V = \{1, 2, ..., n\}$, és hogy a G gráf a C adjacenciamátrixával adott. A pontpárok távolságának kiszámítására egy szintén $n \times n$-es F mátrixot fogunk használni. Kezdetben $F[i,j] := C[i,j]$. Ezután egy ciklust hajtunk végre, amelynek k-adik lefutása után $F[i,j]$ azon $i \rightsquigarrow j$ utak legrövidebbjeinek a hosszát tartalmazza, amelyek közbülső pontjai k-nál nem nagyobb sorszámúak.

Az új $F_k[i,j]$ értékeket a következőképpen számíthatjuk ki a $k-1$-edik iteráció utáni $F_{k-1}[i,j]$ értékekből (itt az index csak F különböző pillanatbeli értékeire utal, és nem ennyi különböző mátrixra). Nyilván egy legrövidebb $i \rightsquigarrow j$ út, melyen a közbülső pontok sorszáma legfeljebb k, vagy tartalmazza a k csúcsot vagy nem. Ha igen, akkor ezen út első fele egy olyan legrövidebb $i \rightsquigarrow k$ út, ami mentén a közbülső pontok sorszáma legfeljebb $k-1$, tehát hosszát $F_{k-1}[i,k]$ már tartalmazza; második fele meg szintén egy ilyen tulajdonságú legrövidebb $k \rightsquigarrow j$ út, aminek a hossza $F_{k-1}[k,j]$. Használtuk itt, hogy a legrövidebb utak között vannak egyszerűek is (lásd a Feladatot a Bellman–Ford-nál), és itt is segítségünkre volt az optimalitás elve.[2]

Ha az út nem tartalmazza k-t, akkor $F_k[i,j] = F_{k-1}[i,j]$. Mivel $F_k[i,k] = F_{k-1}[i,k]$ és $F_k[k,j] = F_{k-1}[k,j]$, az algoritmust végrehajthatjuk az F mátrix egyetlen példányával.

FLOYD algoritmusa

(1) for $i := 1$ **to** n **do**
  for $j := 1$ **to** n **do**
    $F[i,j] := C[i,j]$
(2) for $k := 1$ **to** n **do**
  for $i := 1$ **to** n **do**
    for $j := 1$ **to** n **do**
      $F[i,j] := \min\{F[i,j], F[i,k] + F[k,j]\}$

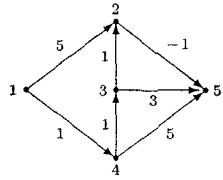
A megelőző gondolatmenet alapján k szerinti indukcióval azonnal adódik a következő, a Floyd-módszer „ciklusinvariánsát" megfogalmazó állítás.

Állítás: *$F[i,j]$ a (2)-beli iteráció k-adik menete után azon legrövidebb $i \rightsquigarrow j$ utak hosszát tartalmazza, amelyek belső csúcsai $1, 2, \ldots, k$ közül valók.* □

[2] A Bellman–Ford- és a Floyd-módszer a *dinamikus programozás* elnevezésű általános algoritmikus megközelítés jegyeit viseli. A dinamikus programozással később külön is foglalkozunk.

Az állítást a $k = n$ esetre alkalmazva következik a módszer helyessége, vagyis hogy végül $F[i,j] = d(i,j)$ igaz minden i,j csúcspárra. Az algoritmus lépésszáma n^3-bel arányos, hiszen a dominálő rész a három egymásba ágyazott **for** ciklus. Ez nagyságrendben ugyanannyi, mintha a Dijkstra mátrixos változatát minden csúcsra mint forrásra lefuttatnánk (pontosabb elemzéssel kiderül, hogy a Floyd-módszer mintegy 50%-kal gyorsabb). Ha a gráf éleinek száma jóval kisebb n^2-nél, és az élsúlyok nemnegatívak, akkor érdemesebb a Dijkstra-algoritmus éllistás változatát használni.

Példa: Nézzük, mihez kezd a Floyd-algoritmus a következő súlyozott élű G gráffal. Látható, hogy G-ben nincs negatív kör.



Az F tömböt négyzetes mátrixként ábrázoljuk. F_i jelentése: az F tömb i-edik iterációs lépés utáni állapota. Az utolsó mátrix a páronkénti legrövidebb utak hosszát tartalmazza.

$$F_0 = F_1 = \begin{pmatrix} 0 & 5 & \infty & 1 & \infty \\ \infty & 0 & \infty & \infty & -1 \\ \infty & 1 & 0 & \infty & 3 \\ \infty & \infty & 1 & 0 & 5 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad F_2 = \begin{pmatrix} 0 & 5 & \infty & 1 & 4 \\ \infty & 0 & \infty & \infty & -1 \\ \infty & 1 & 0 & \infty & 0 \\ \infty & \infty & 1 & 0 & 5 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix},$$

$$F_3 = \begin{pmatrix} 0 & 5 & \infty & 1 & 4 \\ \infty & 0 & \infty & \infty & -1 \\ \infty & 1 & 0 & \infty & 0 \\ \infty & 2 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}, \quad F_4 = F_5 = \begin{pmatrix} 0 & 3 & 2 & 1 & 2 \\ \infty & 0 & \infty & \infty & -1 \\ \infty & 1 & 0 & \infty & 0 \\ \infty & 2 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}.$$

Az $F_0 = F_1$ és $F_4 = F_5$ egyenlőségek azért igazak, mert sem 1, sem 5 nem lehet egy út közbülső csúcsa, hiszen 1-be nem fut él, és 5-ből nem indul ki él. Ugyanezekkel magyarázható a sok ∞ érték az első oszlopban és az utolsó sorban.

A legrövidebb utak nyomonkövetése

A legrövidebb utak nyomonkövetése megint csak egy menet közben karbantartott – itt már $n \times n$-es – P tömb segítségével történhet. Kezdetben $P[i,j] := 0$. Az algoritmushoz annyit kell hozzátenni, hogy ha a (2) ciklus legbelsejében egy $F[i,j]$ értéket megváltoztatunk, mert találtunk egy k-n átmenő rövidebb utat, akkor $P[i,j] := k$. Végül $P[i,j]$ egy legrövidebb $i \rightsquigarrow j$ út „középső" csúcsát fogja tartalmazni. Az $i \rightsquigarrow j$ út összeállításához tehát találnunk kell egy i-ből ebbe a közbülső csúcsba vezető legrövidebb utat, és egy ebből a csúcsból j-be vezető legrövidebb utat. De ezen utaknak is ismerjük egy-egy „középső" csúcsát, és így tovább. A következő program a P tömb segítségével kiír egy legrövidebb $i \rightsquigarrow j$ utat.

```
procedure legrövidebb út(i, j:csúcs);
var k:csúcs;
begin
    k := P[i, j];
    if k = 0 then return;
    legrövidebb út(i, k);
    kiír(k);
    legrövidebb út(k, j)
end;
```

Feladat: Mutassuk meg, hogy az előző kiíró program bizonyos P mátrixok esetén végtelen ciklusba kerülhet, de a Floyd-algoritmus eredményeként kapott P mátrixszal nem.

6.3.2. Tranzitív lezárás

A bemenet ismét egy $G = (V, E)$ irányított gráf. Most nem a legrövidebb utakra, hanem csak arra vagyunk kíváncsiak, hogy mely pontok között vezet irányított út. Persze használhatjuk Floyd módszerét is, hiszen ha az algoritmus lefutása után $F[i,j]$ értéke véges, akkor van $i \rightsquigarrow j$ út, ha pedig végtelen, akkor nincs. Kicsit egyszerűbb algoritmust kapunk, ha a G gráf A adjacencia-mátrixát (amiben csak 0 és 1 értékek szerepelnek) Boole-mátrixként fogjuk fel, és így a műveletek logikai műveletek lesznek. Ez az algoritmus megelőzte Floydét, és S. Warshall nevéhez fűződik.

Definíció (tranzitív lezárt): *Legyen $G = (V, E)$ egy irányított gráf, A az adjacencia-mátrixa. Legyen továbbá T a következő $n \times n$-es mátrix:*

$$T[i,j] = \begin{cases} 1 & \textit{ha } i\textit{-ből } j \textit{ elérhető irányított úttal;} \\ 0 & \textit{különben.} \end{cases}$$

Ekkor a T mátrixot – illetve az általa meghatározott gráfot – az A mátrix – illetve az általa meghatározott G gráf – (reflexív) tranzitív lezártjának *hívjuk.*

A probléma

Adott a (Boole-mátrixként értelmezett) A adjacencia-mátrixával a $G = (V, E)$ irányított gráf. Adjuk meg a G tranzitív lezártját.

Warshall algoritmusa tranzitív lezárásra

Floyd algoritmusában csak a következő változtatásokat kell tennünk. Az (1) ciklusban a kezdőértékek beállítása helyett

$$T[i,j] := \begin{cases} 1 & \text{ha } i = j \text{ vagy } A[i,j] = 1, \\ 0 & \text{különben.} \end{cases}$$

A (2) ciklusban F értékeinek változtatása helyett (ugyanazt megfogalmazva logikai műveletekkel)

$$(+) \qquad T[i,j] := T[i,j] \vee (T[i,k] \wedge T[k,j]).$$

A Floyd-algoritmussal kapcsolatos megállapításaink megfelelői itt is érvényesek. Jelesül a k-adik iteráció után $T[i,j] = 1$, azaz *igaz*, ha van olyan $i \rightsquigarrow j$ út, amelynek belső pontjai k-nál nem nagyobbak; és 0, azaz *hamis*, ha ilyen út nem létezik. A $(+)$ értékadás a (2) ciklus belsejében azt fejezi ki, hogy ilyen $i \rightsquigarrow j$ út akkor létezik, ha

1. már van olyan $i \rightsquigarrow j$ út, amely nem megy keresztül $k-1$-nél nagyobb sorszámú csúcson, vagy
2. vannak olyan $i \rightsquigarrow k$ és $k \rightsquigarrow j$ utak, amelyek nem mennek keresztül $k-1$-nél nagyobb sorszámú csúcson.

Az előző részhez hasonlóan itt is fenntarthatunk egy P tömböt, amiből gyorsan kiolvashatunk egy $i \rightsquigarrow j$ utat, ha ilyen létezik G-ben. Az algoritmus időigénye nyilvánvalóan $O(n^3)$.

6.3.3. Egy alkalmazás: centrum keresése irányított gráfban

A legrövidebb utak témájától elköszönőben a Floyd-módszer egyik alkalmazását mutatjuk be. Először is tisztázzuk, hogy mit értünk egy irányított gráf centrumán. Ezt egy analógiával készítjük elő.

Egy körlap minden x pontjához hozzárendelhetjük a körlap többi pontjától való távolságának $e(x)$ maximumát. Ezt nyilván úgy kaphatjuk meg, hogy meghúzzuk az x-en átmenő átmérőt. Az x két szakaszra osztja az átmérőt, és az $e(x)$ érték ezen szakaszok közül a nem rövidebbnek a hossza. A kör középpontja az a pont, melyre az $e(x)$ érték minimális. Ezzel párhuzamban értelmezzük egy gráf centrumát:

A súlyozott élű G irányított gráf v csúcsára legyen

$$e(v) = \max\{d(w, v) : \quad w \in V\}.$$

A v csúcs *centruma* G-nek, ha $e(v)$ minimális az összes $v \in V$ között.

Algoritmus centrum keresésére:
(1) Először Floyd módszerével kiszámítjuk a G-beli pontpárok közötti távolságokat. Ez $O(n^3)$ műveletet jelent.
(2) Az előző lépésben kapott F mátrix minden oszlopában meghatározzuk a maximális értéket (azaz kiszámítjuk az $e(v)$ értékeket). Itt n-szer keresünk maximumot n elem közül; ennek költsége $O(n^2)$.
(3) Végül megkeressük az n darab $e(v)$ érték minimumát. Ennek a műveletigénye $O(n)$.

A (3)-beli minimumot adó (bármelyik) v csúcs a G centruma lesz. A számítás legköltségesebb része az (1) lépés, így a műveletigény nagyságrendje $O(n^3)$.

6.4. Mélységi bejárás

Gráfokkal kapcsolatos algoritmikus feladatokban gyakran van szükségünk arra, hogy az élek mentén lépdelve valamilyen szisztematikus módon végigjárjuk a gráf csúcsait. Az erre a célra való *bejáró algoritmusok* központi jelentőségűek a gráfalgoritmusok között. Bejáró módszerek adják a gráf-struktúrákon működő összetett algoritmusok vezérlési szerkezetét azzal, hogy szabályozzák a sorrendet, amely szerint a csúcsokhoz kötődő munkákat elvégezzük. A bejáró módszerek különösen alkalmasak gráfok szerkezetének feltérképezésére; segítségükkel pontról pontra lépkedve összegyűjthetjük a kívánt információkat. Bináris fák bejárásaival (pre-, in- és postorder) már foglalkoztunk. Itt két általános, tetszőleges gráfra működő módszert ismerünk meg. Ezek a *mélységi bejárás* vagy *keresés* (depth-first-search, DFS) és a *szélességi bejárás* vagy *keresés* (breadth-first-search, BFS). Mindkettő számtalan gráfalgoritmus alapvázául szolgál.

Hogy szemléletes képet kapjunk a két stratégiáról, nézzük meg az öreg városka girbe-gurba utcáin bolyongó kopott emlékezetű lámpagyújtogató esetét. Minden

este szürkületkor elindul, hogy végigjárja a városka lámpáit (a gráf pontjai), de sosem tudja hol jár, csak arra emlékszik, hogy merről jött, és egy-egy lámpánál az onnan induló utácskákba (a gráf élei) bepillantva látja, hogy arrafelé a következő lámpa ég-e már. Tehát ahol épp éri az alkonyat, felgyújtja az útjába kerülő első lámpát, majd elindul az egyik – még nyilván sötét – utcán. Felgyújtja a következőt is, majd továbbmegy arra, ahonnan nem lát fényt. Ezt addig folytatja, míg egy olyan lámpához nem ér, ahonnan kiinduló összes utcából már fény dereng feléje. Ekkor visszamegy arra, amerről ide érkezett, és onnan próbál másfele, a még meg nem gyújtott lámpák irányába indulni. Ha már erre sincs ilyen utca, akkor még visszább megy. Egy idő után visszaérkezik a kiindulóhelyére, és megpróbál másfele elindulni. Ha már minden irányt végigjárt, újra visszajut a kezdőpontra, ahol mindenfelől lámpák fénye pislákol feléje, úgyhogy nyugodtan megy aludni. Módszerének lényege, hogy addig ment egyre „mélyebben" a városkába, ameddig csak volt olyan hely, ahol még nem járt.

A másik módszer szemléltetésére képzeljük el, hogy az egyik este a kopott emlékezetű lámpagyújtogató elhívja az összes kopott emlékezetű barátját – akik meglehetősen sokan vannak –, hogy segítsenek neki a lámpagyújtogatásban. A szenilis barátok – miután az első lámpát közösen meggyújtották – annyifelé oszlanak, ahány utca indul onnan. Meggyújtják az összes szomszédos lámpát, majd tovább osztódnak, és indulnak a még sötét utcák felé. Tehát így „széltében" terjeszkedve özönlik el a várost.

Hogy lássuk a két stratégia közötti lényeges különbséget, nézzük felülről a várost. Az első módszer szerint haladva a kiindulóponttól viszonylag távoli helyeken már akkor is lesz fény, amikor még a kezdőpont környékén van sötét lámpa. A második viszont a kezdőpont kis környezetében gyújt először teljes világosságot.

Ez a példa rámutat arra is, hogy a szélességi bejárás hatékonyan párhuzamosítható, míg a mélységi bejárásra hasonló megoldást nem ismerünk. Természetesen az előbbinek is van párhuzamosságot nem használó implementációja, ahol a szomszédos pontokat nem egyszerre, hanem valamilyen sorrendben látogatjuk meg.

A szélességi bejárás megoldja a legrövidebb utak problémáját egy forrásból abban a speciális esetben, amikor minden él súlya egy. Először feltérképezi a forrástól egységnyi távolságra levő csúcsokat, majd minden további menetben az eggyel távolabb levőket.

Ebben a részben a mélységi bejárást tárgyaljuk részletesebben, néhány hozzá kapcsolódó fogalommal, algoritmussal és alkalmazással együtt.

6.4.1. Irányított gráfok mélységi bejárása

A mélységi bejárás egyfajta mohó menetelés. Egy kezdőpontból kiindulva addig megyünk tovább irányított élek mentén, ameddig már nem tudunk még be nem

járt csúcsba jutni. Ez esetben visszamegyünk az út utolsó előtti pontjáig, és onnan próbálkozunk másfele tovább lépni és előremenni. Ha ezek a lehetőségek is kimerültek, még eggyel visszamegyünk, és így tovább. Azt, hogy a v pontból már nem tudunk előre lépni, úgy fejezzük ki, hogy a v pont mélységi bejárása befejeződött. Előfordulhat, hogy már a kezdőpont bejárását is befejeztük, de a gráfnak még mindig van olyan csúcsa, ahol nem jártunk. Ez azt jelenti, hogy a kezdőpontunkból nem érhető el minden pont irányított úton. Ekkor egy új kezdőpontot választunk a még be nem járt csúcsok közül, és innen folytatjuk a gráf mélységi bejárását.

Legyen $G = (V, E)$ egy irányított gráf, ahol $V = \{1, \ldots, n\}$. Jelölje $L[v]$ a v csúcs éllistáját ($1 \leq v \leq n$). Vegyünk föl továbbá egy bejárva$[1 : n]$ logikai vektort, amely minden ponthoz megadja, hogy jártunk-e már ott. A G gráf mélységi bejárását a következő programmal valósíthatjuk meg:

```
procedure bejár (* elvégzi a G irányított gráf mélységi bejárását *)
    begin
        for v := 1 to n do
            bejárva[v] := hamis;
        for v := 1 to n do
            if bejárva[v] = hamis then
              mb(v)
    end


procedure mb (v: csúcs)
    var
        w: csúcs;
    begin
(1)     bejárva[v] := igaz; (* meggyújtjuk a lámpát *)
        for minden L[v]-beli w csúcsra do
(2)         if bejárva[w] = hamis then
              mb(w) (* megyünk a következő még sötét lámpához *)
    end
```

Minden csúcsra pontosan egyszer hívjuk meg az *mb* eljárást, és ott egy csúcs minden szomszédját pontosan egyszer vizsgáljuk meg. Ezért az algoritmus időigénye $O(n + e)$, vagy ami ugyanaz, $O(\max(n, e))$. Gondoljuk meg, hogy $e + n$ lépés pusztán a gráf beolvasásához is szükséges. Tehát ennél lényegesen gyorsabb bejáró eljárás nem képzelhető el. A módszer *lineáris idejű*, amin azt értjük, hogy a bemenet hosszával arányos időn belül végez.

A *bejár* eljárásban az esetleges új kezdőpont a legkisebb (sorszámú) még nem bejárt pont lesz. Ez a választás nem lényeges a módszer működése és időigénye szempontjából. A fontos csak annyi, hogy viszonylag kevés munkával találjunk új kiindulópontot.

Mélységi számok és befejezési számok

A mélységi bejárás során sok hasznos információt gyűjthetünk a gráfról. Ennek egyik módja, hogy a csúcsok mellé feljegyezzük az algoritmus néhány fontosabb történését. Itt kétféle ilyen eseménnyel foglalkozunk. A lámpás képet használva: rögzítjük azt az időpontot, amikor meggyújtjuk a lámpát, és azt is, amikor visszaérünk egy lámpához, aminek már minden szomszédja világít.

Mindezeket a *mélységi* és a *befejezési* számok segítségével valósítjuk meg. Felvesszük az mszám$[1:n]$ és bszám$[1:n]$ egész értékű vektorokat. Az mszám$[v]$ azt fogja megadni, hogy a v csúcsot a bejárás során hányadikként látogattuk meg. A bszám$[v]$ pedig azt fogja mutatni, hogy hányadikként fejeződött be a v csúcs mélységi bejárása.

Definíció (mélységi számozás): *A G irányított gráf csúcsainak egy* mélységi számozása *a gráf v csúcsához azt a sorszámot rendeli, mely megadja, hogy az mb eljárás (1) sorában a csúcsok közül hányadikként állítottuk bejárva$[v]$ értékét igaz-ra. A v csúcs mélységi számát mszám$[v]$ jelöli.*

Definíció (befejezési számozás): *A G irányított gráf csúcsainak egy* befejezési számozása *a gráf v csúcsához azt a sorszámot rendeli, mely megadja, hogy a csúcsok közül hányadikként ért véget az $mb(v)$ hívás. A v csúcs befejezési számát bszám$[v]$ jelöli.*

A mélységi és befejezési számok meghatározásához a kódot a következőkkel egészítjük ki: legyenek *msz* és *bsz* egész értékű változók, melyekben a már kiadott legnagyobb mélységi, illetve befejezési sorszámokat szándékozzuk tárolni. A *bejár* eljárás elején beállítjuk a kezdőértékeket. Ez az

$$\text{msz} := 0;$$
$$\text{bsz} := 0;$$

utasításokkal tehető meg. Ugyanezen eljárás első **for** ciklusában az

$$\text{mszám}[v] := 0;$$
$$\text{bszám}[v] := 0;$$

értékadásokkal kinullázzuk a két új tömböt. Írjuk továbbá az *mb* eljárás (1) sora után az

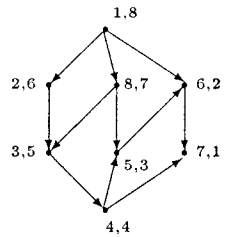
msz := msz + 1;
mszám[v] := msz;

utasításpárt, és a (2) utasítás után (vagyis az **end** elé) a

bsz := bsz + 1;
bszám[v] := bsz;

kódrészletet. A kiegészítésekkel nem növeljük meg lényegesen a futási időt, hiszen mindezek csúcsonként konstans mennyiségű extra munkát jelentenek. A módszer lépésszáma így is $O(n + e)$ marad.

Példa: A következő gráf csúcsai mellett feltüntettük az egyik lehetséges mélységi bejárása szerinti mélységi, illetve befejezési számokat. A kiindulópont a gráf legfelső csúcsa.



Feladat: Tegyük fel, hogy egy bináris fa mélységi bejárását végezzük a gyökértől indulva. Tegyük fel továbbá, hogy a fa élei lefelé, az apától a fiú felé irányítottak, és a csúcsok éllistáin mindig a bal fiú szerepel először (ha létezik egyáltalán). Mutassuk meg, hogy ekkor a csúcsok növekvő mélységi szám szerinti sorrendje éppen a preorder bejárás szerinti sorrend. Ugyanígy a csúcsok növő befejezési szám szerinti sorrendje megegyezik a postorder sorrenddel.

A gráf éleinek osztályozása és a mélységi feszítő erdő

A mélységi bejárás lehetővé teszi a gráf éleinek egy érdekes osztályozását. Hívjuk *faéleknek* azokat az éleket, melyek megvizsgálásukkor még be nem járt pontba mutatnak (azok az utcák, amikbe bepillantva még sötét van, ezért arra indulunk a következő lámpához).

Definíció (faél): *A $G = (V, E)$ irányított gráf $v \to w$ éle* faél *(az adott mélységi bejárásra vonatkozóan), ha megvizsgálásakor a (2) sorban a (bejárva[w] = hamis) feltétel teljesül.*

Jelölje T azt a gráfot, amelynek csúcshalmaza V (vagyis a kiinduló G gráf csúcshalmaza), élei pedig a faélek. Az algoritmus szerkezetéből nyilvánvaló, hogy egy csúcsba legfeljebb egy faél futhat be (egy lámpát csak egyszer gyújtunk meg).

Feladat: Mutassuk meg, hogy ha egy H összefüggő irányított gráfnak van olyan csúcsa, amelybe nem vezet él, és az összes többi csúcsába pontosan egy él fut be, akkor H körmentes (nincs benne kettőnél hosszabb egyszerű irányítatlan kör).

A feladat állítása szerint T egy erdő. A komponenseit olyan fák alkotják, melynek pontjait egy kezdőpont bejárása során értük el. Ennek megfelelően egy ilyen fa élei a kezdőponttól – amit a fa *gyökerének* is szokás nevezni – távolodóan irányítottak.

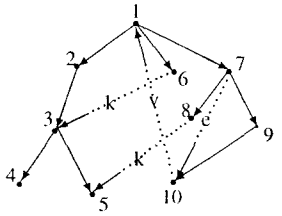
Definíció (mélységi feszítő erdő, feszítőfa): *Az előbb meghatározott T gráfot a G gráf egy* mélységi feszítő erdejének *nevezzük. Ha T csak egy komponensből áll, akkor* mélységi feszítőfáról *beszélünk.*

A gráf azon élei, melyek nem faélek, még további három csoportba sorolhatók. Ezt előkészítendő nevezzük a G gráf y csúcsát az x csúcs T-beli *leszármazottjának*, ha van $x \rightsquigarrow y$ irányított út T-ben. (Tehát x is leszármazottja önmagának.) Az x csúcs *részfáját* az x leszármazottai és a köztük menő faélek alkotják.

Definíció (élek osztályozása): *Tekintsük a G irányított gráf egy mélységi bejárását és a kapott T mélységi feszítő erdőt. (Ezen bejárás szerint) G egy $x \to y$ éle*

faél, *ha $x \to y$ éle T-nek;*
előreél, *ha $x \to y$ nem faél, y leszármazottja x-nek T-ben, és $x \neq y$;*
visszaél, *ha x leszármazottja y-nak T-ben (a hurokél is ilyen);*
keresztél, *ha x és y nem leszármazottai egymásnak.*

A következő ábrán egy gráf egy mélységi feszítőfáját, a csúcsok mélységi számozását és éleinek típusát láthatjuk:



Most nézzük, miként lehet hatékonyan felismerni az egyes éltípusokat. Egy v csúcs minden valódi leszármazottjának a mélységi száma nagyobb mszám$[v]$-nél, hiszen T élei mentén haladva a mélységi számok nőnek. Ezért egy előreél kisebb mélységi számú csúcsból nagyobb sorszámúba, egy visszaél pedig – a hurokélek kivételével – nagyobb sorszámú csúcsból kisebb sorszámúba mutat. Az ábra azt sugallja, hogy a keresztélek is nagyobb sorszámúból kisebb sorszámú csúcsba vezetnek. Valóban így is van: megmutatjuk, hogy ha $x \to y \in E$ és mszám$[x]$ <mszám$[y]$, akkor az $x \to y$ csak faél vagy előreél lehet. Ez esetben ugyanis az $mb(x)$ hívás időpontjában y-t még nem látogattuk meg. Az $x \to y$ élet mindenképpen megvizsgáljuk, mielőtt ez a hívás befejeződne. Ha a vizsgálat pillanatáig y-ban még nem jártunk, ezt az élet faélnek kell választanunk. Különben y az x-nek leszármazottja, ami azt jelenti, hogy az $x \to y$ él előreél.

Az eddigiek alapján – pusztán a mélységi számokat használva – a visszaéleket még nem tudjuk megkülönböztetni a keresztélektől. Segítségül hívjuk ezért a befejezési számokat. Ha az $x \to y$ él visszaél, vagyis x leszármazottja y-nak, akkor az x csúcsot nyilván y bejárása alatt látogatjuk meg, következésképpen az $x \to y$ él vizsgálatakor az y bejárása még nem fejeződött be. Ekkor tehát még bszám$[y] = 0$. Ha viszont $x \to y$ él keresztél, akkor x nincs benne az y részfájában. A már igazolt mszám$[y] <$ mszám$[x]$ egyenlőtlenség szerint y bejárása elkezdődött az x bejárása előtt. E két tényből következik, hogy y bejárása befejeződött, mielőtt x-et meglátogattuk. Az $x \to y$ él vizsgálatakor tehát már bszám$[y] > 0$.

Mindezek alapján a következő táblázat foglalja össze, hogy egy irányított gráf mélységi bejárása során hogyan tudjuk megkülönböztetni a négyféle éltípust:

$x \to y$ egy	ha az él vizsgálatakor
- faél	mszám$[y] = 0$
- visszaél	mszám$[y] \leq$ mszám$[x]$ és bszám$[y] = 0$
- előreél	mszám$[y] >$ mszám$[x]$
- keresztél	mszám$[y] <$ mszám$[x]$ és bszám$[y] > 0$.

A visszaéleknél egyenlőséget is megengedtünk. Ez azt jelenti, hogy az esetleges $x \to x$ hurokéleket visszaéleknek tekintjük. A táblázatban foglalt feltételek ellenőrzése állandó mennyiségű pluszmunkát ad élenként; az algoritmus ezzel kiegészítve is lineáris idejű marad:

Tétel: *A G irányított gráf mélységi bejárása – beleértve a mélységi, a befejezési számozást és az élek osztályozását is – $O(n + e)$ lépésben megtehető.* □.

A következő állítás a mélységi feszítő erdő részfáinak egy hasznos jellemzését adja. Tekintsük a $G = (V, E)$ irányított gráf egy T mélységi feszítő erdejét.

Legyen $x \in V$ egy tetszőleges csúcs, és jelölje T_x a feszítő erdő x-gyökerű részfájának a csúcshalmazát. A T_x elemei éppen az x leszármazottai a szóban forgó mélységi bejárásra vonatkozóan. Legyen továbbá

$$S_x = \left\{ y \in V \;\middle|\; \begin{array}{l} \text{van olyan } G\text{-beli } x \rightsquigarrow y \text{ irányított út, amelyen} \\ \text{a csúcsok mélységi száma legalább mszám}[x] \end{array} \right\}.$$

Az S_x halmaz tehát azokból a pontokból áll, amelyek x-ből elérhetők olyan $u \to v \in E$ élek mentén, melyek végpontjaira mszám$[u] \geq$mszám$[x]$, és mszám$[v] \geq$mszám$[x]$.

Állítás (részfa-lemma): *Tetszőleges $x \in V$ csúcs esetén érvényes a $T_x = S_x$ egyenlőség.*

Bizonyítás: T_x éppen azokból a pontokból áll, amelyek x-ből faélek mentén elérhetők. Az x-gyökerű részfa csúcsait x után érjük el a mélységi bejárás során, ezért a részfa bármely $u \to v$ élére mszám$[u] \geq$mszám$[x]$, és mszám$[v] \geq$mszám$[x]$. Ezzel beláttuk, hogy $T_x \subseteq S_x$.

A fordított irányú tartalmazás igazolására tegyük fel indirekte, hogy létezik egy $y \in S_x \setminus T_x$ csúcs. Legyen $x \rightsquigarrow y$ egy az S_x meghatározásában szereplő irányított út. Az $x \rightsquigarrow y$ út helyett esetleg annak egy kezdődarabját véve feltehetjük, hogy az út utolsó előtti v pontja T_x-ben van. A mélységi bejárás során x-szel kezdődően a T_x pontjait látogatjuk meg; a részfán kívüli pontokra ezért vagy x előtt, vagy T_x csúcsai után kerül sor. Az $y \in S_x$ feltétel szerint mszám$[y] >$mszám$[x]$. Ez $y \notin T_x$ miatt azt jelenti, hogy y-t valamikor a T_x pontjai után látogatjuk meg, amiből arra jutunk, hogy mszám$[y] >$mszám$[v]$. Ebből következik viszont, hogy $v \to y$ faél vagy előreél. Mindkét esetben $y \in T_x$ adódik, ellentmondva feltevésünknek. Ezzel beláttuk, hogy $S_x \subseteq T_x$ is igaz. □

Következmény: *Tegyük fel, hogy a $G = (V, E)$ gráf x csúcsából minden pont elérhető irányított úton. Tegyük fel továbbá, hogy a G mélységi bejárását x-szel kezdjük. Ekkor a mélységi feszítő erdő egyetlen fából áll.*

Bizonyítás: Ennél a bejárásnál mszám$[x] = 1$, amiből a feltétel alapján $S_x = V$. A részfa-lemma szerint tehát $T_x = V$. □

Feladat: Igazoljuk, hogy

$$T_x = \{y \in V \mid \text{mszám}[y] \geq \text{mszám}[x] \text{ és } \text{bszám}[y] \leq \text{bszám}[x]\}.$$

6.4.2. Irányított körmentes gráfok (DAG-ok)

Tennivalói közé tartozik a salétromozás utáni napon annak a pácnak a felvétele, amely marékszámra vett sóból, korianderből, borókamagból, sárga-cukorból, darált fokhagymából, babérlevélből és öt-hat fej vöröshagymából készült. KRÚDY GYULA: A húsvéti sódar titkai

Itt irányított gráfok egy olyan osztályával foglalkozunk, amely gyakran kerül elő a különféle alkalmazásokban. Másrészt arról is nevezetesek ezek a gráfok, hogy körükben néhány elemi algoritmikus feladat gyorsabban és egyszerűbben megoldható, mint általában.

Definíció (DAG): *Egy G irányított gráf* DAG (directed acyclic graph), *ha nem tartalmaz irányított kört.*

DAG-ok a tudomány és a tervezés számos területén felbukkannak. Gyakran hordoz valamiféle helyességgel, konzisztenciával kapcsolatos jelentést az, hogy a modellezésre használt irányított gráf DAG-e, vagy sem. Ízelítőül két példát mutatunk.

1. Teendők ütemezése. Képzeljük el, hogy egy összetett tevékenység a $T_1, \ldots, T_n$ részteendőkre bontható. Ezek általában nem függetlenek egymástól. Gyakran vannak olyan feltételeink, hogy bizonyos (i, j) párokra a T_j teendőt csak a T_i teendő elvégzése után hajthatjuk végre. Például ha a sódarkészítést egy összetett tevékenységnek fogjuk fel, akkor – Krúdy Gyula receptje szerint – a salétromozás előbb kell hogy legyen, mint a pácolás.

Az ilyen feltételrendszerek természetes módon leírhatók irányított gráffal. Ennek csúcsai a T_i teendők. A T_i-ből a T_j-be akkor vezet él, ha a T_i-t a T_j előtt kell elvégezni. A receptnél maradva: a salétromozás részteendőtől él vezet a pácoláshoz; a páckészítés bizonyos részfeladatai – mondjuk a koriander és a babérlevél hozzáadása – között pedig nem fut él egyik irányban sem.

Látni fogjuk később, hogy a teendőket akkor és csak akkor tudjuk a feltételeket kielégítő sorrendben végrehajtani, ha az itt vázolt gráf DAG. Az ilyen ütemezési gráfok élsúlyokkal gazdagított változatai a PERT-gráfok; ezekről még lesz szó a későbbiekben.

2. Várakozási gráfok. Temérdek olyan számítógépes rendszer van, ami egyidőben több felhasználót szolgál ki. Ilyennek tekinthető például egy légitársaság helyfoglalási adatbázisa. A tárolt adatokkal egyszerre több program – szokásos kifejezéssel: tranzakció – dolgozik. Hogy ne legyen nagy kavarodás (mondjuk hogy több

utas ugyanarra a helyre kap jegyet), egy tranzakció zárolja az adatbázis azon részeit, amelyekkel dolgozni kíván. A zár időtartama alatt más tranzakció nem férhet ezekhez. Ha a T_2 tranzakciónak egy olyan A adatra volna szüksége, amit a T_1 használ, és ezért zárol, akkor T_2 vár amíg A felszabadul. Vannak azonban olyan – pattnak nevezett – helyezetek, amikor a várakozás nem segít: amikor néhány tranzakció „körben vár egymásra", és ezért egyikük sem tudja folytatni a munkát. A tranzakciók munkáját felügyelő program a patthelyzet észlelése után kilövi az egyik érintett T-t, hogy a többiek tovább dolgozhassanak.

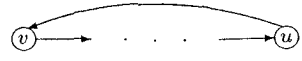
A pattok felismerésére szolgál a *várakozási gráf*. Ez egy irányított gráf, aminek csúcsai az aktív tranzakciók. A T_1 tranzakcióból él fut a T_2-be, ha van olyan A adat, amit T_2 használni (pl. írni vagy olvasni) szeretne, és amit T_1 éppen zár alatt tart. A tranzakcióink között pontosan akkor nincs patthelyzet, ha a várakozási gráf egy DAG. Ez közvetlenül adódik a topologikus rendezésről szóló tételből, amit mindjárt tárgyalni fogunk.

A DAG-tulajdonság ellenőrzése

Fontos tehát, hogy egy irányított gráfról el tudjuk dönteni, tartalmaz-e irányított kört. Ha a gráf egy mélységi bejárása során találunk visszaélet, akkor a gráf nyilván tartalmaz irányított kört, azaz nem DAG. Ugyanis az $u \to v$ visszaél v pontjából vezet (csupa faélekből álló) út u-ba. A következő egyszerű tétel azt mondja ki, hogy ez visszafele is igaz, így a gráf mélységi bejárása elegendő a DAG-tulajdonság ellenőrzéséhez.

Tétel: *Legyen $G = (V, E)$ egy irányított gráf. Ha G egy DAG, akkor egyetlen mélységi bejárása során sincs visszaél. Fordítva: ha G-nek van olyan mélységi bejárása, amelyre nézve nincs visszaél, akkor G egy DAG.*

Bizonyítás: Az első állítást az előbb már beláttuk. A második indoklásához tegyük fel, hogy G nem DAG. Meg kell mutatnunk, hogy tetszőleges mélységi bejárása során lesz visszaél. A G nem DAG, így tartalmaz legalább egy irányított kört. Vegyük ennek a körnek azt a v csúcsát, melynek a szóban forgó mélységi bejárás során a legkisebb a mélységi száma, és vizsgáljuk meg a kör v-be mutató élét. Ennek kezdőpontja legyen u.



Mivel mszám$[v]$ < mszám$[u]$, ez az él csak vissza- vagy keresztél lehet. Azonban v-ből u elérhető irányított úton, nevezetesen a kör élein haladva. Ezen az úton a pontok mélységi száma nem kisebb, mint mszám$[v]$. Alkalmazható a részfalemma: u a v leszármazottja lesz a mélységi feszítő erdőben. Így az $u \to v$ él csak visszaél lehet. □

Ezek után egy irányított gráfról mélységi bejárás segítségével $O(n+e)$ idő alatt el tudjuk dönteni, hogy DAG-e, vagy sem. Ha a bejárás közben találunk visszaélet, akkor a gráf nem DAG, különben pedig igen.

DAG topologikus rendezése

Emlékezzünk vissza a teendők ütemezésének problémájára. Adva van tehát n darab teendőnk, vagyis az ezeket reprezentáló n darab csúcs. Egy $u \to v$ él azt jelenti, hogy az u teendőt a v előtt kell elvégezni. Ilyenkor nyilván a teendők olyan sorrendjét keressük, ahol ha egy teendőt előbb kell elvégeznünk egy másiknál, akkor ez a sorban előbb van. Így a teendőket ebben a sorrendben elvégezhetjük anélkül, hogy megsértenénk az élekben megtestesülő feltételeket.

Definíció (topologikus rendezés): *Legyen $G = (V, E)$ ($|V| = n$) egy irányított gráf. G egy* topologikus rendezése *a csúcsoknak egy olyan $v_1, \ldots, v_n$ sorrendje, melyben $x \to y \in E$ esetén x előbb van, mint y (azaz ha $x = v_i, y = v_j$, akkor $i < j$).*

Tétel: *Egy irányított gráfnak akkor és csak akkor van topologikus rendezése, ha DAG.*

Bizonyítás: $\Rightarrow$: Ha G nem DAG, akkor nem lehet topologikus rendezése, mert egy irányított kör csúcsainak nyilván nincs megfelelő sorrendje.

$\Leftarrow$: Tegyük fel, hogy G egy DAG. Először megjegyezzük, hogy G-ben van olyan csúcs, amibe nem fut be él. Ha ugyanis minden pontba menne él, akkor egy tetszőleges pontból kiindulva lépkedni tudnánk visszafelé az irányított éleken anélkül, hogy valahol is elakadnánk. E séta során egyszer „körbeérnénk", ami ellentmond a DAG-tulajdonságnak.

Ezután a csúcsok száma szerinti teljes indukciót alkalmazunk. Ha G-nek csak egy pontja van, akkor az állítás nyilván igaz. Tegyük fel, hogy $n-1$-pontú DAG-ok esetén is teljesül, és legyen G egy tetszőleges n pontú DAG. G-ből hagyjunk el egy olyan x csúcsot a belőle kiinduló élekkel együtt, amibe nem fut be él. A kapott gráf legyen G_1. Nyilván G_1 is DAG. Az indukciós feltevés szerint a G_1-nek létezik $w_1, \ldots, w_{n-1}$ topologikus rendezése. Mármost az $x, w_1, \ldots, w_{n-1}$ sorrend nyilván a G egy topologikus rendezése. □

A teendők elvégezhetőségének tehát tényleg szükséges és elegendő feltétele, hogy a hozzájuk rendelt gráf DAG legyen. Hasonló mondható a tranzakciós példánkról: a várakozási gráf topologikus rendezhetőségéből következik, hogy valamelyik T biztosan megkaphatja a hőn áhított zárat.

A tétel gondolatmenetében egy DAG-ok topologikus rendezésére szolgáló algoritmus lappang: *válasszunk ki első pontnak egy olyan csúcsot, amibe nem fut él*, majd a maradékból egy ilyen csúcsot másodikként; és így tovább. Az ötlet a hatékonyan kivitelezhető a *sor* adatszerkezet segítségével. A sor alapja egy *elem*ekből álló kettősen láncolt lista. Két alapművelet van: sorba(v, Q) a v elemet a Q sor végére illeszti; első(Q) pedig visszaadja és egyben kitörli Q-ból annak első elemét. Nyilvánvaló, hogy ezek a műveletek a lista elejét, illetve végét jelölő mutatókkal $O(1)$ költséggel megvalósíthatók. A sort szokás még FIFO-listának (first in, first out) is nevezni.

Most egy Q sort használó topologikus rendező algoritmus vázlata következik.

1. A (kezdetben üres) Q sorba illesszük be a G azon csúcsait, amelyekbe nem megy él.
2. Ha Q üres, akkor álljunk meg, különben legyen $u :=$ első(Q), és írjuk ki az u csúcsot.
3. Töröljük a gráfból az $u \to v$ éleket. Ha egy itt előforduló v csúcsba már nem megy él, akkor sorba(v, Q).
4. Menjünk vissza a 2. lépéshez.

Feladat: Mutassuk meg, hogy az előző algoritmus kiírási sorrendje tényleg a G topologikus rendezése, és hogy az eljárás $O(n + e)$ költséggel megvalósítható.

Ugyancsak lineáris költségű topologikus rendező eljárást kaphatunk a mélységi bejárás segítségével: végezzük el a G DAG egy mélységi bejárását, és írjuk ki G csúcsait a befejezési számaik szerint növekvő $w_1, \ldots, w_n$ sorrendben. Ez megtehető az $O(n + e)$ költségkorlát megtartása mellett. A w csúcsot akkor írjuk ki, amikor a bejárása befejeződött, vagyis az mb(w) hívás utolsó utasításaként.

Állítás: *A $w_n, w_{n-1}, \ldots, w_1$ sorrend a G DAG egy topologikus rendezése.*

Bizonyítás: Azt kell belátnunk, hogy ha $w_i \to w_j$ éle G-nek, akkor $i > j$. Ez azon múlik, hogy G-ben nincs visszaél. A $w_i \to w_j$ él tehát vagy fa- vagy előre- vagy keresztél. Ezek mindegyikéről tudjuk (élek osztályozása, részfa-lemma utáni feladat), hogy w_j bejárása előbb fejeződik be, mint w_i bejárása, vagyis $j =$ bszám$[w_j] <$ bszám$[w_i] = i$. A $w_n, \ldots, w_1$ sorrendben így w_i megelőzi w_j-t. □

Legrövidebb és leghosszabb utak

Mint már említettük, a DAG-ok egyik előnyös tulajdonsága, hogy több fontos velük kapcsolatos feladatra gyorsabb algoritmusok ismeretesek, mint irányított gráfokra általában. Nézzük például a legrövidebb utak problémáját egy forrással. Legyen $G = (V, E)$ egy éllistával adott, súlyozott élű DAG és $s \in V$. Az élsúlyok tetszőleges valós számok lehetnek. G-ben nincs negatív kör, mivelhogy egyáltalán nem tartalmaz irányított kört. A feladatot a Bellman–Ford-módszerrel $O(n^3)$ lépésben tudjuk megoldani; nemnegatív élsúlyok esetén pedig a Dijkstra-algoritmus időigénye $O((n + e) \log n)$.

Topologikus rendezést használva a feladat lineáris időben, azaz $O(n + e)$ lépésben megoldható. Tegyük fel tehát, hogy már meghatároztuk a G csúcsainak egy $x_1, x_2, \ldots, x_n$ topologikus rendezését. Feltehetjük, hogy $s = x_1$, mert csak a kisebbtől a nagyobb sorszámú csúcsok felé mehet él. A $d(s, x_i)$ távolságokra érvényes a következő összefüggés:

$$(\dagger) \qquad d(s, x_i) = \min_{(x_j, x_i) \in E} \{d(s, x_j) + c(x_j, x_i)\},$$

ahol $c(x_j, x_i)$ az $x_j \to x_i$ él súlya. Ezt használva $i = 1, 2, \ldots n$-re sorban kiszámíthatjuk a $d(s, x_i)$ távolságokat. A lényeges mozzanat az, hogy ha (x_j, x_i) él, akkor $j < i$, tehát $d(s, x_i)$ számításakor a $d(s, x_j)$ értékek már tényleg a rendelkezésünkre állnak.

Mibe kerül mindez? Az i-edik távolság számításakor b_i számú összeadást végzünk el, ahol b_i az x_i-be befutó élek száma. A minimum meghatározására költött összehasonlítások száma is legfeljebb b_i. Ezeket i-re összegezve: e összeadásra és legfeljebb e összehasonlításra van szükség. Meg kell oldanunk még egy szervezési problémát: adott i-re gyorsan el kell tudnunk érni az x_i csúcsba bemenő éleket. Erről szól a következő feladat.

Feladat: Mutassuk meg, hogy G éllistájából $O(n + e)$ lépésben megkaphatjuk a *fordított éllistáját*. Utóbbiban az x_i csúcs listáján az x_i-be menő éleket leíró cellák vannak (szemben az eredetivel, ahol is x_i listája az x_i-ből kiinduló éleket ábrázolja).

Összesen tehát – a topologikus rendezést és a listafordítást is beleértve – $O(n + e)$ művelettel kiszámíthatjuk a legrövidebb utak hosszát.

A legrövidebb utak helyett érdekelhetnek bennünket a másik végletet jelentő *leghosszabb utak* is. Egy súlyozott élű G irányított gráf u, v pontjaira legyen $l(u, v)$ a leghosszabb *egyszerű* irányított $u \rightsquigarrow v$ utak hossza. Azért szorítkozunk egyszerű utakra, mert különben a pozitív összsúlyú irányított kört tartalmazó gráfokra a definíció értelmetlen volna.

Az $l(u, v)$ mennyiségek meghatározása általában hírhedten időigényes feladat, aminek néhány rokonával (Hamilton-kör keresése, Utazó ügynök) találkozni fogunk a 8.7.5. részben. A nehéz feladat megszelídül, ha a bemenetet jelentő gráf DAG. Először is megjegyezzük, hogy ha G DAG, akkor az $l(u, v)$ definíciójában *egyszerű út* helyett elég szimplán csak *út*at írni. Egy DAG-ban ugyanis minden irányított út egyszerű. A leghosszabb utakra ezután nyilvánvalóan érvényes a (†) rekurzió megfelelője – minimumok helyett maximumokkal:

$$(\ddagger) \qquad l(s, x_i) = \max_{(x_j, x_i) \in E} \{l(s, x_j) + c(x_j, x_i)\}.$$

Ha G egy DAG, akkor a (‡) összefüggések alapján az $l(s, x_i)$ mennyiségeket lényegében ugyanazzal a módszerrel megkaphatjuk, mint a $d(s, x_i)$ távolságokat. A különbség csupán annyi, hogy minimumok helyett maximumokat kell számítani.

Tétel: *Ha G egy éllistával adott súlyozott élű DAG, akkor az egy forrásból induló legrövidebb és leghosszabb utak meghatározásának feladatai $O(n + e)$ lépésben megoldhatók.* □

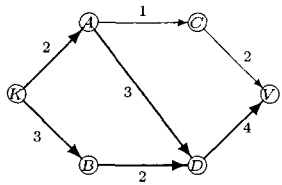
Feladat: Adjunk $O(n + e)$ lépésszámú módszert a legrövidebb és a leghosszabb utak nyomonkövetésére. (Használjuk a Dijkstra-módszernél alkalmazott gondolatot.)

A PERT-módszer

A PERT (Program Evaluation and Review Technique) elnevezésű tervezési módszertan súlyozott élű DAG-okkal ábrázolja egy összetett feladat részei közötti viszonyokat. A DAG csúcsai a nagy feladat részei, eleminek mondható teendői. Az x teendőből irányított él vezet az y teendőbe, ha x-et előbb kell elvégezni, mint y-t. A PERT-gráfokban külön csúcsok – legyenek ezek mondjuk K és V – jelölik a teljes feladat kezdetét és végét. Ha a gráf éleit értelmesen definiáljuk, akkor a csúcsok minden topologikus rendezésekor K az első, V pedig az utolsó csúcs lesz. Az éleken nemnegatív súlyok vannak. Az $x \to y$ él $c(x, y)$ súlya mondja meg, hogy mennyi időnek kell eltelnie az x tevékenység megkezdésétől az y megkezdéséig. Ilyen természetű Krúdy receptjében az a kikötés, hogy a salétromozás és a pácolás között egy napnak el kell telnie.

Hogyan határozhatjuk meg a súlyozott gráf ismeretében az x részfeladat legkorábbi kezdési időpontját? A válasz egyszerű: ez a leghosszabb $K \rightsquigarrow x$ út hossza, azaz $l(K, x)$ lesz. Speciális esetként a teljes feladat elvégzésének minimális időigénye $l(K, V)$. Ezek az idők az előző szakasz algoritmusával lineáris időben, vagyis $O(n + e)$ lépésben megkaphatók.

A PERT-módszertan fontos fogalmai a *kritikus út*, *él* és *csúcs*. Kritikus úton olyan $K \rightsquigarrow V$ utat értünk, amelynek a hossza $l(K, V)$. A kritikus élek és csúcsok pedig a kritikus utakon levő élek, illetve csúcsok. Az elnevezés onnan ered, hogy ha a kritikus utakon késve kezdődnek a tevékenységek, ez az egész munka befejezését késlelteti. A következő rajzon egy PERT-gráf szerepel. A kritikus éleket megvastagítottuk.



A C részfeladat nem kritikus. A legkorábbi kezdési ideje $l(K, C) = 3$, de az egész projekt befejezhető 9 időegység alatt akkor is, ha C-t a K után 7 egységgel indítjuk. A D csúcs viszont kritikus. Ha csak kicsivel is több, mint 5 időegység elteltével indítjuk, akkor az egész munka már nem fejezhető be 9 időegység alatt.

A kritikus csúcsokat és éleket a G PERT-gráf topologikus rendezésében visszafele, azaz V-től K felé haladva határozhatjuk meg. Első lépésként megjelöljük V-t mint kritikus csúcsot. Általában, tegyük fel, hogy az x_i csúcsnál tartunk, és x_i megjelölt (vagyis kritikus) csúcs. Ez esetben az $x_j \to x_i$ élet és az x_j csúcsot akkor kell megjelölnünk, ha (‡)-ban az $x_j \to x_i$ élen keresztül elérjük az $l(K, x_i)$ maximumot. Máshogy mondva: ha $l(K, x_i) = l(K, x_j) + c(x_j, x_i)$ teljesül. Az eljárás során minden élet egyszer veszünk szemügyre, így a költség $O(n + e)$. Érvényes tehát a következő:

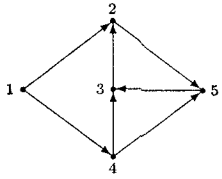
Tétel: *Egy éllistával adott PERT-gráf esetén a részfeladatok legkorábbi kezdési ideje, ezenfelül a kritikus élek és csúcsok lineáris időben meghatározhatók.* □

6.4.3. Erősen összefüggő (erős) komponensek

Irányított gráfoknál az összefüggőséget ugyanúgy definiáljuk, mint az irányítatlan gráfoknál: tekintet nélkül az élek irányítására. Ebben az értelemben az összefüggőségre úgy gondolhatunk, hogy a gráfunk egy darabból van. Irányított gráfokra van azonban egy ennél jóval szigorúbb összefüggőség-fogalom is:

Definíció (erősen összefüggő gráf): *Egy $G = (V, E)$ irányított gráf* erősen összefüggő, *ha bármely $u, v \in V$ pontpárra létezik $u \rightsquigarrow v$ irányított út.*

Ha valaki tévelygett már autóval egy idegen város egyirányú utcáinak szövevényében, akkor aligha fogja vitatni, hogy ez a követelmény erősebb, mint a „gyalogosokra szabott" egyszerű összefüggőség. A következő gráf összefüggő, de nem erősen összefüggő. Az 1 csúcsba nem juthatunk el irányított úton egyetlen más csúcsból sem.



Egy gráf összefüggő komponenseit a gráf csúcsain értelmezett ekvivalenciareláció ($u \sim v$, ha van u-ból v-be menő út) osztályaiként értelmeztük. Hasonlóan járhatunk el, ha az elérhetőségnél az élek irányát is figyelembe akarjuk venni.

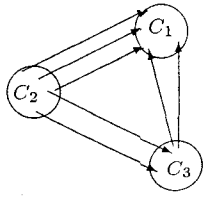
Definíció: *Legyen $G = (V, E)$ egy irányított gráf. Bevezetünk egy relációt V-n: $u, v \in V$-re legyen $u \approx v$, ha G-ben léteznek $u \rightsquigarrow v$ és $v \rightsquigarrow u$ irányított utak.*

Természetesen $v \approx v$, hiszen a nulla hosszúságú utat is útnak tekintjük. Nyilvánvaló az is, hogy a $\approx$ reláció szimmetrikus és tranzitív is. Következésképpen ekvivalenciareláció, ami osztályokba sorolja a gráf pontjait.

Definíció (erős komponensek): *A $\approx$ reláció ekvivalenciaosztályait a G* erősen összefüggő (*röviden:* erős) komponense*inek nevezzük.*

A definícióból azonnal látható, hogy ha C a G-nek erős komponense, akkor a C a pontjait összekötő élekkel együtt egy erősen összefüggő gráf. Az előző példa gráfjának három erős komponense van: $\{1\}$, $\{4\}$ és $\{2, 3, 5\}$.

Állítás: *Egy irányított gráf két erős komponense között az élek csak egy irányba mehetnek.*



Bizonyítás: Ha menne él a C_1 erős komponensből egy másik C_2-be és a C_2-ből a C_1-be is, akkor C_1 és C_2 ugyanabban az erős komponensben volna. □

Így ha egy irányított gráfban csak az erős komponensek viszonyára vagyunk kíváncsiak, akkor elég nyilvántartanunk, hogy mely komponensek között vezet él, és milyen irányban.

Definíció (redukált gráf): *Legyen $G = (V, E)$ irányított gráf. G* redukált gráfja *egy irányított gráf, melynek pontjai a G erős komponensei; a C_1, C_2 komponensek között akkor van $C_1 \to C_2$ él, ha G-ben a C_1 komponens valamely pontjából vezet él a C_2 komponensbe.*

A redukált gráf mindig DAG lesz. Ugyanis egy $C_1 \to C_2 \to \cdots \to C_k \to C_1$ irányított kör a redukált gráfban azt jelentené, hogy $C_1 \cup C_2 \cup \cdots \cup C_k$ a G ugyanazon erős komponensében van.

Most egy gyors (lineáris idejű) módszert mutatunk egy éllistával adott $G = (V, E)$ irányított gráf erős komponenseinek a feltérképezésére. Az algoritmus a mélységi bejáráson alapul, amit mindjárt nem is egyszer, hanem kétszer hívunk segítségül.

Erősen összefüggő komponensek meghatározása

(1) Mélységi bejárással végigmegyünk G-n, közben minden pontnak sorszámot adunk: a befejezési számát.
(2) Elkészítjük a G_{ford} gráfot, melyet úgy kapunk G-ből, hogy minden él irányítását megfordítjuk. Pontosabban: $G_{\text{ford}} := (V, E')$, ahol $u \to v \in E'$ akkor és csak akkor, ha $v \to u \in E$.
(3) Bejárjuk a G_{ford} gráfot mélységi bejárással, a legnagyobb sorszámú csúccsal kezdve (az (1)-beli befejezési számozás szerint). Új gyökérpont választásakor mindig a legnagyobb sorszámú csúcsot vesszük a maradékból.

Tétel: *A (3) pontban kapott fák lesznek G erős komponensei, azaz G-ben $x \approx y$ pontosan akkor igaz, ha x és y egy fában vannak.*

Bizonyítás: $\Rightarrow$: Azt kell igazolnunk, hogy egy erős komponens pontjai egy fába kerülnek a (3)-beli bejárás során. Ennél valamivel több is igaz: a gráf egy erős komponensének a pontjai *bármely* mélységi bejárásnál egyazon fába kerülnek. Ezt a G_{ford} gráfra alkalmazva – aminek nyilván ugyanazok az erős komponensei, mint G-nek – adódik az állítás.

Legyen tehát K egy erős komponens, és legyen x a K legkisebb mélységi számú pontja. Ekkor nyilván $K \subseteq S_x$, így a részfa-lemma miatt K benne van az x-ben gyökerező részfában.

$\Leftarrow$: Fordítva, tegyük fel, hogy x és y egy fában vannak a (3) pont szerinti mélységi bejárás után. Azt kell belátnunk, hogy ekkor $x \approx y$ a G gráfban, azaz x és y egymásból irányított úton elérhetők.

Legyen a v csúcs a gyökere annak a fának, mely a (3) pont szerinti mélységi bejárás után x-et és y-t is tartalmazza. Mivel x leszármazottja v-nek, ezért a G_{ford} gráfban van $v \rightsquigarrow x$ irányított út, tehát a G gráfban van egy L irányított út x-ből v-be.

Legyen x' az L-nek az a pontja, amelynek az első bejárás szerinti mélységi száma a legkisebb. A részfa-lemma miatt L-nek az $x' \rightsquigarrow v$ darabjában levő csúcsok az (1) bejárásnál x' leszármazottai lesznek. Az x' gyökerű részfában viszont az x' befejezési száma a legnagyobb, így v választása miatt $x' = v$. Az L pontjai közül tehát v-t látogattuk meg legelőször, és v-nek a befejezési száma volt a legnagyobb. Ezek együtt azt jelentik, hogy L pontjai a v leszármazottai az első bejárásnál. Így G-ben van $v \rightsquigarrow x$ irányított út.

Beláttuk, hogy a G gráfban $x \approx v$. Hasonlóan adódik, hogy $y \approx v$. A reláció szimmetrikus és tranzitív volta miatt $x \approx y$, azaz x és y a G egyazon erős komponensében vannak. □

Az első lépés $O(n+e)$ időt vesz igénybe. A gráf bejárása alatt kiírjuk a csúcsait befejezési szám szerinti sorrendben. (Emiatt a (3) lépésnél az új gyökér választásához nem kell külön rendezni őket.) A második lépés $O(e)$, a harmadik pedig $O(n+e)$ ideig tart. Tehát az algoritmus összidőigénye $O(n+e)$.

Feladat: Írjuk fel a G erős komponenseit abban a sorrendben, ahogy az előző algoritmus adja őket. Mutassuk meg, hogy az így kapott sorrend a G redukált gráfjának egyik topologikus rendezése.

6.4.4. Irányítatlan gráfok mélységi bejárása

A mélységi bejárás irányítatlan gráfokkal kapcsolatos feladatok megoldásában is hasznos. Egy irányítatlan gráf felfogható irányítottként úgy, hogy minden (u, v) élét az $u \rightarrow v$ és $v \rightarrow u$ élekkel helyettesítjük. Ezzel az átalakítással a 6.4.1. részben elmondottak irányítatlan gráfokra is értelmezhetők.

Az irányított gráfoknál bemutatott algoritmus tehát minden változtatás nélkül itt is működik. Segítségével a csúcsok számozásai (mélységi és befejezési), az élek osztályozása, a mélységi feszítő erdő $O(n+e)$ költséggel megkaphatók. A következő két észrevétel azt mondja, hogy irányítatlan gráfoknál a bejárással kapcsolatos tulajdonságok egyszerűbbek:

– Amikor a G irányítatlan gráfot irányítottá alakítjuk, az összefüggő komponenseiből erősen összefüggő gráfok lesznek. Ezért ha a C komponensből való x csúcsot választjuk a bejárás kezdőpontjául, akkor a részfa-lemma Következménye szerint az x részfájának csúcshalmaza pontosan C lesz. A mélységi feszítő erdő fái így a G összefüggő komponenseinek felelnek meg. A mélységi bejárás tehát irányítatlan gráfok komponenseinek feltérképezésére is alkalmas. Speciális esetként: ha G összefüggő, akkor $O(e)$ költséggel egy feszítőfát kapunk.

– Irányított gráfok bejárásakor négyféle éllel találkoztunk: fa-, előre-, vissza- és keresztéllel. Irányítatlan gráfoknál ebben is egyszerűbb a kép. A (v, w) irányítatlan él típusa legyen a $v \to w$ és $w \to v$ irányított élek közül annak a típusa, *amelyiket a bejáráskor először vizsgáljuk meg*. Az élek típusát így értelmezve egy irányítatlan gráf mélységi bejárásánál *csak faélek és visszaélek lehetnek*. Ennek bizonyítására vegyünk egy tetszőleges (v, w) élet. Az általánosság csorbítása nélkül feltehetjük, hogy az mb() eljárást a v csúcsra hívjuk meg előbb. Az mb(v) hívás során biztosan megvizsgáljuk a $v \to w$ élet. Ha a w-t eme vizsgálat előtt még nem látogattuk meg, akkor $v \to w$ és ezért (v, w) is faél lesz. Ellenkező esetben w az mb(v) eljárás folyamán vált látogatottá, tehát w leszármazottja lesz v-nek a fában. Ez esetben a w bejárása fejeződik be előbb; a $w \to v$ élet előbb látjuk, mint a fordított párját. Ennélfogva (v, w) visszaél lesz.

Alkalmazás: Artikulációs pont keresése

Definíció: *Legyen $G = (V, E)$ összefüggő irányítatlan gráf. A $v \in V$ csúcs* artikulációs *(más szóval* elvágó*)* pontja *G-nek, ha v és a rá illeszkedő élek elhagyásával a gráf több komponensre esik szét, vagyis elveszti összefüggőségét.*

Egy gráf artikulációs pontjait a mélységi bejárás segítségével találhatjuk meg. Hogy megértsük a következő módszert, képzeljünk magunk elé egy összefüggő gráfot, melynek már felépítettük a mélységi feszítőfáját. A gráf tehát csak faéleket és visszaéleket tartalmaz. Az utóbbiak a fa egy ágának két csúcsát kötik össze. Mivel keresztélek nincsenek, a fa két különböző ágát nem köti össze él. Ebből rögtön következik, hogy a fa gyökere pontosan akkor artikulációs pontja a gráfnak, ha egynél több fia van, hiszen ekkor elhagyásával a gráf a gyökér alatti részfáknak megfelelő darabokra esik szét.

Most vizsgáljuk meg, hogy mi történik, ha elhagyunk egy gyökértől különböző v csúcsot a gráfból. Ha csak a faéleket nézzük, akkor azt látjuk, hogy a feszítőfa szétesik azon részfákra, melyeknek gyökerei a v fiai, és a maradék feszítőfadarabra (ilyen van, mert v nem a gyökér). A visszaélek csak úgy tarthatják egybe ezeket a darabokat, ha a v alatti nem üres részfák mindegyikéből megy visszaél a v feletti feszítőfadarabba. Ez azt jelenti, hogy v-nek bármelyik ágán is indulunk el „lefelé"

a feszítőfában, egy visszaélen mindig „fel" tudunk jutni v egy valódi ősébe. Ha v-nek van akár egyetlen olyan fia, melynek részfájából nem vezet visszaél v „fölé", akkor v elhagyásával ez a részfa biztosan le fog válni a gráfról.

A $v \in V$ csúcsról el akarjuk dönteni, hogy vajon minden részfájából vezet-e visszaél egy valódi ősébe. E célból kiszámítjuk a fel$[v]$ értéket. Ez megadja a v csúcshoz annak a „feszítőfában legmagasabban levő" w csúcsnak a mélységi számát, amelyhez el tudunk jutni v-ből úgy, hogy „lefelé" megyünk (esetleg 0 darab) faélen, aztán egy visszaélen „felmegyünk" w-be. Legrosszabb esetben ez a csúcs maga v, különben pedig w mélységi száma kisebb lesz v mélységi számánál, hiszen ekkor w valódi őse v-nek. A v csúcs tehát akkor és csak akkor lesz artikulációs pont, ha van olyan w fia, melynek a fájából nem megy v fölé visszaél, vagyis melyre fel$[w] \geq$ mszám$[v]$. Ezek után foglaljuk össze egy kicsit gondosabban a tennivalókat.

Módszer az artikulációs pontok megkeresésére (összefüggő gráfban):

1. Végezzük el a gráf mélységi bejárását, és határozzuk meg a csúcsok mélységi számát, melyet $v \in V$-re mszám$[v]$ jelöl.

2. Számítsuk ki minden v csúcsra a fel$[v]$ értéket; ez megadja annak a legkisebb mélységi számú w csúcsnak a mélységi számát, melybe vezet visszaél v valamely leszármazottjából. Hogy ezt megvalósítsuk, járjuk be az első pontban kapott feszítőfát a befejezési számok szerinti sorrendben, és ebben a sorrendben töltsük ki a fel[ ] tömböt. Így amikor a fel$[v]$ értéket próbáljuk meghatározni, akkor v minden y fiára fel$[y]$ már ismert.

$$\text{fel}[v] = \min \left\{ \begin{array}{l} \text{mszám}[v], \\ \min\{\text{mszám}[z], \text{ ahol } v \to z \text{ visszaél}\}, \\ \min\{\text{fel}[y], \text{ ahol } y \text{ fia } v\text{-nek}\} \end{array} \right\}$$

3. Artikulációs pontok megkeresése: a feszítőfát bejárva a $v \in V$ csúcsokról ellenőrizzük, hogy elvágó pontok-e. A v-re vonatkozó teszt így hangzik:

(a) a gyökér pontosan akkor artikulációs pont, ha legalább 2 fia van a fában.

(b) a gyökértől különböző v csúcs akkor és csak akkor artikulációs pont, ha van v-nek olyan y fia, hogy fel$[y] \geq$ mszám$[v]$.

Mindhárom lépés gyakorlatilag gráf-, illetve fabejárás, ami $O(n + e)$ időt vesz igénybe. A gráf összefüggő, tehát $e \geq n - 1$, amiből az algoritmus összköltsége $O(e)$.

6.5. A szélességi bejárás

Elevenítsük föl a 6.4.-beli esetet, amikor a lámpagyújtogató elhívta a barátait lámpát gyújtogatni. Az általuk követett bejáró módszert próbáljuk meg átültetni tet-

szőleges irányított gráfra. Olyan bejárási sorrendet akarunk, ami szerint csak akkor foglalkozunk a kezdőponttól távolabbi csúcsokkal, ha a közelieket már mind láttuk. Meglátogatjuk tehát az első csúcsot, majd ennek a csúcsnak az összes szomszédját. Aztán ezen szomszédok összes olyan szomszédját, hol még nem jártunk, és így tovább. Hogyan lehet ezt értelmesen megszervezni? A legjobb ismert megoldás egy *sort* (FIFO-listát) alkalmaz. Ebbe rakjuk be az épp meglátogatott csúcsot, hogy majd a megfelelő időben a szomszédaira is sort keríthessünk.

A módszer általános lépésének lényege, hogy vesszük a sor elején levő x csúcsot. Ezt töröljük a sorból, meglátogatjuk azokat az y szomszédait, amelyeket eddig még nem láttunk, majd ezeket az y csúcsokat a sor végére tesszük. Az algoritmus keretét adó *bejár* eljárás tulajdonképpen ugyanaz, mint a mélységi bejárásé. A bejárva$[1 : n]$ tömb szolgál itt is a már látott csúcsok megjelölésére.

```
procedure bejár (* elvégzi a G irányított gráf szélességi bejárását *)
    begin
        for v := 1 to n do
            bejárva[v] := hamis;
        for v := 1 to n do
            if bejárva[v] = hamis then
              szb(v)
    end

procedure szb (v: csúcs)
    var
        Q: csúcsokból álló sor;
        x, y: csúcsok;
    begin
        bejárva[v] := igaz;
        sorba(v, Q);
        while Q nem üres do begin
            x := első(Q);
            for minden x → y ∈ E élre do
                if bejárva[y] = hamis then begin
                  bejárva[y] := igaz;
(*)               sorba(y, Q)
                end
        end
    end
```

A szélességi bejárás során – a mélységi mintájára – felépíthetjük a gráf egy *szélességi feszítő erdejét. Faélnek* itt is azokat az éleket hívjuk, melyek megvizsgálásukkor még be nem járt pontba mutatnak. Tehát amikor egy már meglátogatott x csúcs szomszédait vizsgáljuk, és mondjuk y-t közülük még nem láttuk, akkor az egyre növekvő erdőhöz hozzávesszük az $x \to y$ élet (az algoritmusban a $(*)$ sor után). Csakúgy, mint a mélységi bejárásnál, a faélek itt is feszítő erdőt alkotnak, aminek a fái természetes módon tekinthetők gyökeres, a gyökértől távolodóan irányított fáknak. E fák kapcsán most is osztályozhatjuk a gráf éleit.

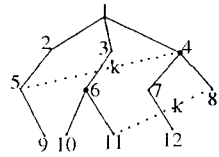
A G irányított gráf szélességi bejárása során a gráf egyik éle sem bizonyulhat előreélnek. Ugyanis amikor eljárásunk az x csúcsot kiveszi a sorból, hogy szomszédait megvizsgálja, akkor x minden még bejáratlan y szomszédját bejárttá nyilvánítja, a sor végére teszi, és az $x \to y$ él így faél lesz. A már bejárt y szomszédok pedig vagy x ősei, vagy egy másik ágon (esetleg egy másik fában) találhatók. Az előbbi esetben az $x \to y$ él *visszaél*, az utóbbiban pedig *keresztél* lesz.

A szélességi bejárás alkalmazható irányítatlan gráfokra is. A szokásos módon az (x, y) irányítatlan élet az $x \to y$ és az $y \to x$ irányított élpár helyettesíti. Az élek osztályozása a mélységi bejárásnál látottakhoz hasonlóan végezhető: az (x, y) irányítatlan él típusa legyen az $x \to y$ és $y \to x$ irányított élek közül annak a típusa, amelyikkel a bejáráskor először találkozunk. Ezzel a megállapodással élve az irányítatlan gráfok szélességi bejárása során *faéleken kívül csak keresztélek keletkeznek*. Nem szerepelhet előre- vagy visszaél. Legyen ugyanis (x, y) éle G-nek, és tegyük fel, hogy x őse y-nak (valamelyik fában). Nézzük, mi a helyzet akkor, amikor x kikerül a Q sorból. Ekkor y-ban még nem járhattunk, hiszen x fiai és így az összes leszármazottai csak ezután következnek. Tehát ekkor bejárva$[y] = \mathit{hamis}$; emiatt az algoritmus (x, y)-t faélnek választja.

Feladat: Legyen G irányítatlan gráf. Mutassuk meg, hogy G bármely szélességi feszítő erdejében a fák ponthalmazai éppen a G összefüggő komponensei. (Legyen v az egyik fa gyökere, w pedig tetszőleges pont a G-nek a v-t tartalmazó komponenséből. Tekintsünk egy v-ből w-be menő utat, és mutassuk meg, hogy ennek a pontjai a szélességi bejárás alatt mind bekerülnek a v fájába.)

A szélességi bejárás költsége $O(n+e)$, mert minden csúcsot pontosan egyszer teszünk be a sorba (amikor látogatottra állítjuk), és minden irányított élet egyszer vizsgálunk meg (amikor a kezdőpontja kikerül a sorból).

A következő ábrán egy összefüggő irányítatlan gráf szélességi feszítőfája látható. A faéleket sima, a keresztéleket szaggatott vonallal jelöltük. A csúcsok melletti szám pedig azt mondja meg, hogy hányadikként látogattuk meg őket (*szélességi számozás*).



Még egyszer a legrövidebb utakról

A szélességi bejárás alkalmazásával az egy forrásból induló legrövidebb utak feladata (lásd 6.2.) lineáris időben megoldható, ha az élsúlyok egységnyiek.

Legyen $G = (V, E)$ egy irányított gráf, és $s \in V$ egy rögzített pontja. Az egyszerűség kedvéért tegyük fel, hogy s-ből minden $v \in V$ csúcs elérhető irányított úton. Megmutatjuk, hogy a szélességi bejárás segítségével a $d(s, v)$ távolságok ($v \in V$) lineáris időben meghatározhatók. Jelentse $D[v]$ a v csúcsnak az s-től való távolságát az s-gyökerű szélességi fában. A $D[v]$ mennyiségek a bejárás során a futási idő nagyságrendjének megváltoztatása nélkül kiszámíthatók. Legyen ugyanis kezdetben $D[s] := 0$; az *szb* eljárásba pedig a $(*)$ sor után tegyük be a kézenfekvő $D[y] := D[x] + 1$; utasítást. A kiegészített algoritmus futási ideje nyilvánvalóan $O(n + e)$.

Tétel: *Az előzőek szerint módosított szélességi bejárás végeztével teljesülnek a következők:*
1. Legyen $s = x_1, x_2, \ldots, x_n$ a csúcsoknak a szélességi bejárás szerinti sorrendje. Ekkor $D[x_1] \leq D[x_2] \leq \ldots \leq D[x_n]$.
2. Ha $x \to y$ éle G-nek, akkor $D[y] \leq D[x] + 1$.
3. $D[v] = d(s, v)$ teljesül minden $v \in V$ csúcsra.

Bizonyítás: 1. Az algoritmusra pillantva azonnal látszik, hogy a csúcsok az $s = x_1, x_2, \ldots, x_n$ sorrendben kerülnek bele a Q sorba. Következésképpen ebben a sorrendben is kerülnek ki a sorból (FIFO-tulajdonság). Innen arra jutunk, hogy ha az $x \neq s$ csúcs előbb van a sorrendben, mint y, akkor $apa(x)$ nem lehet később, mint $apa(y)$, ahol az *apa* a szélességi feszítőfában értendő. Ezt használva egyszerű indukcióval kapjuk, hogy a $D[x_1], D[x_2], \ldots, D[x_n]$ számsorozat nem csökkenő. Ez ugyanis közvetlenül látszik a gyökérre és fiaira. Később pedig nyilván $D[x_i] = D[apa(x_i)] + 1$ és $D[x_{i+1}] = D[apa(x_{i+1})] + 1$. Ha itt a két apa különböző, akkor az indukciós feltevés miatt $D[apa(x_i)] \leq D[apa(x_{i+1})]$, amiből $D[x_i] \leq D[x_{i+1}]$. Ha pedig az apák megegyeznek, akkor $D[x_i] = D[x_{i+1}]$.

2. Tegyük fel, hogy $x \to y$ éle G-nek; meg kell mutatnunk, hogy ekkor $D[y] \leq D[x] + 1$. Nézzük ezért, hogy mi történik, amikor x kikerül a Q sorból, és éppen az (x, y) élet vizsgáljuk. Ha bejárva$[y] = hamis$, akkor y apja x,

vagyis $D[y] = D[x] + 1$. Ha pedig y-t már korábban láttuk, akkor arra következtethetünk, hogy y apja előbb van a szélességi sorrendben, mint x, amiből 1. szerint $D[apa(y)] \leq D[x]$. Az utóbbi egyenlőtlenség mindkét oldalához egyet adva kapjuk, hogy $D[y] \leq D[x] + 1$.

3. Világos egyrészt, hogy $d(s, v) \leq D[v]$ érvényes minden $v \in V$ csúcsra, mivelhogy a gráfban (sőt a fában) van $D[v]$ élből álló $s \rightsquigarrow v$ irányított út. Elég tehát a fordított $D[v] \leq d(s, v)$ egyenlőtlenségeket igazolni. Legyen evégből $s = y_0, y_1, \ldots y_k = v$ egy minimális hosszúságú G-beli irányított út s-ből v-be. A 2. észrevételt alkalmazzuk sorra az út éleire: $D[y_1] \leq D[s] + 1 = 1$, majd $D[y_2] \leq D[y_1] + 1 \leq 2$; így folytatva végül $D[v] = D[y_k] \leq k = d(s, v)$. Az érvelés ezzel teljes. □

A $D[v]$ számok 3. szerint a legrövidebb s-ből v-be menő utak hosszai. A szélességi bejárással tehát a legrövidebb utak feladatának ez a fontos speciális esete (minden élsúly 1) a Dijkstra-algoritmusnál gyorsabban[3], lineáris időben megoldható.

A tétel 1. és 3. állításai mondják ki, hogy az eljárás tényleg a lámpagyújtogató és barátai széltében terjeszkedő stratégiáját valósítja meg: a kezdőpont után először a tőle 1 távolságra levő csúcsokat látogatjuk meg, utána a 2 egységnyire levők jönnek, és így tovább.

Feladat: Legyen G egy éllistával adott irányítatlan gráf, és s egy csúcsa. Adjunk $O(e)$ költségű módszert legrövidebb (legkevesebb élből álló) s-et tartalmazó kör keresésére. (Indítsunk szélességi bejárást s-ből. Legyenek $s_1, s_2, \ldots, s_k$ az s szomszédai. Legyen f a bejárás során kapott első olyan keresztél, amely az s_i gyökerű részfák közül két különböző között fut. Ekkor f a végpontjait összekötő faélekkel együtt megfelelő kört ad. Az f megtalálásában segít, ha a bejárás során a csúcsok mellé feljegyezzük, hogy melyik s_i részfájába tartoznak. Ezt a jellemzőt a csúcsok az apjuktól öröklik.)

A szélességi bejárás két további alkalmazásával később, a párosítások, majd pedig a hálózati folyamok kapcsán ismerkedünk meg.

[3]Talán helyesebb úgy fogalmazni, hogy a szélességi bejáráson alapuló módszerünk gyorsabb, mint a Dijkstra korábban megismert éllistás implementációja. Valójában a most bemutatott algoritmust szemlélhetjük úgy, mint Dijkstra-módszer egy változatát. A már bejárt pontok halmaza a KÉSZ halmaz, amit a faélek mentén bővítgetünk. A tétel 1. és 3. állításából kiolvasható, hogy teljesül a Dijkstra-módszer bővítési elve: a KÉSZ-be mindig egy az s-hez legközelebbi $V \backslash$KÉSZ-beli pont kerül.

6.6. Minimális költségű feszítőfák

egy fa ága vége
nem nő rá a gyökerére
se más ágra
se törzsére
(körmentes)

egy hangya ha elindulna
bármely ágról eljuthatna
bármely ágra
bármely pontba
(összefüggő)

hangyánk ha nem boldogulna
lepottyanva
földön mászna
törzsön kúszna
(ez bizony már erdő volna)

LEHEL JENŐ

A következő két részben irányítatlan gráfokkal (röviden: gráfokkal) foglalkozunk. Ennek megfelelően $G = (V, E)$ egy irányítatlan gráfot jelöl, amelynek n csúcsa és e éle van. Ezután már csak egyszerű utakkal és körökkel lesz dolgunk. A továbbiakban ezért úton és körön mindig *egyszerű* utat és kört értünk. Ebben a részben a minimális költségű feszítőfák keresésének problémáját fogjuk tárgyalni. Először tisztázzuk az alapfogalmakat.

Definíció (minimális költségű feszítőfa): *Legyen $G = (V, E)$ egy összefüggő gráf. A G gráf egy körmentes összefüggő $F = (V, E')$ részgráfja a gráf egy* feszítőfája. *Legyen továbbá az éleken értelmezve egy $c : E \to \mathbb{R}$ súlyfüggvény. Ekkor a G gráf egy F feszítőfája* minimális költségű, *ha költsége (a benne szereplő élek súlyainak összege) minimális G összes feszítőfája közül.*

Egy feszítőfa tehát egy olyan fa, ami a G minden pontját tartalmazza, és az élei a G élei közül valók.

A probléma

Adott egy $G = (V, E)$ összefüggő irányítatlan gráf, és az élein értelmezett $c : E \to \mathbb{R}$ súlyfüggvény. Határozzuk meg a G egy minimális költségű feszítőfáját.

A feladat természetesen merül fel olyan helyzetekben, amikor a G egy (u, v) éle az u és v közötti közvetlen kapcsolat lehetőségét jelenti, az él súlya pedig a kapcsolat kiépítésének, esetleg működtetésének valamiféle költségét. Egy olyan élrendszert keresünk, ami biztosítja, hogy bárhonnan eljuthatunk bárhová; ezek közül pedig a lehető legkisebb költségűt szeretnénk megtalálni (ld. a következő állítást is). Tudomásunk szerint a probléma első érdemi megoldását Otakar Borůvka brnói professzor közölte 1926-ban. A kérdéssel Morvaország nyugati részének a villamosítása kapcsán találkozott: adott helyeket összekötő minimális összhosszúságú vezetékrendszert kellett terveznie.

Mielőtt az algoritmusok taglalásába fognánk, összefoglalunk néhány fákkal kapcsolatos egyszerű tényt.

Állítás:
1. Minden legalább kétpontú fában van olyan csúcs, amiből csak egy él megy ki (elsőfokú csúcs).
2. Bármely összefüggő gráf tartalmaz feszítőfát.
3. Egy n-pontú összefüggő gráf akkor és csak akkor fa, ha $n-1$ éle van.
4. Egy fa bármely két pontja között pontosan egy út vezet.
5. Legyen G egy súlyozott élű összefüggő gráf, F egy minimális költségű feszítőfája. Legyen $g=(u,v)$ a G-nek egy olyan éle, ami nem éle F-nek, és tegyük fel, hogy az F-beli u-ból v-be vezető úton van olyan g' él, amelyre $c(g) \leq c(g')$. Ekkor az F-ből a g hozzávételével és a g' elhagyásával adódó F' gráf is egy minimális költségű feszítőfa G-ben.

Bizonyítás: 1. A DAG-ok topologikus rendezhetőségéről szóló tétel ötlete működik itt is. Lépkedjünk a fa élei mentén, ügyelve arra, hogy ne használjuk kétszer egymás után ugyanazt az élet. A körmentesség miatt nem érhetünk vissza korábban már látott pontba. Így lesz olyan csúcs, amiből nem tudunk továbblépni. Ez egy elsőfokú csúcs.

2. Ha a gráfban van legalább 3 pontú egyszerű kör, akkor hagyjuk el ennek egy (u,v) élét. A gráf ezután is összefüggő marad; u-ból v továbbra is elérhető a körbeli kerülő úton. Ezt ismételgetve végül körmentes összefüggő gráfot, azaz egy feszítőfát kapunk.

3. Először belátjuk, hogy egy n-pontú fának $n-1$ éle van. Ugyanis ha a fából törlünk egy elsőfokú csúcsot a hozzá csatlakozó éllel együtt, akkor egy $n-1$-pontú fát kapunk. Ebből is törölhetünk egy ilyen lógó élet. Ezt ismételve $n-1$ lépés után az élek elfogynak, hiszen csak egy csúcs marad.

Fordítva: legyen F egy n-pontú, $n-1$-élű összefüggő gráf. Legyen F' ennek egy feszítőfája (lásd 2.). Az előzőleg igazolt állítás szerint F'-nek is $n-1$ éle van, amiből $F=F'$.

4. Tegyük fel, hogy az u pontból az u' pontba két út vezet. Legyen $f=(v,w)$ az egyik út olyan éle, ami nincs a másik úton. Nyilvánvaló ekkor, hogy f nélkül is eljuthatunk v-ből w-be, vagyis a gráf az f törlése után is összefüggő marad. Ez pedig 3. szerint képtelenség; egy fa az egyik élének a törlése után nem maradhat összefüggő.

5. Az $F \cup \{g\}$ gráfban van olyan kör, amelynek g' éle. A g' törlésével kapott F' gráf tehát összefüggő marad. A csúcsainak száma ugyanannyi, mint F-é, így 3. szerint egy feszítőfája G-nek. Végezetül pedig nyugtázzuk, hogy F' költsége nem lehet nagyobb, mint az F költsége, hiszen az előbbi az utóbbiból a nem pozitív $c(g)-c(g')$ mennyiség hozzáadásával kapható meg. □

A piros-kék algoritmus

A minimális feszítőfák keresésére való algoritmusok legtöbbje egy tőről származtatható. E módszerek közös vonása, hogy valamilyen módon sorra nézik a G éleit; bizonyosakat bevesznek a végül kialakuló minimális feszítőfába, másokat pedig eldobnak. A módszerek működését érdemes úgy szemlélni, hogy színezzük a G éleit[4]. Két színt használunk: a *kék* élek belekerülnek a végeredményt jelentő minimális feszítőfába, a *pirosak* pedig nem. Az élek festegetése során ügyelni fogunk arra, hogy az eddig kialakult (részleges) színezés mindig *takaros* legyen.

Definíció (takaros színezés): *Tekintsük a súlyozott élű G gráf éleinek egy részleges színezését, melynél bármely él piros, kék vagy színtelen lehet. Ez a színezés* takaros, *ha van G-nek olyan minimális költségű feszítőfája, ami az összes kék élet tartalmazza, és egyetlen piros élet sem tartalmaz.*

A színezés során két szabályt használunk. Ez annyit tesz, hogy egy élet csak akkor festünk be, ha a szabályok egyike alkalmazható.

KÉK SZABÁLY:	Válasszunk ki egy olyan $\emptyset \neq X \subset V$ csúcshalmazt, amiből nem vezet ki kék él. Ezután egy legkisebb súlyú X-ből kimenő színezetlen élet fessünk kékre.
PIROS SZABÁLY:	Válasszunk G-ben egy olyan egyszerű kört, amiben nincs piros él. A kör egyik legnagyobb súlyú színtelen élét fessük pirosra.

Kezdetben G-nek nincs színes éle. Ebből a helyzetből indulva a két szabályt tetszőleges sorrendben és helyeken alkalmazzuk, amíg csak lehetséges. Nevezzük ezt a nagyvonalú algoritmust *piros-kék algoritmus*nak. Első látásra talán meglepő, hogy mindenképpen célt érünk, függetlenül attól, hogy milyen sorrendben színezzük az éleket. Ezt fejezi ki a következő két állítás:

Tétel: *A piros-kék eljárás működése során mindig takaros színezésünk van. Ezen felül a színezéssel sosem akadunk el: végül G minden éle színes lesz.*

Bizonyítás: Először belátjuk, hogy a színezés mindig takaros. Ez kezdetben, amikor még minden él színtelen, nyilván teljesül. Tegyük fel mármost, hogy egy takaros színezésünk van. Legyen F a G egy olyan minimális költségű feszítőfája, amely minden kék élet tartalmaz, és egyetlen pirosat sem. Tegyük fel továbbá,

[4]Az algoritmusoknak ezt az interpretációját Robert E. Tarjan javasolta.

hogy ebben a helyzetben a gráf f élét festjük be. Két eset van aszerint, hogy melyik szabályt használjuk:
(a) *A kék szabályt használjuk.* Ekkor f nyilván kék lesz. Ha f éle F-nek, akkor F maga mutatja, hogy a színezés takaros marad. Ha f nem éle F-nek, akkor nézzük azt az $X \subset V$ csúcshalmazt, amire a kék szabályt alkalmaztuk. Az F-ben van olyan út (mert feszítőfa), ami az f két végpontját összeköti. Ezen az úton pedig van olyan f' él, ami kimegy X-ből, ugyanis f kilép X-ből. Az F választása miatt f' nem lehet piros. A kék szabály szerint kék sem lehet, továbbá $c(f') \geq c(f)$ is teljesül. Legyen F' az F-ből az f' törlésével és az f hozzáadásával kapott gráf. A $g = f$ és $g' = f'$ választással alkalmazható az állítás 5. pontja. Eszerint F' egy minimális feszítőfa. Az F' mutatja, hogy az f befestése utáni színezés is takaros.
(b) *A piros szabályt használjuk.* Ekkor f piros lesz. Ha f nem éle F-nek, akkor F tanúsítja a bővebb színezés takarosságát is. Ha $f \in F$, akkor az f törlésével az F két komponensre esik. Pillantsunk most arra a körre, amelyre a piros szabályt alkalmaztuk. Ennek van olyan $f' \neq f$ éle, ami a két komponens között fut. A régi színezés takarossága és a piros szabály miatt az f' színtelen és $c(f') \leq c(f)$. Az f' végpontjait összekötő F-beli út tartalmazza az f élet. Az 5. ismét alkalmazható ($g = f', g' = f$): az F-be f helyett f'-t véve a kapott F' egy minimális költségű feszítőfa lesz, ami szavatolja, hogy az új színezés is takaros.

Nézzük a második állítást: tegyük fel, hogy van még egy f színtelen él. A színezés takarossága miatt a kék élek egy erdőt alkotnak. Ennek az erdőnek az összefüggő komponenseit a továbbiakban *kék fáknak* fogjuk nevezni. Ha f végpontjai ugyanabban a kék fában vannak, akkor a piros szabály alkalmazható arra körre, aminek az élei f és az f végpontjait összekötő (egyetlen) kék út élei. Ha f különböző kék fákat köt össze, akkor pedig a kék szabály működik; X legyen az egyik olyan fa csúcshalmaza, amihez f csatlakozik. (Ez utóbbi esetben nem biztos, hogy f fog színt kapni a következő lépésben.) A színezés tehát folytatható. □

Következmény: *Ha a piros-kék algoritmussal befestjük az összefüggő $G = (V, E)$ gráf minden élét, akkor a kék élek egy minimális költségű feszítőfa élei. Sőt, ez már akkor is igaz, amikor van $|V| - 1$ kék élünk (és esetleg van még színezetlen él).*

Bizonyítás: Az első állítás azonnal következik abból, hogy a végső színezés is takaros. A második pedig az elsőből: végül összesen $|V| - 1$ kék él lesz. Ha már van ennyi, akkor több nem keletkezhet. □

A piros-kék algoritmus tanulsága, hogy bárhol és bármikor is alkalmazzuk sorra a két szabályt, végül mindenképpen egy minimális költségű feszítőfát kapunk. A recept *helyessége* szempontjából tehát közömbös a sorrend. Mint később

látni fogjuk, a *hatékonyság* szempontjából viszont nem. A következő nevezetes algoritmusok a piros-kék módszer pontosított változatainak tekinthetők.

PRIM MÓDSZERE: Legyen s a G egy rögzített csúcsa. Minden egyes színező lépéssel az s-et tartalmazó F kék fát bővítjük. Kezdetben az F csúcshalmaza $\{s\}$, végül pedig az egész V. A következő kék élnek az egyik legkisebb súlyú élet választjuk azok közül, amelyek F-beli pontból F-en kívüli pontba mennek.

KRUSKAL MÓDSZERE: A következő befestendő f él legyen mindig a legkisebb súlyú színtelen él. Ha az f két végpontja ugyanazon kék fában van, akkor az él legyen piros, különben pedig kék.

Prim algoritmusa minden lépésben a kék szabályt használja; X-nek választható az F csúcshalmaza. A Kruskal-módszer a piros szabály szerint színez, amikor f-et pirosra festi. A szabály arra az egyszerű körre alkalmazható, aminek az élei f és a két végpontját összekötő kék út élei. Ha pedig f az F_1 és F_2 kék fákat köti össze, akkor X-et az F_1 csúcshalmazának választva teljesülnek a kék szabályban foglalt feltételek. Az előzőek szerint mindkét módszer végül egy minimális költségű feszítőfát ad.

Valamivel bonyolultabb stratégiát követ a Borůvka-módszer, aminek a működése menetekre osztható. Egy menetben több élet választunk ki, és azok mindegyikét kékre színezzük. Az egyszerűség kedvéért itt tegyük fel, hogy G-nek nincs két egyforma súlyú éle.

BORŮVKA MÓDSZERE: Minden egyes kék fához válasszuk ki a legkisebb súlyú belőle kimenő (színtelen) élet. Színezzük kékre a kiválasztott éleket.

Egy élet két fához is kiválaszthatunk; természetesen ekkor is csak egyszer kell kékre festeni. Ez a módszer is a piros-kék algoritmus specializált változta. Tegyük fel ugyanis, hogy az algoritmus valamelyik menetében az $f_1, f_2, \ldots, f_k$ éleket választotta ki. A számozás esetleges megváltoztatása árán feltehetjük, hogy az éleket súly szerint növekvően írtuk fel, azaz $c(f_1) < c(f_2) < \cdots < c(f_k)$. Jelöljön továbbá F_i egy olyan kék fát, amihez f_i-t választottuk, és legyen X_i az F_i csúcshalmaza. A Borůvka-módszer menete, amely az f_i éleket kékre festi, úgy tekinthető, hogy az X_i $(i = 1, 2, \ldots, k)$ halmazokra sorra alkalmazzuk a kék szabályt. Annyit kell csak észrevenni, hogy az $f_1, f_2, \ldots, f_{i-1}$ élek egyikének sincs pontja X_i-ben, ellenkező esetben ugyanis nem az f_i-t választottuk volna F_i-hez. Az f_i él színezése előtt X_i-ből nem megy ki kék él, így a kék szabály szerint f_i befesthető.

A módszer hatékonyan párhuzamosítható. Egy menetben az élek kiválasztása és színezése párhuzamosan végezhető. A menetek száma pedig igen kedvező korlát alatt marad.

Feladat: Mutassuk meg, hogy a Borůvka-módszer meneteinek száma legfeljebb $\log_2 n$; ennyi menet után már csak egy kék fa létezhet.

Az előzőekben bemutattuk három fontos feszítőfa-algoritmus lényegi lépéseit. A módszerek helyessége következik abból, hogy mindegyik a piros-kék algoritmus speciális esete. Ezután a Prim- és a Kruskal-algoritmusok hatékony megvalósításával foglalkozunk. Itt az adatszerkezetek játsszák a főszerepet.

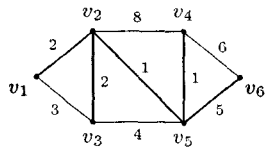
6.6.1. Prim módszere

Legyen $G = (V, E)$ egy összefüggő gráf, $c : E \to \mathbb{R}$ az élein értelmezett súlyfüggvény, és $V = \{1, \ldots, n\}$. R. C. Prim módszere egy mohó algoritmus, ami a piros-kék módszer részletesen kidolgozott változatának tekinthető. Az $s = 1$ csúcsból indulva a kék szabály alkalmazásával lépésről lépésre bővítgetjük az s-et tartalmazó kék fát. A kék éleket az F halmazban tároljuk. Végül F egy minimális költségű feszítőfa éleit tartalmazza. A kék élek kiválasztásához hasznos lesz még egy U változó, ami az aktuális kék fa csúcsait tartalmazza. Kezdetben U csak az 1-es csúcsból áll. A kék szabály alkalmazása úgy történik, hogy kiválasztjuk azon élek közül a legkisebb költségűt, melyek egyik végpontja U-beli, a másik pedig $V \setminus U$-beli. Vagyis mondhatjuk, hogy az U-hoz legközelebbi „külső" csúcsot jelöljük ki. Ezt a csúcsot U-ba, a kiválasztott élet pedig a fokozatosan növekvő kék fába tesszük.

procedure Prim (G: gráf; **var** F: élek halmaza);
  var
    U: csúcsok halmaza;
    u, v: csúcsok;
  begin
    $F := \emptyset$;
    $U := \{1\}$;
    while $U \neq V$ **do begin**
(*) legyen (u, v) egy legkisebb súlyú olyan él,
      melyre $u \in U$ és $v \in V \setminus U$;
      $F := F \cup \{(u, v)\}$;
      $U := U \cup \{v\}$
    end
  end

A Prim-algoritmus helyessége, vagyis hogy végül F a G gráf egy minimális költségű feszítőfájának éleit tartalmazza, következik a piros-kék algoritmus ha-

sonló tulajdonságából. A következő rajzon egy súlyozott élű irányítatlan gráf látható. Az egyetlen minimális költségű feszítőfájának az éleit megvastagítottuk.



Ha Prim módszerével a v_1 csúcstól indulunk, akkor a pontok a $v_1, v_2, v_5, v_4, v_3, v_6$ sorrendben kerülnek be az U halmazba. Ennek megfelelően F-be először a (v_1, v_2) élet vesszük, amit (v_2, v_5), (v_5, v_4), (v_2, v_3) és végül (v_5, v_6) követ.

Ezután a Prim-algoritmus megvalósításának finomabb részleteivel foglalkozunk.

Naiv implementáció:

Tegyük fel, hogy a gráf az élsúlyokat tartalmazó C adjacencia-mátrixával[5] adott. A $(*)$ sor megvalósításához minden lépésben ki kell tudjuk választani az épp aktuális U és $V \setminus U$ halmazok közt futó legkisebb súlyú élek egyikét. Az összes él megvizsgálása nyilván nagyon sok időt venne igénybe. Ezért minden $V \setminus U$-beli csúcshoz tároljuk, hogy milyen messze van az U halmaztól, vagyis a belőle az U halmazba futó élek közül egy minimálisnak a súlyát. Tartsuk még nyilván ezen élek U-beli végpontját is. Erre szolgál a következő két tömb:

$$\text{KÖZEL}[i] = \begin{cases} * & \text{ha } i \in U \\ \text{egy az } i\text{-hez legközelebbi } U\text{-beli csúcs} & \text{ha } i \in V \setminus U \end{cases}$$

$$\text{MINSÚLY}[i] = \begin{cases} * & \text{ha } i \in U \\ C[i, j] & \text{ha KÖZEL}[i] = j \neq * \end{cases}$$

A következő kék él az $(i, \text{KÖZEL}[i])$ élek közül kerül majd ki. Ezért ezeket az éleket *kékes* éleknek nevezzük. Kezdetben $U = \{1\}$. Ennek megfelelően a kezdőértékek:

[5] Mint ahogy a Dijkstra-algoritmusnál, itt is feltesszük, hogy $C[i, j] = \infty$, ha (i, j) nem éle G-nek.

$$\text{KÖZEL}[i] := \begin{cases} * & \text{ha } i = 1 \\ 1 & \text{ha } i \neq 1 \end{cases}$$

$$\text{MINSÚLY}[i] := \begin{cases} * & \text{ha } i = 1 \\ C[i,1] & \text{ha } i \neq 1 \end{cases}$$

Ezek után a $(*)$ sor teendőit így valósíthatjuk meg:

- *A következő kék él kiválasztása:* megkeressük a MINSÚLY[ ] tömb minimumát, vagyis a legrövidebb kékes él hosszát. Mondjuk ez legyen k-nál. A minimumkeresés költsége: $O(n)$ elemi lépés. Nyilvánvaló, hogy a $V \setminus U$-beli csúcsok közül k egyike az U-hoz legközelebbieknek. Tehát az (u, v) él lehet a (KÖZEL$[k], k)$ él; ezt fogjuk F-be tenni, k-t pedig U-ba. Ennek megfelelően a két tömb k-hoz tartozó értékét $*$-ra kell állítanunk: MINSÚLY$[k]$:= KÖZEL$[k] := *$. E teendők időigénye $O(1)$).

- *A két tömb felfrissítése:* (Erre azért van szükség, mert az U halmaz bővült k-val. Így elképzelhető, hogy valamely $V \setminus U$-beli csúcshoz tartozó tömbértékeket meg kell változtatni, mert a csúcs közelebb van k-hoz, mint bármely régi U-beli csúcshoz.) Ehhez a $C[k, i]$ és a MINSÚLY$[i]$ értékeket $(i \in V \setminus U)$ kell összevetni. Ez $O(n)$ költséggel megtehető a következő kódrészlet végrehajtásával (minden $i \in V \setminus U$ pontra):

```
if KÖZEL[i] ≠ * and C[k,i] < MINSÚLY[i] then begin
   KÖZEL[i] := k;
   MINSÚLY[i] := C[k,i]
end
```

Mindent egybevetve, a **while**-ciklus belsejének végrehajtása $O(n)$ időt tesz ki. Mivel a ciklus n-szer fut le, az algoritmus összidőigénye naiv implementáció esetén $O(n^2)$.

Kupacos-éllistás implementáció:

Ha a gráfunk viszonylag ritka, vagyis az élek száma jóval kevesebb, mint n^2, akkor érdemes éllistát használni a gráf tárolására, és az algoritmust a következőképp megvalósítani.

A **while**-ciklus magjának minden egyes lefutásakor azon élek közül kell a minimális súlyút kiválasztanunk, melyek egyik végpontja U-ban, a másik $V \setminus U$-ban

van. Az U halmaz bővítése után lesznek újabb ilyen tulajdonságú élek, ezeket hozzá kell venni a többihez (nem törődünk az esetleg ottmaradó most már U-n belüli élekkel). Vagyis MINTÖR és BESZÚR műveleteket kell hatékonyan implementálnunk, amihez a kupac adatszerkezet lesz segítségünkre. Tehát építsünk és tartsunk fenn kupacot az U és $V \setminus U$ közti élekből (rendezés élsúly szerint). Ez kezdetben csak az 1-es csúcsból kiinduló éleket tartalmazza. Ezek után a ciklus belsejében az (u, v) él kiválasztása néhány MINTÖR művelettel megvalósítható. Azért néhánnyal és nem feltétlenül eggyel, mert nem foglalkozunk külön azoknak az éleknek a kupacból való törlésével, melyeknek egy csúcs hozzávétele után mindkét végpontja U-beli lesz. Így a MINTÖR eredményeképp kaphatunk ilyen élet is. Ilyenkor újabb MINTÖR-rel próbálkozunk, míg nem találunk U-ból kimenő élet. A következő kék él meghatározása után BESZÚR műveletekkel hozzávesszük a kupachoz a kiválasztott v csúcsból a $V \setminus U$ halmazba menő éleket.

A kupac mérete sosem haladja meg e-t. Ezért a kezdeti kupacépítés legfeljebb $O(e)$, az egyes műveletek végrehajtása pedig $O(\log e)$ időt vesz igénybe. Összesen kevesebb, mint e darab BESZÚR és legfeljebb e darab MINTÖR műveletet végzünk. Tehát az algoritmus költsége itt $O(e \log e)$. Mint ahogy azt megjegyeztük, a kupacos-éllistás implementációt kevés élet tartalmazó gráfoknál érdemes használni.

Johnson módszere:

A következő – D. B. Johnsontól származó – elképesztően erőteljes implementáció merít a két előző megközelítés gondolataiból. Nyilvántartjuk egyrészt a kékes éleket, mint ahogy azt a naiv implementációnál tettük; másrészt kupacot használunk a minimum kiválasztására. A $j \in V \setminus U$ csúcshoz tároljuk egy belőle kiinduló kékes él adatait. Erre a célra egy $\boxed{j \mid i \mid c(i,j)}$ szerkezetű cellát állítunk össze. Itt i egy a j-hez legközelebbi U-beli csúcs, a $c(i,j)$ pedig az (i,j) kékes él súlya. A cellákat a c-érték mint kulcs szerint d-kupacban tartjuk. Először csak az 1-es csúcsból kimenő élek végpontjainak cellái lesznek a kupacban. Kezdetben tehát a kupac a $\boxed{j \mid 1 \mid c(1,j)}$ alakú cellákból áll, ahol $(1,j)$ éle G-nek.

A kupac létrehozásának (kupacépítés) a költsége $O(n)$. A Prim-algoritmus $(*)$ sorában a kékre festendő (u, v) él meghatározása egy MINTÖR-rel megtehető. Ezután – a naiv változathoz hasonlóan – gondoskodnunk kell a kékes élek halmazának a frissítéséről. Ehhez meg kell vizsgálnunk a v éllistáján szereplő (v, w) éleket. (Itt v az U halmazba éppen bekerült csúcs, ezért a belőle kimenő élek szóba jöhetnek mint kékes élek.) Ha most $w \in U$, akkor nincs további teendőnk ezzel az éllel. Ha $w \notin U$, akkor két eset lehetséges. Előfordulhat egyfelől, hogy w-ből eddig nem ment él U-ba. Ekkor (v, w) nyilván egy kékes él lesz. Ennek megfelelően egy BESZÚR művelettel a $\boxed{w \mid v \mid c(v,w)}$ cellát a kupacba tesszük. Másfelől

az is lehetséges, hogy w-ből már megy ki kékes él, de a (v, w) él ennél rövidebb. Ezt onnan látjuk, hogy a w cellája a kupacban van, és a benne levő súly nagyobb, mint $c(v, w)$. Ekkor a cellát átírjuk, aminek az eredménye $\boxed{w \mid v \mid c(v,w)}$ lesz. A módosított cellában a kulcs értéke csökkent, tehát egy FOGYASZT segítségével illeszthetjük vissza a kupacba. Ezek a módosítások a G éleihez köthetők. Egy él kapcsán legfeljebb egyszer módosítunk; mégpedig akkor, amikor egyik végpontja éppen bekerül U-ba, és a másik nincs U-ban. A fogyasztások száma tehát legfeljebb e. A beszúrások száma pedig legfeljebb $n-1$, hiszen egy csúcs cellája legfeljebb egyszer kerül a kupacba. Figyelembe véve, hogy a kupacban legfeljebb $n-1$ cella van, a műveletek költsége összesen $O(n + nd \log_d n + n \log_d n + e \log_d n)$. Az első tag a kupacépítés, a második az $n-1$ darab MINTÖR, a harmadik a beszúrások, az utolsó pedig a fogyasztások lépésszáma. A formula hasonlít a Dijkstra-módszer d-kupacos változatának elemzésénél kapott kifejezéshez. Hasonló a következtetés is, amit levonhatunk belőle. Tegyük fel, hogy G nem túl ritka, mondjuk $n^{1,5} \leq e$. Legyen ekkor $d = \lceil e/n \rceil$. Nyilvánvaló, hogy $d \geq \sqrt{n}$, amiből $\log_d n \leq 2$. Ezt figyelembe véve

$$\begin{aligned} O(n + nd \log_d n + n \log_d n + e \log_d n) &= O(n + nd + n + e) \\ &= O(n + n \cdot e/n + n + e) = O(e). \end{aligned}$$

Viszonylag sűrű gráfok esetén tehát a kupacműveletek összköltségére lineáris korlátot kaptunk.

Maradt még egy elvarratlan szál. A w csúccsal a kezünkben gyorsan el kell tudnunk dönteni, hogy az már/még a kupacban van-e; ha igen, akkor meg kell találni a celláját. Ezt egy MERRE$[2 : n]$ tömb segítségével egyszerűen megtehetjük. Legyen MERRE$[w] = \infty$, ha w cellája még nincs a kupacban, és legyen $*$, ha w már U-ba került. Ha pedig a w cellája a kupacban van, akkor MERRE$[w]$ legyen egy mutató erre a cellára. A tömb segítségével a w-vel kapcsolatos döntés konstans időben meghozható. A tömb karbantartásához szükséges munka $O(e)$. Ez azért igaz, mert egy kupacművelet után állandó mennyiségű munkával aktualizálható a tömb; a kupacműveletek száma pedig $O(e)$. A Johnson-implementáció tehát *lineáris költségű*, ha G viszonylag sűrű ($n^{1,5} \leq e$) gráf.

6.6.2. Kruskal módszere

Legyen továbbra is $G = (V, E)$ egy összefüggő gráf, $c : E \to \mathbb{R}$ az élein értelmezett súlyfüggvény, és $V = \{1, \ldots, n\}$. Kruskal algoritmusa, mely G egy minimális költségű feszítőfáját konstruálja meg, szintén egyfajta mohó stratégiát követ. Míg Prim módszerénél egyetlen kék fát növesztettünk, itt több helyen kezdjük el építgetni, és kapcsoljuk fokozatosan össze a még diszjunkt darabokat. A piros éleket figyelmen kívül hagyva elmondhatjuk, hogy a kék élek F halmazát bővítjük a

legkisebb súlyú olyan éllel, amely az eddig kiválasztott élekkel együtt nem alkot kört. Nyilván egy ilyen él két diszjunkt kék fát köt össze. A kiinduló helyzetben n egypontú fánk van. Minden lépésben a legkisebb súlyú, kört nem okozó él hozzávételével a kék komponensek száma eggyel csökken. Így $n-1$ él kiválasztása után egyetlen kék fa lesz, nevezetesen G egy minimális költségű feszítőfája. Nagy vonalakban ez így írható le (H kezdetben a gráf összes éleinek halmaza):

procedure Kruskal (G: gráf; **var** F, H: élek halmaza);
begin
  $F := \emptyset$; $H := E$;
  while $H \neq \emptyset$ **do begin**
    Töröljük a H minimális súlyú (v, w) élét.
    Ha $F \cup \{(v, w)\}$-ben nincs kör
    (azaz a v, w pontok különböző kék fákban vannak), akkor
      $F := F \cup \{(v, w)\}$
  end
end

Ha tehát a (v, w) él kört eredményez, akkor eldobjuk (piros él), ha pedig nem, akkor bevesszük a feszítőfa (kék) élei közé. Mint korábban láttuk, a Kruskal-módszer is a piros-kék algoritmus egy változata. Érvényes tehát a következő:

Állítás: *A Kruskal-algoritmus eredményeként végül F a G gráf egy minimális költségű feszítőfájának éleit tartalmazza.* □

Az előző példa gráfjára alkalmazva a Kruskal-algoritmus először a (v_2, v_5) és a (v_4, v_5) éleket választja valamilyen sorrendben; utána a minimális feszítőfa 2 súlyú élei jönnek és végül a (v_5, v_6) él.

Ezután itt is a hatékony implementáció részleteinek kimunkálásával foglalkozunk. Feltesszük, hogy G éllistával adott. A minimális élek kiválasztására az egyik lehetséges megoldás, hogy a G éleiből súly szerint kupacot építünk (költség $O(e)$). Ez esetben a minimális súlyú él kiválasztása, és a kupac-tulajdonság helyreállítása $O(\log e)$ időbe kerül. Ha tehát H-t kupacként valósítjuk meg, akkor a vele kapcsolatos összköltség $O(e \log e)$ lesz (kupacépítés és legfeljebb e MINTÖR). Ugyanebben a nagyságrendben maradunk, ha az éleket előzetesen rendezzük súly szerint, és az eredményül kapott rendezett lista lesz a H.

Már csak egy fontos kérdés maradt hátra: hatékony megoldás kellene annak az eldöntésére, hogy a kiválasztott (v, w) él az F eddigi éleivel kört alkot-e. E döntés eredményétől függően lesz az él piros vagy kék. Tartsuk nyilván az aktuálisan egy kék fába tartozó csúcsokat mint halmazokat. Kétféle műveletre lesz szükségünk.

Amikor a két különböző fát összekötő élet hozzávesszük F-hez (vagyis kékre színezzük), akkor a megfelelő két halmaz unióját kell képeznünk, hiszen ezekből így egyetlen összefüggő komponens lesz. Ahhoz pedig, hogy eldöntsük egy élről, hogy két különböző kék fát köt-e össze, az egyes végpontokról meg kell tudnunk mondani, *hogy melyik halmazban vannak. Ezen műveletek hatékony megvalósítását* szolgálja a következő adatszerkezet.

6.6.3. Az UNIÓ-HOLVAN adatszerkezet

Legyen adott egy véges S halmaz. Ennek egy felosztását – szakszóval: partícióját – szeretnénk tárolni. Emlékeztetőül: a nem üres $U_1, \ldots, U_k \subseteq S$ halmazok az S egy partícióját alkotják, ha $\cup_{i=1}^{k} U_i = S$ és $i \neq j$ esetén $U_i \cap U_j = \emptyset$. Az U_i halmazok tehát hézagtalanul és egyrétűen lefedik S-et. Két műveletünk van. Egy $x \in S$ esetén meg kell tudnunk mondani, hogy az x melyik U_j részhalmazba esik. A másik művelet a partíció két osztályának az egyesítése. Valamivel formálisabban:

> Adott egy n elemű S halmaz, és ennek bizonyos $U_1, \ldots, U_m$ részhalmazai, melyekre $U_i \cap U_j = \emptyset$ $(i \neq j)$ és $U_1 \cup \ldots \cup U_m = S$ (vagyis az U_j részhalmazok S egy partícióját adják).
>
> *Műveletek:*
>
> UNIÓ$(U_i, U_j) = (\{U_1, \ldots, U_m\} \cup \{U_i \cup U_j\}) \setminus \{U_i, U_j\}$ (az U_i, U_j halmazokat $U_i \cup U_j$ helyettesíti).
>
> HOLVAN(v) eredménye (itt $v \in S$) annak az U_i halmaznak a neve, amelynek v eleme.

Egy UNIÓ hatására a felosztás durvább lesz, minthogy eggyel csökken a komponensek száma. Az UNIÓ-HOLVAN adatszerkezet nevezetes alkalmazásainál több UNIÓ és HOLVAN műveletből álló lépéssor végrehajtására van szükség *abból a kezdőhelyzetből indulva, amelyben mindegyik U_j egyelemű.* Mi is élünk ezzel a feltevéssel.

Kruskal algoritmusában tényleg valami ilyesmire van szükség: legyen ugyanis $S = V(G)$, az U_j halmazok pedig az F által meghatározott kék gráf fáinak a csúcshalmazai. Az éppen vizsgált (v, w) él az F-hez adva pontosan akkor nem eredményez kört, ha a végpontok különböző komponensben vannak, azaz HOLVAN$(v) \neq$ HOLVAN(w). A körmentesség tehát két HOLVAN kérdéssel ellenőrizhető. Miután a (v, w) élet F-be tettük, egyesítenünk kell a v-t és a w-t tartalmazó kék fákat. Ezt egy UNIÓ művelettel érhetjük el. Az induló feltétel is teljesül, ugyanis kezdetben n darab egypontú fánk van.

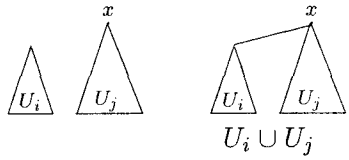
Célunk tehát az UNIÓ és a HOLVAN műveletek hatékony implementálása. A legegyszerűbb út, ha felveszünk egy n hosszú tömböt, és S minden eleméhez

tároljuk, hogy aktuálisan melyik részhalmazban van. Ekkor egy HOLVAN végrehajtása $O(1)$, egy UNIÓ pedig n-nel arányos időt vesz igénybe. Az előbbinél a tömb egy elemét kell lekérdeznünk, az utóbbinál pedig végig kell mennünk a tömbön, és a megfelelő két halmaznevet azonosra változtatnunk. Ennél azonban létezik lényegesen hatékonyabb megoldás is.

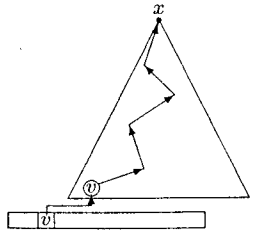
Implementáció fákkal

Egy U_j részhalmaznak egy gyökeres, felfelé irányított fa felel meg. U_j elemeit a fa csúcsaiban tároljuk, egy szülőmutatóval együtt (ami egy nullértéket tartalmaz a gyökérnél). Egy részhalmaz *neve* legyen az őt ábrázoló fa gyökere. A gyökérben nyilvántartjuk még a fa méretét is. Van továbbá egy S elemeivel indexelt tömbünk. Ennek a v indexű ($v \in S$) eleme egy mutató a v fabeli előfordulására. A műveletek megvalósítása:

- **UNIÓ:** $U_i \cup U_j$ fáját a következőképpen készítjük el:
  Tegyük fel, hogy $|U_i| \leq |U_j|$. Ekkor az U_j fa x gyökeréhez gyermekként hozzákapcsoljuk U_i gyökerét.



- **HOLVAN:** A $v \in S$ elemet tartalmazó részhalmaz nevét, azaz a megfelelő fa gyökerét a szülőkhöz menő mutatók végigkövetésével találhatjuk meg.



Tegyük fel, hogy az UNIÓ hívásakor az U_i és U_j halmazok a gyökerükkel adottak (ez a feltevés valós a Kruskal-módszernél). Az UNIÓ költsége ekkor nyilván $O(1)$, hiszen a gyökereknél adminisztráljuk az egyes fák méretét, és csak a kisebbik fa gyökerének szülőmutatóját kell átállítanunk. Vegyük észre, hogy amikor egy v csúcs új gyökér alá kerül, akkor egy szinttel lesz távolabb a gyökértől, míg az új fájának a mérete legalább az eredeti duplájára változik. Ezért egy csúcs legfeljebb $\log_2 n$-szer kerülhet új gyökér alá, így a szintszáma[6] legfeljebb $\log_2 n$ lehet. Ebből adódik, hogy a HOLVAN költsége $O(\log n)$. Ha nem vigyáztunk volna, hogy mindig a kisebb fát kapcsoljuk a nagyobbhoz, akkor a fa mélységére nem tudnánk ilyen korlátot adni; a HOLVAN költsége akár n-nel arányos is lehetne. Vegyük észre azt is, hogy ebben az elemzésben használtuk a kezdeti partícióval kapcsolatos feltételezést.

Kruskal algoritmusában egy él mindkét végpontjáról egyszer kérdezzük le, hogy melyik komponensben van. Összesen $n - 1$ élet választunk be a feszítőfába. Ez tehát $2e$ HOLVAN, és $n - 1$ UNIÓ műveletet jelent. Ezek időigénye $O(e \log n + n) = O(e \log n)$, vagy ami ugyanaz: $O(e \log e)$. Ebbe belefér a kiindulási helyzetet jelentő n egypontú fa kialakításának az $O(n)$ költsége is. Ugyancsak $O(e \log e)$ az összköltsége a minimális súlyú élek kiválasztásával kapcsolatos munkáknak, akár kupacot használunk, akár előzetesen rendezzük az éleket.

Következmény: *A Kruskal-algoritmus költsége* $O(e \log e)$. □

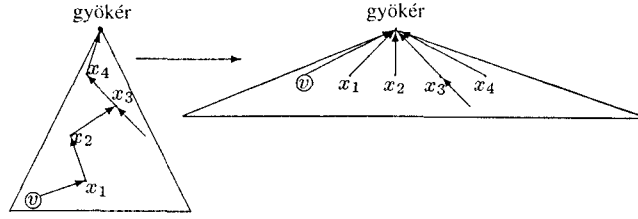
Az UNIÓ-HOLVAN adatszerkezetet eddig jórészt afféle „individualista" szemlélettel néztük, amennyiben az egyes műveletek egyetlen végrehajtásának az idejére adódó korlátokkal foglalkoztunk. Valójában az érdekes alkalmazásoknál nem egy, hanem egy egész sor UNIÓ-t és HOLVAN-t kell végrehajtanunk. Ilyenkor érdemes lehet a hosszabb műveletsorok idejét javítani. Az elgondolás hasonlít az S-fáknál és az önszervező tábláknál látottakhoz: amikor egy HOLVAN kérdést megválaszolunk, akkor némi további munkával igazítunk az éppen elért fa alakján azt remélve, hogy ez a befektetés később megtérül. Ebben nem is csalatkozunk.

A HOLVAN gyorsítása: útösszenyomás

Az UNIÓ és a HOLVAN műveletek közül az utóbbi a költséges; ennél érdemes takarékoskodni az idővel. A HOLVAN(v) időigénye a v-től a fája gyökeréig vivő út hosszával arányos. Ha tehát gondoskodunk arról, hogy az elemek viszonylag közel legyenek a gyökérhez, akkor jobb időket kaphatunk. Egy ilyen ötlet az *útösszenyomás*. Ennek lényege, hogy HOLVAN(v) meghívása esetén v fájában a v-től a gyökérhez vezető út minden pontját közvetlenül a gyökér alá csatlakoztatjuk át.

[6] Egy csúcs *szintszáma* a tőle a gyökérig vezető úton levő élek száma.

Ezt legegyszerűbben úgy tehetjük meg, hogy még egyszer végigmegyünk ezen az úton, és átállítjuk a szülőmutatókat. Így egy kis pluszmunkával javítunk a fa alakján. Az x_i csúcsok és a leszármazottaik közelebb kerülnek a gyökérhez.



Az útösszenyomást alkalmazó implementáció időigényének a becslése meglehetősen bonyolult; csak az eredményt ismertetjük.

Tétel: *Legyen $|S| = n$, és tegyük fel, hogy kezdetben mindegyik U_j egyelemű. Ha egy olyan utasítássorozatot hajtunk végre (útösszenyomással), melyben $n-1$ UNIÓ és $m \geq n-1$ HOLVAN szerepel, akkor ennek az időigénye $O(m\alpha(m))$.*

A korlátban szereplő $\alpha : \mathbb{Z}^+ \to \mathbb{Z}^+$ függvény az *inverz Ackermann-függvény*. Az α cirkalmas definíciójától eltekintünk. Csak annyit jegyzünk meg róla, hogy $\alpha(m)$ a végtelenhez tart, ha $m \to \infty$, ezt azonban a képzeletet jócskán próbára tevő lassúsággal teszi. Lassabban nő például, mint a logaritmus k-szori önmagába helyettesítésével adódó $\log\log\cdots\log m$ függvény ($k \in \mathbb{Z}^+$ tetszőleges). Igaz továbbá, hogy $\alpha(m) \leq 4$, ha $m < 2^{65536}$. A praktikus méretű feladatoknál tehát $\alpha(m)$ állandónak (≤ 4) tekinthető.

A Kruskal-algoritmussal kapcsolatos teendők két csoportba sorolhatók. Az egyik csoportot a minimális élek választásai jelentik, a másikat pedig az UNIÓ-HOLVAN kezelése. Az első csoport összköltségére $O(e \log e)$ korlátot kaptunk. Gyakran előfordul azonban, hogy ezek a rendezéssel kapcsolatos teendők $O(e)$ lépésben is megoldhatók. Ilyen helyzet például, amikor az élsúlyok kis egészek, és ezért az élek a radix-módszerrel lineáris időben rendezhetők. Az is megeshet, hogy a bemenetben már eleve rendezetten kapjuk az éleket. Ezekben az esetekben az általánosnál jobb korlátot nyerünk.

Alkalmazás: *Ha az élek rendezésével kapcsolatos teendők $O(e)$ időben megoldhatók, akkor a Kruskal-algoritmus $O(e\alpha(e))$ időben megvalósítható.*

Bizonyítás: A második csoportba tartozó teendőkre alkalmazható a tétel: $n-1$ UNIÓ-t és legfeljebb $2e$ HOLVAN-t hajtunk végre. A feltétel szerint az összköltség becslésében a domináns tag az e műveletekre vonatkozó $O(e\alpha(e))$ lesz. □

6.6.4. Megjegyzések

1. Az általunk szemügyre vett algoritmusok (Borůvka, Prim és Kruskal módszerei) az élsúlyokkal csak összehasonlításokat végeznek. Nem vizsgálják például a bitjeiket, nem adják össze őket, stb. Izgalmas nyitott kérdés, hogy van-e az ilyen értelemben összehasonlítás alapú algoritmusok között lineáris költségű. Közel lineáris módszerek léteznek. A Borůvka-módszernek például van $O(e \log \log n)$ költségű megvalósítása (A. C. Yao, 1975). A lineáris időkorlát elérhető véletlen választásokkal (D. R. Karger, P. N. Klein, R. E. Tarjan, 1994). Módszerük várhatóan $O(e)$ lépésben talál egy minimális költségű feszítőfát.

M. Fredman és D. E. Willard (1990) módszere lineáris idejű ugyan, de kulcsmanipulációs jellegű, amennyiben az élsúlyok bitjeivel is végez számításokat.

Érdekes kapcsolódó eredmény, hogy ha adott G és annak egy F élhalmaza, akkor lineáris időben eldönthető, hogy F minimális költségű feszítőfa-e (Komlós János 1985, V. King 1993).

2. A minimális feszítőfák problémájának több figyelemre méltó rokona és változata van. Ezekből most két feladatban mutatunk mintát. Az első a probléma *online* változatáról szól. Tegyük fel, hogy a G éleit egyesével kapjuk valamilyen sorrendben, és szeretnénk gyorsan meghatározni az eddig látott élekből álló gráf egy minimális költségű feszítő erdejét (amelynek az összefüggő komponensei ugyanazok, mint az eddigi élekből álló gráfnak). Évégből tegyük fel, hogy az eddigi élek közül már kiválasztottunk egy kék feszítő erdőt, és jön a következő f él. A recept egyszerű: színezzük az f élet kékre; ha emiatt kialakul egy kék kör, akkor abból töröljünk egy maximális súlyú élet.

Feladat: Mutassuk meg, hogy az előző algoritmus helyes; ha az üres gráfból indulva sorra alkalmazzuk az $f_1, f_2, \ldots, f_i$ élekre, akkor a kék élek halmaza az $f_1, f_2, \ldots, f_i$ élekből álló G_i gráf egy olyan minimális költségű feszítő erdeje lesz, amelynek ugyanannyi összefüggő komponense van, mint G_i-nek. Speciális esetként: ha G összefüggő, akkor végül – amikor már G minden élét láttuk – egy minimális feszítőfát kapunk. (Használjunk i szerinti indukciót.)

Egy feszítőfa költségét úgy definiáltuk, hogy az a benne szereplő élek súlyainak az összege. Bizonyos esetekben másféle költségszámítás is értelmes lehet. Például egy fa költségeként értelmezhetjük a legnagyobb súlyú élének a súlyát[7].

Feladat: Mutassuk meg, hogy a piros-kék algoritmus minimális feszítőfát ad akkor is, ha így értelmezzük egy fa költségét.

[7]Ez a költségfogalom merül fel, amikor az élsúlyok a csúcsok közötti kapcsolatok valamiféle megbízhatatlanságát mérik. A nagyobb súlyú él tehát kevésbé megbízható. A minimális feszítőfa ilyenkor egy olyan összeköttetés-rendszert jelent, amiben a leggyengébb láncszem a lehető legmegbízhatóbb.

6.7. Maximális párosítás páros gráfokban

A lovagok és udvarhölgyek problémája egyike volt bevezető példáinknak (az első fejezetben). Itt ennek a kérdésnek az általánosításával, a *párosítási feladattal* foglalkozunk. A párosítási feladatnak több fontos változata van. Ezek közös magja, hogy egy gráfban minél nagyobb olyan élrendszert (szakszóval: *párosítást*) keresünk, amelyben semelyik két élnek sincs közös végpontja. A *nagy* az előző mondatban utalhat a kiválasztott élek számára vagy – ezen túlmenően – a kiválasztott élek súlyainak összegére.

Párosítási feladatként fogalmazható meg egy sereg erőforrás-kiosztási probléma. Például tegyük fel, hogy adottak gépek és munkák véges halmazai: G és M. Egy gép egy időszakban csak egy munkával tud foglalkozni. Ismert továbbá, hogy egy gép mely munkák elvégzésére alkalmas. Szeretnénk egy időszakban minél jobban kihasználni a gépeket: minél több géphez akarunk munkát rendelni úgy, hogy ne sértsük meg az előző feltételeket. A problémának megfelelő gráf ponthalmaza $G \cup M$. A $g \in G$ gépet akkor köti él az $m \in M$ munkához, ha g képes az m elvégzésére. A megoldások a gráf maximális élszámú párosításai.

A következőkben a párosítási feladat legegyszerűbb esetével foglalkozunk, amikor is a szóban forgó gráf páros, és nincsenek élsúlyok (vagyis minden élsúly 1). Lássuk először az idevágó fogalmakat:

Definíció (páros gráf): *A $G = (V, E)$ gráfot* párosnak *nevezzük, ha V csúcshalmaza felosztható két diszjunkt részre – V_1-re és V_2-re – úgy, hogy minden él ezen két halmaz között fut, vagyis $(x, y) \in E$ esetén $x \in V_1$ és $y \in V_2$ vagy fordítva.*

Ha egy $G = (V, E)$ páros gráfot úgy adunk meg, hogy $G = (V_1, V_2; E)$, akkor V_1 és V_2 a fenti tulajdonságú partíciója a V-nek.

Definíció (párosítás): *Legyen $G = (V, E)$ egy tetszőleges gráf. Az E élhalmaz $E' \subseteq E$ részhalmaza G egy* párosítása, *ha a $G' = (V, E')$ gráfban minden pont foka legfeljebb egy.*

A párosításban szereplő éleket, illetve a párosítás által fedett pontokat (vagyis melyek foka G'-ben egy) *párosítottnak* fogjuk hívni.

Definíció (maximális párosítás): *A G gráf egy E' párosítása* maximális, *ha G minden E'' párosítására $|E''| \leq |E'|$.*

Ezek után megfogalmazhatjuk pontosan a feladatot:

A probléma: Adott egy $G = (V_1, V_2; E)$ páros gráf. Határozzuk meg G egy maximális párosítását.

6.7.1. A magyar módszer

Az itt bemutatásra kerülő algoritmus első változatát König Dénes[8] közölte 1916-ban. A módszert König Dénes és a súlyozott feladatot megoldó Egerváry Jenő tiszteletére nevezik világszerte magyar módszernek. A következő két fogalom elvezet bennünket az algoritmus lényegéhez.

Definíció (alternáló út): *Legyen G egy tetszőleges gráf, és E' a G egy párosítása. Egy G-beli utat E'*-alternáló útnak *hívunk, ha felváltva tartalmaz párosított és nem párosított éleket.*

Ha a G gráf egy adott E' párosításához találunk olyan P alternáló utat, amely két különböző párosítatlan csúcsot köt össze, akkor ez az út párosítatlan éllel kezdődik és fejeződik be. P-nek tehát eggyel kevesebb párosításbeli éle van, mint nem párosításbeli. Ezért az E'-nél jobb párosítást kapunk, ha elhagyjuk belőle a P út eddig párosított éleit, de hozzávesszük az eredetileg párosítatlanokat. Az ilyen tulajdonságú utat, melynek segítségével javítani tudunk egy meglévő párosításon, javító útnak nevezzük. Formálisan:

Definíció (javító út): *Legyen E' a $G = (V, E)$ gráf egy párosítása. Ekkor egy olyan E'-alternáló út, melynek mindkét végpontja párosítatlan, E'*-re nézve javító út, *vagy röviden E'*-javító út.

Arra jutottunk, hogy ha a G gráfban adott egy E' párosítás és egy P út, mely E'-javító út, akkor az $E'' = E' \oplus P$ egy az E'-nél nagyobb párosítás. Itt $\oplus$ a halmazok körében értelmezett szimmetrikus differencia műveletét jelöli (a P utat ilyenkor élek halmazaként fogjuk fel). Két halmaz szimmetrikus differenciája az a halmaz, melynek elemei a két halmaz valamelyikében szerepelnek, de mindkettőben nem:

Jelölés (szimmetrikus differencia): *Legyenek H_1, H_2 tetszőleges halmazok.*

$$H_1 \oplus H_2 = (H_1 \setminus H_2) \cup (H_2 \setminus H_1).$$

[8]König Dénes (1884-1944) a Budapesti Műszaki Egyetem tanára volt. A *Theorie der endlichen und unendlichen Graphen* című, 1936-ban Lipcsében megjelent munkája az első könyv, amely rendszerbe foglalva tárgyal gráfokkal kapcsolatos ismereteket. A gráfelmélet ettől kezdve tekinthető önálló tudományos témának. A mű fontosságát jelzi, hogy 1950-ben reprint kiadása született az Egyesült Államokban, 1990-ben pedig angolra fordították. Tudomásunk szerint ő hirdetett meg először a történelemben gráfelméleti tárgyú egyetemi előadást; ez az 1927/28-as tanévben történt.

A tudósportré mosolygós vonásaként említhetjük a *Mathematikai mulatságok* címmel 1902-ben és 1905-ben megjelent két könyvecskéjét. Ezekben a gyűjteményekben érdekes matematikai rejtvényeket adott közre. A feladatok szépsége, a magyarázatok áttetsző tisztasága teszi őket nagyszerű olvasmánnyá. A két füzetet 1992-ben ismét kiadták (Typotex Kft.).

Ezek alapján az algoritmus úgy néz ki, hogy kiindulunk valami könnyen szerzett párosításból (ez lehet üres is), és minden lépésben az aktuális párosításra nézve keresünk egy javító utat. A javító út mentén egy nagyobb párosítást kapunk. De mi van akkor, ha már egy párosításhoz nem találunk javító utat? A következő tétel pont azt mondja ki, hogy ekkor nyugodtan megállhatunk, mert az aktuális párosításunk már maximális.

Tétel: *Legyen $G = (V, E)$ egy tetszőleges gráf és E' egy párosítása. Ha E'-re nézve nincs javító út G-ben, akkor E' a G egy maximális párosítása.*

Bizonyítás: Legyen M a G egy tetszőleges maximális párosítása. Vizsgáljuk meg a $G' = (V, E' \oplus M)$ gráfot. Minden pontra mindkét párosításban legfeljebb egy él illeszkedhet, ezért G'-ben minden pont foka legfeljebb kettő. Ez azt jelenti, hogy G' komponensei utak és körök, amelyek felváltva tartalmaznak E'-beli és M-beli éleket. Így a körök csak páros hosszúak lehetnek, vagyis mindkét párosításból ugyanannyi él szerepel bennük. A feltétel szerint egyik út sem lehet E'-javító, M maximális volta miatt pedig egyik sem lehet M-javító. Ezért G'-ben minden út páros hosszúságú, és mindkét párosításból ugyanannyi élet tartalmaz. Ebből következik, hogy $|E'| = |M|$, vagyis az E' párosítás is maximális. □

Már csak az maradt hátra, hogy megmutassuk, miként lehet egy gráf egy adott párosításához javító utat találni. Vegyük észre, hogy az előző tétel nem csak páros gráfokról szólt. A javító út keresésénél azonban már lényegesen kihasználjuk G páros voltát. Egy $G = (V_1, V_2; E)$ páros gráfban meg tudjuk adni a csúcsok egy olyan halmazát, melyben a lehetséges javító utaknak pontosan az egyik végpontja szerepel. Például V_1 ilyen lesz, hiszen egy páros gráfban minden javító út – mivel páratlan hosszú – egy V_1 és egy V_2-beli pontot köt össze.

A javító út kereséséhez egy *alternáló erdőt* növesztünk. Ennek csúcsait szintekre osztva érdemes elképzelni. A nulladik szinten a V_1-beli párosítatlan csúcsok vannak. Ezután szintről szintre növesztgethetjük a lehetséges javító utakat úgy, hogy felváltva veszünk fel párosítatlan és párosított éleket. Az egyre bővülő gráf egy erdő lesz, melynek szintjein felváltva szerepelnek V_1-beli és V_2-beli pontok. Ha valamelyik páratlan szinten megjelenik egy még párosítatlan v csúcs, akkor javító utat találtunk. Az erdőben a v-től a nulladik szintre visszavivő út javító út lesz.

Javító út keresése alternáló erdő építésével

Adott a $G = (V_1, V_2; E)$ páros gráf, és egy E' párosítása. Egy az E'-hez tartozó alternáló erdőt a következő módon, szintről szintre haladva építhetünk fel:

0. szint:	V_1 azon pontjai, melyeket E' nem fed le, vagyis a párosítatlan pontok.
⋮	
$2k-1$. szint:	V_2 azon még fel nem vett pontjai, melyek egy párosítatlan, azaz egy $E \setminus E'$-beli éllel elérhetők egy $2k-2$. szintbeli pontból; ezen éllel együtt.
$2k$. szint:	V_1 azon még fel nem vett pontjai, melyek egy párosított, azaz egy E'-beli éllel elérhetők egy $2k-1$. szintbeli pontból; ezen éllel együtt.
⋮	

Egy csúcsot legfeljebb egyszer veszünk fel, mégpedig az első lehetséges szinten. A kialakuló gráfban nincs kör, hiszen egy csúcshoz az alacsonyabb sorszámú szintekről csak egy él illeszkedik. A fa növesztése könnyen megvalósítható a szélességi bejárás kis módosításával. Induláskor a nulladik szint pontjait tesszük a Q sorba. Ezután pedig a bejárás során a páros szintről induló élek közül csak a párosítatlanokkal foglalkozunk; a páratlan szintről indulók közül pedig csak az E'-beliekkel. Ha tehát G éllistával adott, akkor az alternáló erdőt $O(e)$ költséggel megkaphatjuk.

Állítás: *A $G = (V_1, V_2; E)$ páros gráfban akkor és csak akkor van az E' párosításra nézve javító út, ha az E'-hez tartozó alternáló erdőben valamelyik páratlan szinten megjelenik egy párosítatlan pont.*

Bizonyítás: Az elégségesség nyilvánvaló, ugyanis egy páratlan szinten levő párosítatlan pontot az erdőben a nulladik szinttel összekötő út egy javító út.

A fordított állításhoz először megmutatjuk, hogy a V_1 párosítatlan csúcsaiból (ezek vannak az erdőben a nulladik szinten) alternáló úton elérhető pontok mindegyikét beválasztjuk valamikor az alternáló erdőbe. Ennek igazolására tegyük fel, hogy $v_0, v_2, \ldots, v_k$ egy alternáló út, és $v_0 \in V_1$ egy párosítatlan csúcs; i szerinti indukcióval megmutatjuk, hogy v_i bekerül az erdőbe. Ez nyilván igaz, ha $i = 0$.

Az út v_{2j} csúcsa V_1-ben van és a (v_{2j}, v_{2j+1}) él párosítatlan. Tehát ha v_{2j}-t beválasztottuk, akkor v_{2j+1} is bekerül, ha előbb nem, akkor a v_{2j} utáni szintre. Hasonló okfejtést alkalmazhatunk a páros indexű csúcsokra: az út v_{2j-1} csúcsa V_2-ben van és $(v_{2j-1}, v_{2j}) \in E'$. Így v_{2j-1} után v_{2j} is sorra kerül, ha korábban ez még nem történt meg.

Tegyük fel mármost, hogy G-ben a v és w csúcsok egy javító út végpontjai. Ezek a pontok nyilván párosítatlanok, és egyikük – mondjuk v – a V_1-ben van. Ekkor pedig $w \in V_2$. A v szerepel az erdő nulladik szintjén, és w-t is be fogjuk

választani valamikor, hiszen v-ből alternáló úton elérhető. A w az erdőnek csak egy páratlan sorszámú szintjére kerülhet, mivel V_2-beli csúcsokat csak páratlan szintekre veszünk fel.

□

Amint azt már megállapítottuk, az alternáló erdő növesztésének költsége $O(e)$. Nyilvánvaló másfelől, hogy legfeljebb $n/2$-szer kell javító utat keresnünk. *Ezért a magyar módszer időigénye éllistával adott bemenet esetén* $O(ne)$. A témától búcsúzóban megjegyezzük, hogy vannak a magyar módszernél hatékonyabb (és persze sokkal kevésbé egyszerű) algoritmusok is. Ilyen például J. E. Hopcroft és R. M. Karp 1973-ból való módszere, ami $O(\sqrt{n}e)$ lépésben talál maximális párosítást páros gráfokban.

6.8. Maximális folyamok hálózatokban

Hempergő jóság vagy,
termést bőséggel adj...
RADNÓTI MIKLÓS: Himnusz a Nílushoz

Az emberiség történelme a kezdetektől fogva szorosan kötődik a nagy folyamokhoz. Valami hasonlót mondhatunk a kombinatorikus optimalizálás és a hálózati folyamok viszonyáról. A hálózati folyamok természetes és meglepően gazdag modellt kínálnak olyan helyezetek leírására, amelyek lényege, hogy egy rendszerben anyag áramlik bizonyos – forrásnak nevezett – keletkezési helyektől bizonyos – nyelőnek nevezett – felhasználási helyek felé. Az *anyag* itt ezerféle dolgot jelenthet; gondolhatunk csővezetéken áramló folyadékra, elektromos áramra, számítógépes hálózaton közvetített információra, utcákon mozgó járművekre stb.

A hálózat egyes csomópontjai között közvetlen összeköttetések vannak; ezeket az anyag szállítására alkalmas médiumoknak (csővezeték, kábel, utca, stb.) tekintjük. A legegyszerűbb modellben egy ilyen kapcsolatot az *irányával* és a *kapacitásával* jellemzünk. Az előbbi mondja meg, hogy a két csomópont között milyen irányú áramlás lehetséges, az utóbbi pedig az időegység alatt átvihető maximális mennyiséget jelenti. Például egy kommunikációs vonal esetén a kapacitás lehet a másodpercenként átvihető bitek száma. Az anyagáramlást, a *folyamot* úgy képzeljük el, hogy a forrásokban állandó sebességgel keletkezik az anyag, és ugyanekkora sebességgel enyészik el a nyelőkben. A többi – a forrásoktól és a nyelőktől különböző – csomóponton csak átáramlik; ami beérkezik, az távozik is. Ezeknél a csúcsoknál a folyamra teljesül az elektrodinamikából ismerős Kirchoff csomóponti törvény megfelelője. Nézzük mindezeket pontosabban, azzal az egyszerűsítő

feltevéssel, hogy a hálózatban csak egy forrás és egy nyelő van! A forrást és a nyelőt hagyományosan s (source) és t (target, terminal) jelöli.

Definíció (hálózat): *Adott egy $G = (V, E)$ irányított gráf és ennek két különböző pontja, s és t, melyeket* forrás*nak, illetve* nyelő*nek hívunk. Adott még egy az éleken értelmezett $c : E \to \mathbb{R}^+$ pozitív értékű kapacitásfüggvény. A kényelem kedvéért terjesszük ki c értelmezési tartományát: legyen $c(u, v) := 0$ minden $u, v \in V$, $(u, v) \notin E$ esetén. Ekkor a (G, s, t, c) négyest* hálózat*nak nevezzük.*

Az anyag áramlását – a folyamot – úgy írhatjuk le, hogy a hálózat minden u, v csúcspárjára megmondjuk, mennyi az időegység alatt az (u, v) összeköttetésen (vagy összeköttetéseken, ha több közvetlen kapcsolat is van u és v között) átmenő $f(u, v)$ mennyiség.

Definíció (folyam): *Az $f : V^2 \to \mathbb{R}$ függvényt* folyamnak *hívjuk, ha teljesülnek a következő feltételek:*

(1) minden u, v pontpárra $f(u, v) = -f(v, u)$,

(2) minden $u \notin \{s, t\}$ pontra $\sum_{v \in V} f(u, v) = 0$,

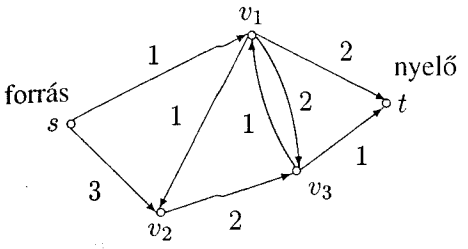
(3) minden u, v pontpárra $f(u, v) \leq c(u, v)$.

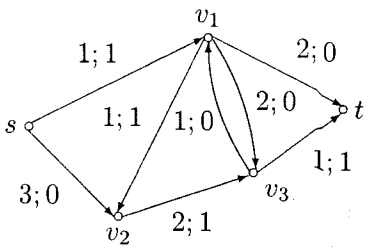
Ha $f(u, v) = c(u, v)$, akkor az (u, v) párat telítettnek *mondjuk (akár éle G-nek, akár nem). Az f* folyam értéke*, melyet $|f|$-fel jelölünk, az s-ből kimenő összes folyam, azaz $|f| := \sum_{v \in V} f(s, v)$.*

Az (1) feltétel egy kényelmes és eléggé szemléletes technikai kikötés. Azt a tényt, hogy u-ból v-be $f(u, v)$ mennyiséget juttatunk el, úgy is nézhetjük, hogy v-ből u-ba vittünk $-f(u, v)$-t. Az $f(u, v) > 0$ eseménynek pedig azt a jelentést tulajdoníthatjuk, hogy (u, v) mentén $f(u, v)$ mennyiségű anyag megy *ki* u-ból, míg $f(u, v) < 0$ azt jelenti, hogy $-f(u, v)$ megy *be* u-ba. Erre is tekintettel a (2) követelmény a Kirchoff-törvény megfogalmazása. A (3) egyenlőtlenség fejezi ki, hogy az u-ból v be (közvetlenül, kerülő utak nélkül) átvitt mennyiség nem haladhatja meg az (u, v) kapcsolat kapacitását. Az (1) és (3) alapján világos, hogy ha (u, v) és (v, u) egyike sem éle a hálózatnak, akkor $f(u, v) = f(v, u) = 0$, vagyis az ilyen párok között nincs érdemleges történés. Ebből és az (1) feltételből következik, hogy az f folyam megadásához elegendő azon $f(u, v)$ értékek ismerete, melyekre $u \to v$ éle G-nek.

A következő rajz bal oldalán egy hálózat látható. Csak a pozitív kapacitású éleket tüntettük fel, a kapacitásukkal együtt. A jobboldali rajzon egy f folyam jellemző adatai is szerepelnek. Az éleken levő számok közül az első a kapacitás, a második a rajta átáramló mennyiség. A $v_1 \to t$ él kapacitása 2, a rajta átfolyó mennyiség pedig $f(v_1, t) = 0$. A folyam értéke $|f| = 1$, és például (v_1, v_2), valamint (t, v_1) telített párok. A v_1, v_3 csúcspár között két kapcsolat van, az egyik

$v_1 \to v_3$, a másik $v_3 \to v_1$. Ezeken most 0 folyam megy át. Az összképet illetően ugyanez lenne a helyzet, ha mindkét élen 1 egység áramlana keresztül. Ekkor is igaz lenne, hogy $f(v_1, v_3) = f(v_3, v_1) = 0$. Azt szeretnénk hangsúlyozni, hogy $f(u, v)$ meghatározásához *mindegyik* u és v között menő élet figyelembe kell venni.





A maximális folyam problémája alapvető jelentőségű algoritmikus feladat mind az elméleti, mind a gyakorlati alkalmazások szempontjából. A probléma megfogalmazása és az első megoldás is L. R. Ford és D. R. Fulkerson (1957) nevéhez fűződik.

A probléma:

Adott egy (G, s, t, c) hálózat; találjunk hozzá egy maximális értékű f folyamot.

A kérdés tehát az, hogy a hálózatot folyamatosan működtetve a forráspontból maximálisan mennyi anyag tud elindulni, illetve a nyelőpontba beérkezni egységnyi idő alatt úgy, hogy a Kirchoff-törvényt és a kapacitások adta korlátokat megtartsuk. Természetesen a válaszunkban meg kell mondanunk a szállítás útvonalait is. Előre bocsátjuk, hogy a maximális folyamok értéke szükségképpen nemnegatív. Minden hálózaton van ugyanis egy nulla értékű folyam, a *nulla-folyam*, amit az $f(u, v) := 0$ egyenlőségekkel definiálhatunk $(u, v \in V)$.

Ahhoz, hogy a megoldás közelébe jussunk, vagy egyáltalán, egy maximális folyam létezését igazolni tudjuk, érdemes a *vágás* fogalmát bevezetnünk. A következő szakaszban a kombinatorikában oly gyakran előforduló minimax tételek[9] egyikét bizonyítjuk. A tétel segítségével – amely folyamok és vágások között létesít kapcsolatot – lerakjuk a megoldás alapköveit.

[9]Egy minimax tétel egy mennyiség maximumának és egy másik mennyiség minimumának az egyenlőségét mondja ki.

6.8.1. Kapcsolat a minimális vágással: a Ford–Fulkerson-tétel

Definíció (s,t**-vágás**): *Legyen* (G,s,t,c) *egy hálózat;* $A,B \subseteq V$, $A \cup B = V$ *és* $A \cap B = \emptyset$ *(azaz* A *és* B *partícionálják* V*-t). Legyen továbbá* $s \in A$, $t \in B$. *Ekkor az* A,B *halmazpárt* s,t*-vágásnak, vagy röviden vágásnak hívjuk. Az* A,B *vágás* kapacitásán *a* $c(A,B) := \sum_{u\in A, v\in B} c(u,v)$ *mennyiséget értjük. Ha* f *egy folyam, akkor definiáljuk az* A,B *vágáson áthaladó folyamot. Ezt* $f(A,B)$*-vel jelölve:* $f(A,B) := \sum_{u\in A, v\in B} f(u,v)$.

Figyeljük meg, hogy $|f| = f(\{s\}, V\setminus\{s\})$. Tehát a folyam értéke az s-et valóban elhagyó folyam mennyiségét jelenti. Ezt úgy kell érteni, hogy az s-et elhagyó összes folyamból levonjuk az s-be visszatérő összes folyamot, hiszen azon v pontokra, melyekből s-be folyik pozitív mennyiségű folyam, $f(s,v)$ negatív, így az összeget a megfelelő értékkel csökkenti. Ezzel összhangban az A,B vágáson áthaladó folyam azt fejezi ki, hogy mekkora mennyiségű anyag jut át rendszerünk A „partjáról" a B-re. Nyilván ugyanannyi, mint amennyi s-ből elindul, mivel útközben nem hagyunk ott sehol semmit. Ezt fejezi ki szabatosan a következő lemma:

Lemma: *Tetszőleges* A,B s,t*-vágásra és* f *folyamra* $|f| = f(A,B)$.

Bizonyítás: Figyelembe véve, hogy $B = V \setminus A$, az $f(A,B)$ összeg így írható:

$$f(A,B) = \sum_{u\in A, v\in B} f(u,v) = \sum_{u\in A, v\in V} f(u,v) - \sum_{u\in A, v\in A} f(u,v) = |f| - 0 = |f|.$$

Itt $\sum_{u\in A, v\in V} f(u,v) = |f|$, mert ha $u \neq s$, akkor $\sum_{v\in V} f(u,v) = 0$ a folyam definíciójának (2) feltétele miatt; $u = s$ esetén pedig az összeg a folyam értékét adja. Az (1) feltétel szerint meg $f(u,v) + f(v,u) = 0$, ezért $\sum_{u\in A, v\in A} f(u,v) = 0$. □

Ezen a ponton láthatjuk, hogy a forrásnál keletkező anyag hiánytalanul eljut a nyelőhöz, ha a szállítás útját-módját a folyam-definícióban kikötöttek szerint választjuk meg. Az előző lemma alkalmazható ugyanis az $A = V \setminus \{t\}$, $B = \{t\}$ speciális esetre. Arra jutunk, hogy $|f| = f(V\setminus\{t\}, \{t\})$; a jobb oldalon éppen a t-ben elnyelődő mennyiség szerepel.

Az A,B vágás $c(A,B)$ kapacitása korlátot ad az A-ból B-be átvihető anyag mennyiségére. Szemléletesen azt várnánk, hogy $s \in A$ és $t \in B$ miatt a vágás kapacitása korlátozza az $|f|$ értéket is. Ez így is van:

Állítás: *Tetszőleges* A,B s,t*-vágásra és* f *folyamra* $|f| \leq c(A,B)$.

Bizonyítás:

$$|f| = \sum_{u \in A, v \in B} f(u,v) \leq \sum_{u \in A, v \in B} c(u,v) = c(A,B).$$

Az első egyenlőségnél a lemmát, az egyenlőtlenségnél pedig a folyam-definíció (3) feltételét használtuk. □

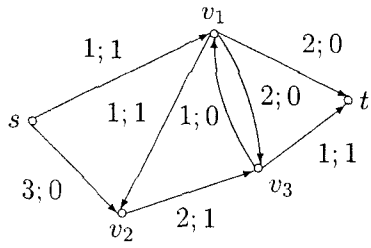
Tehát egy folyam értéke nem lehet nagyobb egy tetszőleges vágás kapacitásánál. A megígért minimax tétel így már sejthető: az egyenlőség el is érhető, azaz a maximális folyam értéke egyenlő a minimális vágáskapacitással. Hogy ezt bizonyítani tudjuk, szükségünk lesz néhány további fogalomra.

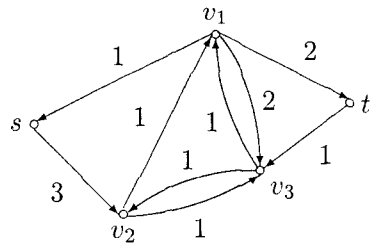
A javító gráf

Tegyük fel, hogy már eljutottunk valameddig egy nagy folyam tervezgetésében. Van egy f folyamunk, és az érdekel bennünket, hogy lehet-e javítani rajta, és ha igen, akkor hogyan. E célból nézzünk a folyam rajzára, és próbáljuk meg feltérképezni a még kiaknázatlan lehetőségeket. Minden egyes (u,v) csúcspárra vizsgáljuk meg, hogy mennyi anyagot tudnánk *még* átvinni közvetlenül u-ból v-be. A folyam-definíció (3) feltétele szerint az (u,v)-n átvihető mennyiség legfeljebb $c(u,v)$, ezért a növelési lehetőség e kapcsolat mentén $c(u,v) - f(u,v)$. Ezek a mennyiségek nem negatívak, hiszen f egy folyam; az f növelésére ott van esélyünk, ahol $c(u,v) - f(u,v) > 0$. Ezek az észrevételek alapot nyújtanak a következő fogalomhoz.

Definíció (javító gráf): *Adott egy $(G = (V,E), s, t, c)$ hálózat, és rajta egy f folyam. Jelölje $r(u,v) := c(u,v) - f(u,v)$ a maradék kapacitások függvényét. Az f folyamhoz tartozó* javító gráf *a $G_f = (V, E_f)$ gráf az élein értelmezett r kapacitásfüggvénnyel, ahol $E_f = \{(u,v);\, r(u,v) > 0\}$.*

A következő ábra bal felén az előző példabeli hálózat és folyam szerepel. Mellette a folyamhoz tartozó G_f javító gráf látható, az éleken a maradék kapacitásokkal. Vegyük észre, hogy G_f-nek vannak olyan élei is, amelyek G-nek nem élei. Ilyen például a $t \to v_3$. Ennek jelenléte indokolt, minthogy $c(t,v_3) = 0$, $f(t,v_3) = -1$, amiből $r(t,v_3) = 1$.





Az r függvény pozitív értékeket vesz fel a G_f élein, így (G_f, s, t, r) egy hálózat. Könnyen belátható, hogy egy $f' : V^2 \to \mathbb{R}$ függvény akkor és csak akkor folyam G_f-ben, ha $f + f'$ folyam G-ben; ekkor pedig $|f + f'| = |f| + |f'|$. Ha tehát találunk egy pozitív értékű f' folyamot G_f-ben, akkor $f + f'$ egy az f-nél nagyobb értékű folyam lesz az eredeti hálózatban. A folyam növelésének feladatát ezzel visszavezettük egy rokon kérdésre: pozitív értékű folyamot kell találnunk a (G_f, s, t, r) hálózatban.

Hogyan keressünk ilyen folyamot? Tegyük fel, hogy G_f-ben találunk egy irányított $s \rightsquigarrow t$ utat, és az úton szereplő élek kapacitásainak minimuma $d > 0$. Ez az út egy d értékű f' folyamot határoz meg G_f-ben: f' vegyen fel d-t az út élein, $-d$-t az út éleinek fordítottjain és nullát az összes többi páron.

Definíció (növelő út, kritikus él): *Legyen (G, s, t, c) hálózat, és f egy folyam rajta. A G_f-beli irányított $s \rightsquigarrow t$ utakat* növelő utak*nak hívjuk. Egy növelő úton szereplő élek maradék kapacitásainak minimumát az úthoz tartozó* kritikus kapacitás*nak, a megfelelő éleket pedig* kritikus élek*nek*[10] *nevezzük.*

Példánkban $sv_2v_3v_1t$ növelő út, ami mentén eggyel növelhetjük a folyamot. Növelő út sv_2v_1t is, ahol a (v_2, v_1) él menti növelés az eredeti gráfban a (v_1, v_2) él menti csökkentést jelenti (kevesebbet viszünk a rossz irányba). Mindkét növelő út kritikus kapacitása 1. Az első út kritikus élei $v_2 \to v_3$ és $v_3 \to v_1$, a másodiké pedig egyedül $v_2 \to v_1$.

Az eddigiekből kezd alakot ölteni egy algoritmus: a meglevő f folyamunkhoz készítsük el a G_f javító gráfot, ebben keressünk növelő utat, e mentén növeljük meg a folyam értékét a kritikus kapacitással; azután ezt ismételjük. Kiindulásnak jó lesz a nulla-folyam. Kérdés, mi van akkor, ha már nincs G_f-ben növelő út. A következő tétel egyebek között azzal biztat bennünket, hogy ebben az esetben *készen vagyunk, vagyis f már maximális.*

[10] A *kritikus él* kifejezést - egészen más jelentéssel - már használtuk a PERT-módszer kapcsán. A két fogalom között nincs tartalmi kapcsolat.

Tétel (Ford–Fulkerson-tétel): *Legyen f egy folyam a (G, s, t, c) hálózatban. A következő állítások egyenértékűek:*

(a) f egy maximális (értékű) folyam;

(b) f-hez nem létezik növelő út;

(c) létezik olyan A, B s, t-vágás, melyre $c(A, B) = |f|$.

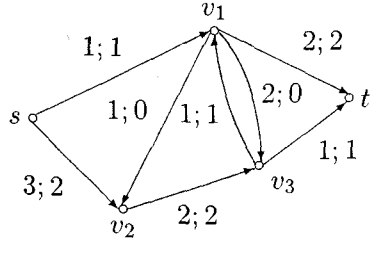
Bizonyítás:

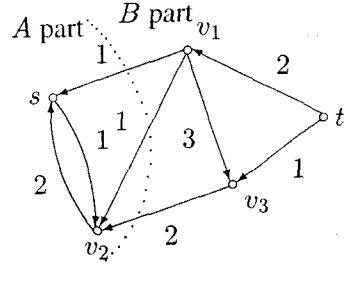
(a)⇒(b): Ha f-hez létezik növelő út a G_f gráfban, akkor f értéke az előzőek szerint ennek az útnak a $d > 0$ kritikus kapacitásával növelhető; ekkor f nem lehet maximális, mert van $|f| + d$ értékű folyam is.

(b)⇒(c): Tegyük fel, hogy nem létezik növelő út f-hez. Legyen A azon pontok halmaza, amelyek G_f-ben s-ből irányított úton elérhetők, és legyen $B := V \setminus A$. Nyilvánvaló, hogy $s \in A$, és $t \in B$. Az A, B pár tehát egy s, t-vágás. Legyenek most $u \in A$, $v \in B$ tetszőleges pontok. Az (u, v) pár nem éle G_f-nek. Ez annyit tesz, hogy az (u, v) pár telített, vagyis $c(u, v) = f(u, v)$. Ezeket az egyenlőségeket összegezve az összes $u \in A$, $v \in B$ párra azt kapjuk, hogy $c(A, B) = f(A, B)$. A lemmából tudjuk, hogy $f(A, B) = |f|$, amiből $c(A, B) = |f|$.

(c)⇒(a): Az állítás szerint $|f| \leq c(A, B)$ igaz bármely f folyamra és A, B vágásra. Az $|f| = c(A, B)$ egyenlőség ezért csak úgy lehetséges, hogy f egy maximális folyam, A, B pedig egy minimális (kapacitású) vágás. □

A következő ábra a Ford–Fulkerson-tételben felmerült helyzetet szemlélteti. A bal oldali rajzon a korábban már vizsgált hálózat egy f maximális folyamát mutatjuk. Látható, hogy $|f| = 3$. Mellette a folyamhoz tartozó javító gráf található; ezen feltüntettük a bizonyításból adódó A, B vágást. A vágás A partját az s és v_2 csúcsok alkotják. Az eredeti hálózat rajzára pillantva meggyőződhetünk róla, hogy a vágás kapacitása $c(A, B) = c(s, v_1) + c(v_2, v_3) = 1 + 2 = 3$, ami éppen a folyam értéke.





A Ford–Fulkerson-tétel érdekes következménye, hogy egy folyam maximális voltának van egyszerűen és gyorsan ellenőrizhető bizonyítványa. Képzeljük el,

hogy valaki mutat nekünk egy kusza hálózatot, rajta egy f folyammal, és azt állítja, hogy f maximális. Hogyan győzhet meg bennünket erről egyszerűen? A tétel szerint elegendő egy olyan A, B vágást mutatnia, aminek a kapacitása éppen a folyam értéke. Mivel $|f|$ és $c(A, B)$ is gyorsan kiszámolható, egykettőre bizonyosságot szerezhetünk az állítás igazságáról. Az algoritmikus problémáknak azzal a tulajdonságával, hogy a válasznak van-e hatékonyan ellenőrizhető bizonyítványa, a 8. fejezetben foglalkozunk alaposabban.

6.8.2. A Ford–Fulkerson-algoritmus

A maximális folyam keresésére szolgáló *Ford–Fulkerson-algoritmus* lényegi lépéseit tulajdonképpen már elmondtuk: folyamok $f_0, f_1, \ldots, f_k = f^*$ sorozatát konstruáljuk meg sorban egymás után. Az f_0 az azonosan nulla folyam. Az f_i birtokában f_{i+1}-et úgy kapjuk, hogy az f_i javító gráfjában keresünk egy javító utat; az út mentén a d_i kritikus kapacitással növelve kapjuk az f_{i+1} folyamot. Érvényes tehát az $|f_{i+1}| = |f_i| + d_i$ összefüggés. Akkor állunk meg, amikor a folyamhoz már nem létezik növelő út. A Ford–Fulkerson-tételből következik, hogy ekkor egy maximális folyamunk van.

Feladat: Mutassuk meg, hogy ha a hálózat éllistával adott, akkor a növelő lépés, vagyis f_i-ből f_{i+1} számítása $O(e)$ elemi művelettel kivitelezhető, ahol e a G éleinek száma. (Feltehető, hogy G összefüggő. A G_{f_i} javító gráfnak legfeljebb csak $2e$ éle lehet, így a hálózat és f_i ismeretében $O(e)$ költséggel megépíthető. Javító utat ezután a G_{f_i} gráf s-ből rajtoló szélességi bejárásával kaphatunk.)

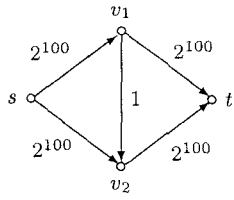
Mit mondhatunk a növelő lépések számáról? Ha az élkapacitások egészek, akkor legfeljebb $|f^*|$ számú növelés után készen vagyunk (f^* egy maximális folyamot jelöl), ugyanis ekkor a d_i kritikus kapacitások pozitív egészek lesznek. Rögzítsük ennek a fejtegetésnek egy fontos következményét:

Következmény: *Legyen (G, s, t, c) egy hálózat, és tegyük fel, hogy a $c(u, v)$ kapacitások mind egész számok. Ekkor van olyan f^* maximális folyam, hogy az $f^*(u, v)$ értékek egészek $(u, v \in V)$. Ebből adódóan a maximális folyam értéke is egész.*

Bizonyítás: A Ford–Fulkerson-algoritmus működéséből látható, hogy az $f_i(u, v)$ értékek egészek lesznek. Ez nyilvánvaló a nulla-folyamra. Utána pedig használhatjuk, hogy az $f_{i+1}(u, v) - f_i(u, v)$ különbség a $0, \pm d_i$ egészek valamelyike. □

Ha tehát a kapacitások egészek, akkor legfeljebb $|f^*|$ növeléssel eljutunk egy maximális folyamig. Ez egyfelől megnyugtató, hiszen látjuk, hogy véges sok lépés

elég. Másfelől a növelések száma ettől még égbekiáltóan nagy is lehet, amint azt az alábbi hálózat mutatja.



Ha ebben a hálózatban felváltva mindig az sv_1v_2t és az sv_2v_1t növelő utak mentén javítunk (a kritikus kapacitás 1 lesz), akkor a nulla-folyamból 2^{101} javítás után kapunk maximális folyamot. Ugyanakkor két javító menet is elég lenne: először az sv_1t, majd az sv_2t út mentén.

Ha az élkapacitások racionális számok, akkor a közös nevezőjükkel való szorzás után egész kapacitásaink lesznek. Így ebben az esetben is véges sok (noha esetleg nagyon sok) növelő lépést hajtunk végre.

Ha irracionális kapacitásokat is megengedünk, akkor előfordulhat, hogy a javító utak balszerencsés választása miatt az eljárás nem ér véget véges sok lépésben. Sőt, még az is megeshet, hogy az $|f_i|$ értékek a maximális folyamokénál kisebb értékhez konvergálnak.

Mindezekből nyilvánvaló, hogy nagyon nem mindegy, miként választunk növelő utat. A találomra való választással racionális kapacitások esetén ugyan egy működő, ámde lassú algoritmust kapunk, irracionális kapacitásokkal pedig a módszer nem is működik.

6.8.3. Edmonds–Karp és Dinic algoritmusai

J. Edmonds és R. M. Karp (1972) nagyon egyszerűen hangzó receptet javasolt az előző szakaszban említett bajok orvoslására. Módszerük rokonszenves vonása, hogy a javító menetek számára csak n-től és e-től (azaz csupán a hálózat pontjainak és éleinek számától) függő korlát adható. Javaslatuk a Ford–Fulkerson-módszer pontosítása, amennyiben arról rendelkezik, hogy miként válasszunk növelő utat az adott f folyamhoz:

> EDMONDS–KARP-HEURISZTIKA: A folyam növelésére mindig a legrövidebb – vagyis a legkevesebb élből álló – növelő utak egyikét válasszuk.

Edmonds és Karp ötletéről nem látszik elsőre, hogy miért olyan jó. Az mindenesetre biztató, hogy az előző rajz hálózatában a nulla-folyamból kiindulva két menetben eljut a maximálisig.

A gondosabb elemzéshez tegyük fel, hogy adott a (G, s, t, c) hálózat és rajta egy f folyam. Tekintsük a G_f javító gráfot; legyen benne π egy legrövidebb növelő út. Ennek hosszát (élszámát) jelölje l. Osszuk rétegekre a G_f csúcsait az s-től való távolság szerint. Az s-ből egyetlen irányított éllel elérhető csúcsok alkotják az első réteget, és t az l-edik rétegbe kerül. Ezt a beosztást kapjuk, ha s-től indulva szélességi bejárással végigmegyünk az s-ből irányított úton elérhető v csúcsokon, és közben számítjuk a $D[v]$ értékeket (lásd a 6.5. szakaszt). Világos, hogy a v és w csúcsok pont akkor vannak egy rétegben, ha $D[v] = D[w]$. Kiemeljük a felosztás két fontos tulajdonságát:

(1) Egy él legfeljebb egy réteggel mehet előre.

Ez nyilvánvaló, hiszen az s-től i távolságra levő pontból kiinduló él végpontjának a távolsága nem lehet több, mint $i + 1$. A G_f egy $x \to y$ élét nevezzük *vastagnak*, ha $D[y] = D[x] + 1$. Az (1) tény hasznos folyománya a következő:

(2) Az l hosszúságú $s \rightsquigarrow t$ utak csupa vastag élből állnak, és nincs l-nél rövidebb $s \rightsquigarrow t$ út.

Nézzük most, miként módosul a kép, ha π mentén növeljük f-et a kritikus kapacitással. Hogyan kapható meg az új f' folyam $G_{f'}$ javító gráfja G_f-ből? A π út legalább egy éle – nevezetesen egy kritikus él – telített lesz, így eltűnik a javító gráfból. Ezenkívül legfeljebb l darab új él jelenik meg a $G_{f'}$-ben (π élei ellenkező irányítással, ha eddig még nem szerepeltek), de ezek az új élek mind visszafelé, alacsonyabb sorszámú rétegbe mennek. Arra jutunk innen, hogy (1) és (2) a $G_{f'}$-re vonatkozóan is igaz marad (noha esetleg a rétegezés már nem tükrözi hűen a távolságokat). Ebből viszont azonnal látszik, hogy *a következő növelő út sem lehet rövidebb l-nél.*

Nézzük meg, hogy hányszor adódhat egymás után l hosszú növelő út. Ha van ilyen $G_{f'}$-ben, akkor a növelés utáni javító gráfban eggyel kevesebb vastag él lesz (legalább egy kritikus él törlődik), és az (1), (2) tények is érvényben maradnak, mert a növelés eredményeként csak visszafelé mutató élek keletkezhetnek.

A (2) tulajdonság öröklődése miatt addig lesz l élből álló növelő út, amíg marad vastag élekből álló $s \rightsquigarrow t$ út. A vastag élek viszont minden növelésnél fogynak. Így legfeljebb e ilyen növelés lehetséges. Érvényes tehát a következő:

Tétel: *Az Edmonds–Karp-heurisztika szerinti növelésnél a növelő utak hosszai nem csökkenő sorozatot alkotnak. Ebben a sorozatban egy adott úthosszúság legfeljebb e-szer fordulhat elő. Következésképpen legfeljebb $e(n-1)$ növelés lehetséges. A heurisztika alkalmazásával $O(e^2 n)$ elemi lépésben kapunk maximális folyamot.*

Bizonyítás: Az első két állítást már beláttuk. A harmadik – a növelések számáról szóló – azonnal következik ezekből, hiszen a növelő utak egyszerű utak. A lehet-

séges hosszúságaik ezért $1, 2, \ldots, n-2, n-1$, és bármely hosszúság legfeljebb e-szer léphet fel.

Az utolsó állításhoz elég igazolni, hogy *egy* növelés $O(e)$ időben elvégezhető. A növelés költségéről korábban mondottakhoz csak annyit kell hozzátenni, hogy a javító gráfban az s-ből indított szélességi bejárásnál a faéleket követve éppen egy legrövidebb $s \rightsquigarrow t$ utat kapunk. □

A bizonyítást jobban szemügyre véve kiderül, hogy f javításakor az l hosszú növelő utak élei mind vastag élek G_f-ben. Az összes ilyen javítás műveletigényére $O(e^2)$ korlátot kaptunk (legfeljebb e növelés darabonként $O(e)$ költséggel). Kihasználva, hogy a vastag élekből álló gráf egy DAG, ezek a növelések gyorsabban is elvégezhetők. De nézzük pontosabban, mi is a cél:

Legyen (H, s, t, r) állistával adott hálózat, ahol H egy DAG, és r az élein értelmezett kapacitásfüggvény. Olyan g folyamot – ún. blokkoló folyam*ot keresünk, amelyhez minden $s \rightsquigarrow t$ irányított úton van telített él.*

Feladat: Igazoljuk, hogy egy maximális folyam egyben blokkoló folyam is. Mutassuk meg, hogy a megfordítás nem igaz: adjunk példát blokkoló folyamra, ami nem maximális.

Ha találunk egy g blokkoló folyamot a G_f vastag éleiből álló H gráfban, akkor $f + g$ már biztosan nem növelhető a vastag élek mentén, azaz minden további növelés csak l-nél hosszabb úton lehetséges.

Dinic módszere DAG-beli blokkoló folyam keresésére

Tegyük fel, hogy a H DAG-nak n csúcsa és m éle van. E. A. Dinic (1970) algoritmusa a nulla-folyamból indulva szerkeszt egy g blokkoló folyamot. Az algoritmus történései három csoportba sorolhatók; ezek a *nyomulás*, a *növelés* és a *takarítás*.

Nyomulás során egy s-ből induló π irányított utat építünk. Amíg csak lehetséges, π-t az utolsó pontjából induló valamelyik él végpontjával bővítjük. Akkor állunk meg a nyomulással, amikor vagy t-be értünk, vagy már nem tudunk tovább menni, mert nem vezet ki él a π utolsó pontjából. Mivel H egy DAG, a π sosem állhat több, mint n csúcsból. Ebből következően egy nyomulás $O(n)$ költséggel megvalósítható.

Növelésre akkor kerül sor, ha a nyomulással t-be értünk. Ez esetben π egy növelő út. Az aktuális g folyamot π mentén növeljük a d_π kritikus kapacitással. Ezután az út éleinek kapacitását d_π-vel csökkentjük, és töröljük a kritikus éleket (amelyek kapacitása nullára változott). Egy növelés műveletigénye arányos a π hosszával, vagyis $O(n)$-nel becsülhető.

Ha a nyomulással egy $x \neq t$ csúcsnál akadunk el (azaz nem tudunk tovább menni), akkor *takarítás* következik. Ez azt jelenti, hogy töröljük H-ból az x-be menő éleket. Egy takarítás a törölt élek számával arányos lépésszámban végehezvihető.

A takarítás és a növelés végeztével (s-ből induló) nyomulásba kezdünk. A munka akkor fejeződik be, amikor már nincs s-ből kimenő él.

Tétel: *Dinic módszerével* [11] $O(mn)$ *művelettel kapunk blokkoló folyamot a* H *DAG-ban.*

Bizonyítás: A módszer helyessége nyilvánvaló: amikor ugyanis egy élet törlünk, akkor rajta keresztül már nem vezethet az aktuális g-t növelő út. Így a munka végeztével nem létezik csupa H-beli élekből álló növelő út. Úgy is fogalmazhatunk, hogy minden $s \rightsquigarrow t$ irányított úton van telített él, tehát a végső g egy blokkoló folyam H-ban.

Ami a költségeket illeti, vegyük észre, hogy a takarítások és a növelések száma együttesen sem lehet több m-nél, mivel ezek élek törlésével járnak. A takarítások összköltsége $O(m)$, ugyanis ezek során legfeljebb m élet törölhetünk; a növeléseké pedig $O(mn)$. Nézzük most a nyomulásokat: a legutolsó kivételével minden nyomulást növelés vagy takarítás követ, így legfeljebb $m + 1$ nyomulás lehetséges. Ezek együttes lépésszáma ezért $O(mn)$. Mindent számba véve $O(m) + O(mn) + O(mn) = O(mn)$ lépés elegendő. □

Ha az Edmonds–Karp-heurisztika helyett Dinic módszerét alkalmazzuk az f folyam növelésére, akkor legfeljebb $n - 1$ ilyen növelésre lesz szükség. Az előző tétel alapján egy növelés költsége $O(en)$, mert a vastag élekből álló gráfnak legfeljebb e éle lehet. A Dinic-algoritmussal tehát $O(en^2)$ *lépésben kapunk maximális folyamot.*

Megjegyezzük, hogy DAG-beli blokkoló folyam keresésére van $O(n^2)$ lépésszámú módszer is, amivel $O(n^3)$ költséggel adódik maximális folyam. Az első ilyen módszert A. V. Karzanov találta 1974-ben. A folyamalgoritmusoktól búcsúzóban megemlítjük, hogy a legjobb ismert módszerek költsége $O(en \log(n^2/e))$ (A. V. Goldberg, R. E. Tarjan, 1986) és $O(en)$, ha $e > n^{5/3+\epsilon}$ (N. Alon, J. Cheriyan, T. Hagerup, 1990). Érdekes nyitott kérdés, hogy létezik-e minden hálózaton $O(en)$ lépésszámban célt érő folyamalgoritmus.

[11] Valójában Dinic módszerének egyszerűsített változatát ismertettük. Például takarítás után nem kell feltétlenül s-től kezdeni a nyomulást. Jobban járunk, ha csak a π út x előtti pontjáig megyünk vissza, és onnan nyomulunk előre.

6.8.4. Alkalmazások

Három példával szeretnénk érzékeltetni a folyamok széleskörű és sokszínű alkalmazási lehetőségeit. Hogy egyszerűsítsünk a dolgokon, olyan hálózatokra szorítkozunk, amelyekben bármely u, v pontpár között legfeljebb csak az egyik irányba megy él. Ezáltal a $v \to w$ élen átfolyó mennyiséget azonosíthatjuk az $f(v, w)$ értékkel. Így persze az is feltehető, hogy ha $v \to w$ éle a hálózatnak, akkor $f(v, w) \geq 0$. Amikor pedig egy g folyamot adunk meg, akkor elég a $g(v, w)$ értékeket leírni, ahol $v \to w \in E$.

Példáink kapcsán az olvasó könnyen meggyőződhet róla, hogy a javasolt egyszerűsítés nem jelenti az általánosság csorbítását. Ha ugyanis $u \to v$ és $v \to u$ is éle a hálózatnak, akkor utóbbit helyettesíthetjük egy $v \to v^* \to u$ élpárral, ahol v^* egy új csúcs. A $v \to v^*$ és a $v^* \to u$ élekre ugyanazt (pl. kapacitást) írjuk, mint ami a $v \to u$ élen szerepel.

Élidegen utak irányított gráfokban

Legyen G egy irányított gráf, s és t két csúcsa. *Szeretnénk minél több G-beli $s \rightsquigarrow t$ irányított utat találni azzal a feltétellel, hogy bármely két út élidegen (másként mondva: közös él nélküli) legyen.*

Nézzük evégből a (G, s, t, c) hálózatot, ahol a G minden (u, v) élének a $c(u, v)$ kapacitása 1. A kapacitások egészek, tehát van olyan f maximális folyam, amelyre az $f(u, v)$ értékek egészek minden $u, v \in V$ csúcspárra. Ebből azonnal következik, hogy ha $u \to v \in E$, akkor $f(u, v)$ vagy 0 vagy 1. Tegyük fel, hogy a folyam értéke k. Megmutatjuk, hogy ekkor van G-ben k darab páronként élidegen $s \rightsquigarrow t$ út. Ennek első lépéseként ajánljuk a következő feladatot:

Feladat: Mutassuk meg, hogy ha $k > 0$, akkor van olyan π út s-ből t-be, amelynek az élein f 1-et vesz fel. (Induljunk el s-ből egy olyan $u \to v \in E$ élen, amelyre $f(u, v) = 1$. A v-be érve töröljük ezt az élet, majd lépjünk tovább egy olyan $v \to w$ élen, melyre $f(v, w) = 1$, és töröljük ezt az élet is. Így folytatva sosem akadhatunk el egy $x \neq t$ csúcsban; x-ből ugyanis legalább annyi 1-es indul ki, mint amennyi befut oda. Előbb-utóbb t-be érünk, mert az élek fogynak.)

Ha tehát $k > 0$, akkor nézzünk egy a feladat szerinti π utat. Töröljük G-ből a π éleit, és feledkezzünk meg a rajtuk átmenő egységnyi folyamról. A kapott gráfban egy $k - 1$ értékű folyam marad, amire $k - 1 > 0$ esetén ismételjük az előző útkeresést. Így folytatva k darab élidegen irányított $s \rightsquigarrow t$ utat kapunk.

Fordítva, ha $\pi_1, \pi_2, \ldots, \pi_k$ páronként élidegen irányított $s \rightsquigarrow t$ utak G-ben, akkor ezek mindegyikén egységnyi folyamot küldhetünk a forrásból a nyelőbe,

ami összesen egy k értékű folyamot jelent. A maximális folyam értéke tehát megegyezik a legnagyobb élidegen $s \rightsquigarrow t$ utakból álló rendszer elemszámával.

Az eddigiek egy $O(en^2)$ költségű algoritmust sugallnak egy ilyen útrendszer megtalálására: először kiszámítunk egy f egész értékkészletű maximális folyamot, majd ebből összerakunk egy maximális útrendszert. A költségek közül a domináns rész a Dinic-módszer lépésszáma, amivel az f folyamot számítjuk.

Feladat: Mutassuk meg, hogy f ismeretében $O(e)$ lépésben kaphatunk k darab páronként élidegen $s \rightsquigarrow t$ utat. (Lásd az előző feladathoz mellékelt ötletet.)

Mielőtt továbbmennénk, lássuk, mit mond esetünkben a Ford–Fulkerson-tétel. A tétel szerint van egy A, B vágás, aminek a $c(A, B)$ kapacitása éppen az útjaink k száma. Más szóval A-ból B-be k irányított él megy. Ez a k él *lefogja* az összes $s \rightsquigarrow t$ utat abban az értelemben, hogy minden ilyen út tartalmaz A-ból B-be menő élet (mert $s \in A$ és $t \in B$). Ezzel egyrészt egy másik bizonyítást kaptunk arra, hogy k-nál több páronként élidegen út nem létezhet. Másfelől lényegében igazoltuk K. Menger nevezetes gráfelméleti tételét, amely szerint *a G-beli páronként élidegen $s \rightsquigarrow t$ utak maximális száma megegyezik az $s \rightsquigarrow t$ utakat lefogó élek minimális számával.*

Itt $maximum \leq minimum$ nyilvánvaló, mert egy lefogó élrendszer az útjaink mindegyikéből tartalmaz élet. Másfelől az előbb kiderült, hogy van olyan páronként élidegen $s \rightsquigarrow t$ utakból álló rendszer, ami éppen annyi útból áll, mint egy lefogó élhalmaz (utóbbi az A-ból B-be menő élek összessége). Innen pedig már adódik a $maximum \geq minimum$ egyenlőtlenség.

Maximális párosítás páros gráfokban

Egyszerű megfogalmazása ellenére ... máig is talán ez az egész párosításelmélet kulcseredménye.
LOVÁSZ LÁSZLÓ, MICHAEL D. PLUMMER[12]

Egy olyan problémát fogalmazunk meg a folyamok nyelvén, amivel már találkoztunk: a maximális párosítás keresését páros gráfokban. Ezt elsősorban a megfeleltetés egyszerűsége és elméleti érdekessége miatt tesszük, és nem azért, mert gyors algoritmust remélünk belőle. A folyamalgoritmusokra épülő megoldások nem tudnak versenyezni a párosítási feladat sajátosságaira kihegyezett közvetlen módszerekkel, így a magyar módszerrel sem.

[12] Ezzel a méltatással vezetik fel König Dénes minimax tételét a párosítások elméletével foglalkozó nagymonográfiájuk (*Matching Theory*, Akadémiai Kiadó, Budapest, 1986) 4. oldalán.

Legyen $G = (V_1, V_2; E)$ egy páros gráf, amiben maximális párosítást keresünk. Készítsünk belőle hálózatot az alábbiak szerint: először is irányítsuk az E-beli éleket V_1-től V_2 felé; ezek kapacitása legyen $|V_1| + 1$. Vegyünk fel még a meglevő pontokon kívül egy s forrást és egy t nyelőt. Az s-ből vezessünk 1 kapacitású éleket a V_1 pontjaiba, és indítsunk ugyancsak 1 kapacitású éleket a V_2 pontjaiból t-be.

Megmutatjuk, hogy *az így kapott hálózatban a maximális folyamok értéke megegyezik a G-beli maximális párosítások méretével.* Legyen f egy csupa egész értékű maximális folyam, és legyen $|f| = k$. A V_1 pontjaiba legfeljebb egységnyi folyam folyhat be, ezért egy $u \in V_1$ ponthoz legfeljebb egy olyan $(u, v) \in E$ él illeszkedhet, amelyre $f(u, v) = 1$. A többi ilyen élen f nullát vesz fel. Hasonló érvényes a V_2 pontjaira. Ezek szerint azok az $(u, v) \in E$ élek, melyekre $f(u, v) = 1$, egy párosítást alkotnak G-ben. A rajtuk átmenő összes folyam egyrészt megegyezik az élek számával. Másrészt a lemma szerint k-val is, hiszen ez a mennyiség az $\{s\} \cup V_1, \{t\} \cup V_2$ vágáson átáramló $f(\{s\} \cup V_1, \{t\} \cup V_2)$ folyam. Van tehát a G-ben k élből álló párosítás.

Fordítva, ha az $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k) \in E$ élek egy párosítást alkotnak ($u_i \in V_1, v_i \in V_2$), akkor az $s \to u_i \to v_i \to t$ utak mindegyikén egységnyi folyamot vihetünk a hálózatunkban s-ből t-be; ezek összesen egy k értékű folyamot adnak. A maximális folyamok értéke tényleg egyenlő a maximális párosítások élszámával.

Próbáljuk meg itt is értelmezni a Ford–Fulkerson-tétel állítását, ahogy azt az élidegen utaknál tettük! A tétel alapján létezik egy A, B s, t-vágás, amelynek a k kapacitása éppen a maximális folyamok értéke, vagyis a maximális párosítások élszáma. Legyenek $x_1, x_2, \ldots, x_l$ a V_1 azon pontjai, amelyek *nincsenek* A-ban, és legyenek $y_1, y_2, \ldots, y_m$ az A halmaz V_2-beli pontjai. Az A-ból B-be menő élek között nem lehet olyan, ami V_1-ből V_2-be mutat, mert egy ilyen él kapacitása túl nagy: $|V_1| + 1 > k$. Ebből arra következtethetünk, hogy az E-beli élek mindegyike illeszkedik az x_i vagy az y_j pontok valamelyikéhez. Az x_i és az y_j pontok ebben az értelemben egy a G éleit *lefogó ponthalmazt* alkotnak. Mekkora ez a halmaz? Az A-ból B-be menő élek éppen az $s \to x_i$ és az $y_j \to t$ élek. Ezek kapacitása 1, ami azt jelenti, hogy a számuk éppen a vágás kapacitása, k. Van tehát a G-ben a maximális párosítások élszámával egyező méretű lefogó ponthalmaz.

Ezzel a birtokunkba jutott a gráfelmélet egyik tündöklő ékköve, König Dénes minimax tétele, amely így hangzik: *egy G páros gráf maximális párosításainak a mérete megegyezik a G éleit lefogó ponthalmazok minimális elemszámával.*

A $maximum \leq minimum$ egyenlőtlenség itt is nyilvánvaló: egy k élű párosítás éleit nem lehet k-nál kevesebb ponttal lefogni. Az előző okfejtés adja a jóval tartalmasabb $maximum \geq minimum$ egyenlőtlenséget.

Feladat: A bevezető fejezetben a lovagok és udvarhölgyek problémájáról megemlítettük, hogy pontosan akkor *nincs* megoldása, ha a megfelelő gráfban van König-akadály. Bizonyítsuk be ezt az állítást. (Alkalmazzuk König minimax tételét.)

Hálózatok alsó korlátokkal

A hálózatfogalom, amivel eleddig foglalkoztunk, meglehetősen egyszerűnek mondható. Gyakran előfordul, hogy az alkalmazások követelményei ennél finomabb, több tényezőre is érzékeny modelleket kívánnak meg. Most – rövid ízelítő gyanánt – egy ilyen árnyaltabb hálózat folyamairól ejtünk pár szót.

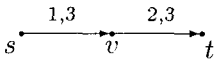
Tegyük fel, hogy a $c(u,v)$ kapacitások mellett – amelyek felülről korlátozzák az u-ból közvetlenül v-be áramló folyamot – *alsó korlátjaink* is vannak az $f(u,v)$ mennyiségekre. Az ilyen hálózatokat formálisan egy (G,s,t,c,k) alakú ötössel írhatjuk le. Ebben az első négy összetevő értelmezése a régi. A k a G élein értelmezett, nemnegatív értékű függvény, ami az élekhez rendelt alsó korlátokat mondja meg. A korábbi folyam-definíció kiegészítéseként megköveteljük, hogy $k(u,v) \leq f(u,v)$ is teljesüljön a G minden $u \to v$ élére.

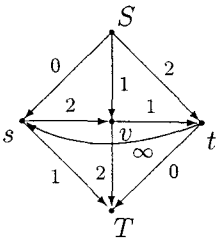
Az egyszerűbb modellnél a folyamok létezése felettébb jámbor kérdésnek bizonyult: a nulla-folyam minden hálózaton eleget tett a definíciónak. Az új modellhez nyilvánvalóan nincs ilyen univerzális választás.

Most megmutatjuk, hogy (régi értelemben vett) *maximális folyamok segítségével hatékonyan eldönthető, hogy egy $\mathcal{H} = (G,s,t,c,k)$ alakú hálózathoz létezik-e folyam.*

Évégből egy alsó korlátok nélküli $\mathcal{H}'$ hálózatot készítünk. Kényelmi okokból tegyük fel, hogy G-ben nincs $s \to t$ és $t \to s$ él. Vegyünk fel a $G = (V,E)$ pontjai mellé egy új forrást és egy új nyelőt; legyenek ezek S és T. A $\mathcal{H}'$ hálózat ponthalmaza $V \cup \{S,T\}$ lesz. A G éleit megtartjuk, de új kapacitásokat rendelünk hozzájuk. Az $(u,v) \in E$ él $c'(u,v)$ kapacitása a $\mathcal{H}'$ hálózatban legyen $c'(u,v) := c(u,v) - k(u,v)$. (Nyilván feltehetjük, hogy $c'(u,v) \geq 0$; ellenkező esetben nem létezhet folyam az eredeti hálózatban, mivel már az $u \to v$ élre kirótt feltételek sem teljesíthetők.) Adjunk még a $\mathcal{H}'$-höz egy $S \to v$ és egy $v \to T$ élet minden $v \in V$-re. Az $S \to v$ él kapacitása legyen $c'(S,v) := \sum_{(u,v)\in E} k(u,v)$, vagyis a v-be érkező G-beli élek alsó korlátjainak az összege. Hasonlóan, legyen $c'(v,T) := \sum_{(v,w)\in E} k(v,w)$, a v-ből kimenő G-beli élek korlátjainak az összege. Végül csapjunk még a hálózathoz egy ∞ kapacitású $t \to s$ élet.

A következő rajzon az átalakítást szemléltetjük. A baloldali gráf éleire írtuk az alsó korlátokat és a kapacitásokat. Így például $k(s,v) = 1$ és $c(v,t) = 3$. Mellette a megfelelő $\mathcal{H}'$ hálózat szerepel; az éleken a c' kapacitásokat tüntettük fel.





Tétel: *A $\mathcal{H} = (G, s, t, c, k)$ hálózatban akkor és csak akkor létezik folyam, ha a $\mathcal{H}'$ hálózat (S-ből T-be menő) maximális folyamának az értéke $\sum_{(u,v)\in E} k(u, v)$.*

Megjegyezzük, hogy a $\mathcal{H}'$-re vonatkozó állítás egyenértékű azzal, hogy egy maximális folyamra nézve az $S \to v$ és a $v \to T$ élek mind telítettek. Az $\{S\}$, $V \cup \{T\}$ és a $V \cup \{S\}, \{T\}$ vágások kapacitása ugyanis éppen $\sum_{(u,v)\in E} k(u, v)$.

Bizonyítás: Tegyük fel először, hogy van egy f folyamunk $\mathcal{H}'$-ben, aminek az értéke $\sum_{(u,v)\in E} k(u, v)$. Ebből egy g folyamot konstruálunk $\mathcal{H}$-ban. A G gráf minden (u, v) élére végezzük el a következőket: vonjunk le $k(u, v)$-t $f(S, v)$-ből és $f(u, T)$-ből és adjunk ugyanennyit $f(u, v)$-hez.

A módosítások során megtartottuk a Kirchoff-törvényt a G csúcsaiban (az átmenő folyamból ugyanannyit vontunk le, mint amennyit hozzáadtunk). A módosítások után az S-hez és T-hez illeszkedő éleken áramló mennyiség 0, így ezeket figyelmen kívül hagyhatjuk. Ha még a (t, s) élről és a rajta átmenő folyamról is elfeledkezünk, akkor azt kapjuk, hogy a G élein értelmezett $g(u, v) := f(u, v) + k(u, v)$ függvény s és t kivételével mindenütt eleget tesz a Kirchoff-törvénynek. Legyen most $u \to v$ egy éle G-nek. A

$$0 \leq f(u, v) \leq c'(u, v) = c(u, v) - k(u, v)$$

egyenlőtlenségekből látjuk, hogy $k(u, v) \leq f(u, v) + k(u, v) \leq c(u, v)$. Itt a középen álló mennyiség nem más, mint $g(u, v)$. Tehát g tényleg egy folyam a $\mathcal{H} = (G, s, t, c, k)$ hálózatban.

A megfordítást úgy bizonyítjuk, hogy egy $\mathcal{H}$-beli g folyamból egy alkalmas f-et konstruálunk. Ezt lényegében az előző eljárás fordítottjával tehetjük meg. Először egy f' függvényt definiálunk a $\mathcal{H}'$ élein. Legyen $f'(t, s) = |g|$, $f'(u, v) = g(u, v)$, ha $u \to v \in E$ és $f'(S, v) = f'(v, T) = 0$, ha $v \in V$. Az így kapott f' a $V \cup \{S, T\}$ minden pontjában betartja a Kirchoff-törvényt. Ezután módosítjuk f'-t. Egymás után véve az $(u, v) \in E$ éleket adjunk $k(u, v)$-t $f'(S, v)$-hez és $f'(u, T)$-hez, és vonjunk le $k(u, v)$-t az $f'(u, v)$-ből. Az előzőek mintájára könnyű megmutatni, hogy az eredményül kapott f egy maximális folyam a $\mathcal{H}'$ hálózatban, amelyre $|f| = \sum_{(u,v)\in E} k(u, v)$. Ennek a részleteit az olvasóra hagyjuk. □

A tételből következik, hogy folyamalgoritmus alkalmazásával $O(en^2)$ lépésben eldönthető, hogy van-e folyam a (G, s, t, c, k) hálózatban (n a G pontjainak, e pedig az éleinek a száma).

Feladat: Tegyük fel, hogy a $\mathcal{H}'$ konstrukciójában a $t \to s$ élhez ∞ helyett egy $d > 0$ kapacitást rendelünk. Mutassuk meg, hogy az így módosított $\mathcal{H}'$-höz pontosan akkor létezik $\sum_{(u,v)\in E} k(u, v)$ értékű maximális folyam, ha $\mathcal{H}$-ban van olyan g folyam, amelyre $|g| \leq d$.

Feladat: Tegyük fel, hogy a $\mathcal{H} = (G, s, t, c, k)$ hálózathoz tartozó kapacitások és alsó korlátok mind egészek. Legyen $d \in \mathbb{Z}$. Igazoljuk, hogy ha a $\mathcal{H}$-hoz van olyan g folyam, amelyre $|g| \leq d$, akkor olyan g^* folyam is van, amelyre $|g^*| \leq d$ szintén teljesül, és ezenfelül a $g^*(u, v)$ számok mind egészek. (A $\mathcal{H}, d$ párhoz tartozó $\mathcal{H}'$ hálózat kapacitásai mind egészek. Ennek egy egész értékkészletű maximális folyamából a tételbeli eljárást követve kaphatunk egész értékkészletű g^*-ot.)

Egy ütemezési feladat

Zárópéldaként egy ütemezési feladatot körvonalazunk, amihez – talán kissé váratlanul – egy alsó korlátokkal rendelkező hálózat ad megoldást. Tegyük fel, hogy egy légitársaság a $J_1, J_2, \ldots, J_m$ járatokat szeretné üzemeltetni, és d darab azonos típusú (mondhatni: csereszabatos) repülőgépe van erre a célra. Minden J_i, J_j járatpárra ismert, hogy van-e elég idő arra, hogy a J_i teljesítése után egy gép felkészüljön a J_j repülésére. Ha J_i, J_j-re a válasz igenlő, akkor azt mondjuk, hogy *J_j követheti J_i-t.*

A követhetőségi viszonyok – melyek egy irányított gráfot jelentenek a járatok halmazán – ismeretében szeretnénk eldönteni, hogy megvalósíthatók-e a járatok legfeljebb d repülőgéppel.

Egy olyan hálózatot készítünk, amelyben a J_i légijáratnak két csúcs, i és i', valamint az $i \to i'$ él felel meg ($1 \leq i \leq m$). Ha J_j követheti J_i-t, akkor vezessünk irányított élet i'-ből j-be. Vegyünk még fel egy s forrást és egy t nyelőt, és adjuk a hálózathoz az $s \to i$ és $i' \to t$ éleket ($1 \leq i \leq m$). Az összes él kapacitása legyen 1. Az $i \to i'$ alakú élek alsó korlátja legyen 1, a többi élé pedig 0.

Állítás: *A $J_1, J_2, \ldots, J_m$ járatok akkor és csak akkor teljesíthetők legfeljebb d géppel, ha a hálózathoz van olyan g folyam, amelyre $|g| \leq d$.*

Bizonyítás: Tegyük fel először, hogy létezik egy g folyam a hálózaton, amelyre $|g| \leq d$. Az előző feladat alapján feltehető, hogy g minden élen egész számot vesz fel. Ebből a kapacitások és a korlátok figyelembe vételével adódik, hogy $g(u, v) \in \{0, 1\}$ minden $u \to v$ irányított élre. Vegyük szemügyre azt a részgráfot, aminek

az élei az egységnyi folyamot szállító élek. Ez – az élidegen utaknál látott módon – felbontható legfeljebb d darab

$$(\dagger) \qquad s \to i_1 \to i_1' \to i_2 \to i_2' \to \cdots \to i_l \to i_l' \to t$$

út egyesítésére. Két ilyen útnak s és t kivételével nincs közös pontja. A $(\dagger)$ út létezése azt jelenti, hogy a $J_{i_1}, J_{i_2}, \ldots, J_{i_l}$ járatok egy géppel teljesíthetők. A $k(i,i') = 1$ feltételek miatt minden (i,i') él szerepel valamelyik ilyen úton. Tehát legfeljebb d géppel tényleg minden járat lebonyolítható.

Fordítva, tegyük fel, hogy van egy legfeljebb d gépet használó, minden járatot teljesítő ütemezésünk. Nézzük ebben egy gép egymás utáni feladatait. Ha ezek a $J_{i_1}, J_{i_2}, \ldots, J_{i_l}$ járatok (ebben a sorrendben), akkor a $(\dagger)$ út éleit a hálózat tartalmazza. Ezen az úton egységnyi folyamot küldhetünk s-ből t-be. Ezt minden gépre elvégezve egy legfeljebb d értékű g folyamot kapunk. A kapacitásfeltételek teljesülnek, mert az így kapott $(\dagger)$ utaknak s és t kivételével nincs közös élük, hiszen egy járat egy géphez tartozik. Az (i,i') élekre kiszabott $g(i,i') \geq 1$ feltételeket is megtartjuk, mert az ütemezésben minden járatnak van gazdája. Megállapíthatjuk immár, hogy g eleget tesz a követelményeknek. □

Feladat: Mutassuk meg, hogy a követhetőségek gráfjának és d-nek az ismeretében $O(m^4)$ lépésben eldönthető, hogy van-e a $J_1, J_2, \ldots, J_m$ járatoknak legfeljebb d géppel való ütemezése. (Az előzőek segítségével alakítsuk át a feladatot maximális folyam meghatározásává; ennek megoldására használjuk Dinic módszerét.)