Algoritmusok és adatszerkezetek II. előadásjegyzet

Élsúlyozott gráfok és algoritmusaik

Ásványi Tibor – asvanyi@inf.elte.hu

2021. december 14.

Tartalomjegyzék

Hivatkozások

[1] Ásványi Tibor, Algoritmusok és adatszerkezetek II.
Útmutatások a tanuláshoz, jelölések, tematika,
fák, gráfok,
mintaillesztés, tömörítés
http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet/

[2] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.,
magyarul: Új Algoritmusok, *Scolar Kiadó*, Budapest, 2003.
ISBN 963 9193 90 9
angolul: Introduction to Algorithms (Third Edititon),
The MIT Press, 2009.

[3] Fekete István, Algoritmusok jegyzet
http://ifekete.web.elte.hu/

[4] Rónyai Lajos – Ivanyos Gábor – Szabó Réka, Algoritmusok,
TypoTEX Kiadó, 1999. ISBN 963 9132 16 0
https://www.tankonyvtar.hu/hu/tartalom/tamop425/2011-0001-526_ronyai_algoritmusok/adatok.html

[5] Tarjan, Robert Endre, Data Structures and Network Algorithms,
CBMS-NSF Regional Conference Series in Applied Mathematics, 1987.

[6] Weiss, Mark Allen, Data Structures and Algorithm Analysis,
Addison-Wesley, 1995, 1997, 2007, 2012, 2013.

[7] Ásványi Tibor, Algoritmusok és adatszerkezetek I. előadásjegyzet (2021)
http://aszt.inf.elte.hu/~asvanyi/ad/ad1jegyzet/ad1jegyzet.pdf

[8] Ásványi Tibor, Detecting negative cycles with Tarjan's breadth-first scanning algorithm (2016)
http://ceur-ws.org/Vol-2046/asvanyi.pdf

1. Élsúlyozott gráfok és ábrázolásaik ([2] 22)

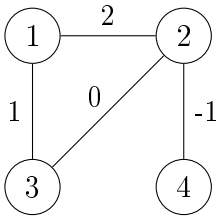
1.1. Definíció. Élsúlyozott gráf *alatt egy* $G = (V, E)$ *gráfot értünk a* $w : E \to \mathbb{R}$ *súlyfüggvénnyel, ahol* V *a csúcsok (vertices) tetszőleges, véges halmaza,* $E \subseteq V \times V \setminus \{(u, u) : u \in V\}$ *az élek (edges) halmaza.*

A $w : E \to \mathbb{R}$ *minden egyes élhez hozzárendeli annak* súlyát, *más néven* hosszát, *illetve* költségét. *(Az* élsúly, élhossz *és* élköltség *elnevezések szinonímák.)*

1.2. Definíció. *Élsúlyozott gráfban tetszőleges út* hossza, *más néven* költsége, *illetve* súlya *az út mentén található élek összsúlya. Hasonlóképpen tetszőleges gráf/fa súlya az élei súlyainak összege.*

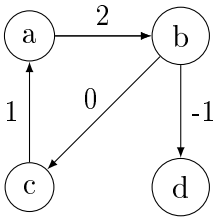
1.1. Grafikus ábrázolás

Az éleket súlyukkal címkézzük.



1 – 2, 2 ; 3, 1.
2 – 3, 0 ; 4, -1.

1. ábra. Ugyanaz az élsúlyozott irányítatlan gráf grafikus (balra) és szöveges (jobbra) ábrázolással.



a → b, 2.
b → c, 0 ; d, -1.
c → a, 1.

2. ábra. Ugyanaz az élsúlyozott irányított gráf grafikus (balra) és szöveges (jobbra) ábrázolással.

1.2. Szöveges ábrázolás

Az irányítatalan gráfoknál „$u - v_{u_1}, w_{u_1}; \cdots ; v_{u_k}, w_{u_k}$." azt jelenti, hogy $(u, v_{u_1}), \cdots , (u, v_{u_k})$ élei a gráfnak, sorban $w(u, v_{u_1}) = w_{u_1}, \cdots , w(u, v_{u_k}) = w_{u_k}$ súlyokkal. (Ld. az 1. ábrát!)

Az irányított gráfoknál pedig „$u \to v_{u_1}, w_{u_1}; \cdots ; v_{u_k}, w_{u_k}$." azt jelenti, hogy a gráfban az u csúcsból az $(u, v_{u_1}), \cdots , (u, v_{u_k})$ irányított élek indulnak ki, most is sorban $w(u, v_{u_1}) = w_{u_1}, \cdots , w(u, v_{u_k}) = w_{u_k}$ súlyokkal. (Ld. a 2. ábrát!)

1.3. Szomszédossági mátrixos (adjacency matrix), más néven csúcsmátrixos reprezentáció

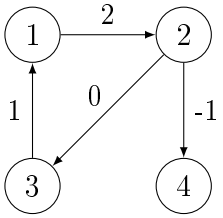
A szomszédossági mátrixos, vagy más néven csúcsmátrixos ábrázolásnál a $G = (V, E)$ gráfot a $w : E \to \mathbb{R}$ súlyfüggvénnyel ($V = \{v_1, \cdots , v_n\}$) egy $A/1 : \mathbb{R}_\infty[n, n]$ mátrix reprezentálja, ahol $n = |V|$ a csúcsok száma, $1..n$ a csúcsok sorszámai, és tetszőleges $i, j \in 1..n$ csúcssorszámokra

$$A[i, j] = w(v_i, v_j) \iff (v_i, v_j) \in E$$

$$A[i, i] = 0$$

$$A[i, j] = \infty \iff (v_i, v_j) \notin E \wedge i \neq j$$

A 3. ábrán látható irányított gráfot például a mellette lévő szomszédossági mátrix reprezentálja.



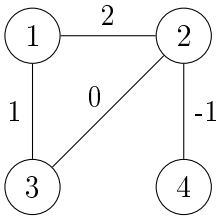
A	1	2	3	4
1	0	2	∞	∞
2	∞	0	0	-1
3	1	∞	0	∞
4	∞	∞	∞	0

3. ábra. Ugyanaz az élsúlyozott, irányított gráf grafikus (balra) és szomszédossági mátrixos (jobbra) ábrázolással.

A főátlóban mindig nullák vannak, mert csak egyszerű gráfokkal foglalkozunk (amelyekben nincsenek hurokélek), és tetszőleges csúcsból önmaga közvetlenül, nulla költségű úton érhető el.

Vegyük észre, hogy irányítatlan esetben a szomszédossági mátrixos reprezentáció mindig szimmetrikus, ui. $(v_i, v_j) \in E$ esetén $(v_j, v_i) = (v_i, v_j) \in E$.

Az 1. ábráról már ismerős irányítatlan gráf csúcsmátrixos ábrázolása a 4. ábrán látható.



A	1	2	3	4
1	0	2	1	∞
2	2	0	0	-1
3	1	0	0	∞
4	∞	-1	∞	0

4. ábra. Ugyanaz az élsúlyozott, irányítatlan gráf grafikus (balra) és szomszédossági mátrixos (jobbra) ábrázolással.

1.4. Szomszédossági listás (adjacency list) reprezentáció

A szomszédossági listás ábrázolás hasonlít a szöveges reprezentációhoz. A $G = (V, E)$ gráfot a $w : E \to \mathbb{R}$ súlyfüggvénnyel ($V = \{v_1, \cdots, v_n\}$) az $A : Edge^*[n]$ pointertömb segítségével ábrázoljuk, ahol az $Edge$ típus a következő.

$Edge$
$+v : \mathbb{N}$
$+w : \mathbb{R}$
$+next : Edge*$

A $next$ és a v attributumok szerepe ugyanaz, mint az élsúlyozatlan gráfoknál, w pedig a megfelelő él súlya. Az élsúlyozatlan gráfokhoz hasonlóan az élsúlyozott gráfoknál is: irányítatlan gráfok esetén minden élet kétszer ábrázolunk, irányított gráfok esetén csak egyszer. Élsúlyozott, irányított gráfra láthatunk példát az 5. ábrán.

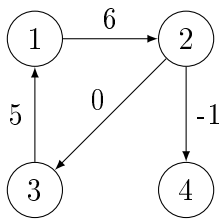
1.5. Élsúlyozott gráfábrázolások tárigénye

1.5.1. Szomszédossági mátrixok

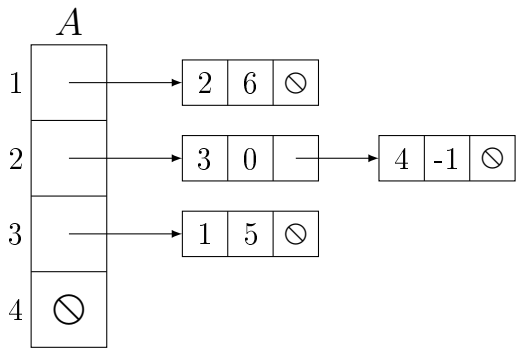
Tárigényük hasonlóan számolható, mint élsúlyozatlan esetben. Feltéve, hogy egy valós számot egy gépi szóban tárolunk, a szomszédossági mátrixos (más néven csúcsmátrixos) ábrázolás tárigénye alapesetben n^2 szó. Irányítatlan gráfoknál, csak az alsóháromszög mátrixot tárolva, $n * (n-1)/2$ szó. Mivel $n * (n-1)/2 \in \Theta(n^2)$, az aszimptotikus tárigény mindkét esetben $\Theta(n^2)$.

1.5.2. Szomszédossági listák

Mindegyik élben eggyel több mező van, mint az élsúlyozatlan esetben. Ez az aszimptotikus tárigényt nyilván nem befolyásolja.



$A[1] \to v = 2$; $A[1] \to w = 6$; $A[1] \to next = \oslash$
$A[2] \to v = 3$; $A[2] \to w = 0$; $A[2] \to next \to v = 4$
; $A[2] \to next \to w = -1$; $A[2] \to next \to next = \oslash$
$A[3] \to v = 1$; $A[3] \to w = 5$; $A[3] \to next = \oslash$
$A[4] = \oslash$

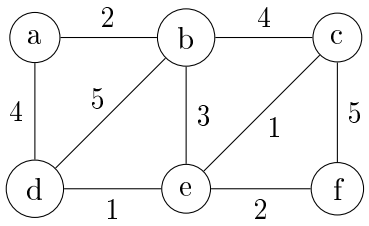


5. ábra. Ugyanaz az élsúlyozott, irányított gráf grafikus (balra) és szomszédossági listás (jobbra) ábrázolással.

1.6. Élsúlyozott gráfok absztrakt osztálya

$\mathcal{G}_w$
$+ V : \mathcal{V}\{\}$ $+ E : \mathcal{E}\{\}$ // $E \subseteq V \times V \setminus \{(u,u) : u \in V\}$ $+ w : E \to \mathbb{R}$ // weights of edges

2. Minimális feszítőfák ([2] 23)



a – b, 2 ; d, 4.
b – c, 4 ; d, 5 ; e, 3.
c – e, 1 ; f, 5.
d – e, 1.
e – f, 2.

6. ábra. Összefüggő, élsúlyozott, irányítatlan gráf. A csúcsok pl. városok, az élek lehetséges összeköttetések, a megépítésük költségeivel. A lehető legkisebb építési költséggel szeretnénk elérni, hogy tetszőleges városból bármelyik másikba el lehessen jutni.

A 6. ábrán megfogalmazott (és még sok más hasonló) feladat megoldása céljából definiáljuk a *minimális feszítőfa* (MST = Minimum Spanning Tree) fogalmát. Ebben a fejezetben tetszőleges összefüggő, irányítatlan, élsúlyozott gráf *minimális feszítőfáját* keressük. (Az élsúlyok negatívak is lehetnek.)

2.1. Definíció. *A $G = (V, E)$ irányítatlan gráf* feszítő erdeje *a $T = (V, F)$ gráf, ha $F \subseteq E$, valamint T (irányítatlan) erdő (azaz T olyan irányítatlan gráf, aminek mindegyik komponense irányítatlan fa, a fák páronként diszjunktak, és együtt éppen lefedik a G csúcshalmazát).*

2.2. Definíció. *A $G = (V, E)$ irányítatlan, összefüggő gráf* feszítőfája *a $T = (V, F)$ gráf, ha $F \subseteq E$, és T (irányítatlan) fa.*

2.3. Definíció. *Amennyiben $G = (V, E)$ élsúlyozott gráf (fa, erdő stb.) a $w : E \to \mathbb{R}$ súlyfüggvénnyel, akkor a G súlya az élei súlyainak összege:*

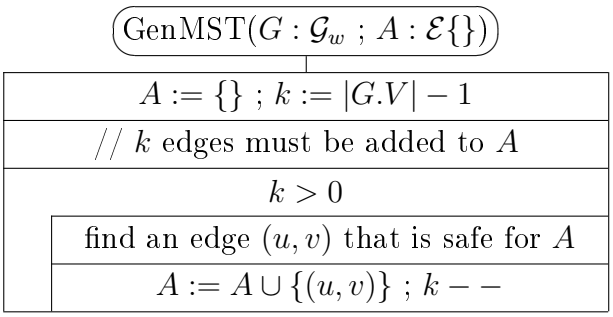
$$w(G) = \sum_{e \in E} w(e)$$

2.4. Definíció. *A G irányítatlan, összefüggő, élsúlyozott gráf* minimális feszítőfája *(*minimum spanning tree: MST*) T, ha T a G feszítőfája, és G bármely T' feszítőfájára $w(T) \leq w(T')$.*

2.1. Egy általános módszer

Az alábbi általános módszer az $A = \{\}$ üres élhalmazból indul, és ezt úgy bővíti újabb és újabb élekkel, hogy A végig a G összefüggő, irányítatlan, élsúlyozott gráf valamelyik minimális feszítőfája élhalmazának a részhalmaza

marad: Éppen az így választott éleket nevezzük az A *élhalmazra nézve biztonságosnak* (*safe for* A). Amikor az élek száma eléri a $|G.V|-1$ értéket, az A szükségszerűen feszítőfa, és így minimális feszítőfa is lesz.



2.5. Definíció. *Tegyük fel, hogy $G = (V, E)$ élsúlyozott, irányítatlan, összefüggő gráf, és $A \subseteq$ a G valamelyik minimális feszítőfája élhalmazának! Ekkor az $(u, v) \in E$ él* biztonságosan hozzávehető *az A élhalmazhoz (*safe for A*), ha $(u, v) \notin A$ és $A \cup \{(u, v)\} \subseteq$ a G valamelyik (az előzővel nem okvetlenül egyező) minimális feszítőfája élhalmazának.*

2.6. Következmény. *Tegyük fel, hogy $G = (V, E)$ élsúlyozott, irányítatlan, összefüggő gráf! Ha egy kezdetben üres A élhalmazt újabb és újabb biztonságosan hozzávehető éllel bővítünk, akkor $|G.V|-1$ bővítés után éppen a G egyik minimális feszítőfáját kapjuk meg.*

A kérdés most az, hogyan tudunk mindig biztonságos élet választani az A élhalmazhoz. Ehhez lesz szükségünk az alábbi fogalmakra és tételre.

2.7. Definíció. *Ha $G = (V, E)$ gráf és $\{\} \subsetneq S \subsetneq V$, akkor a G gráfon $(S, V \setminus S)$ egy* vágás.

2.8. Definíció. *$G = (V, E)$ gráfon az $(u, v) \in E$ él* keresztezi *az $(S, V \setminus S)$* vágást, *ha $(u \in S \wedge v \in V \setminus S) \vee (u \in V \setminus S \wedge v \in S)$.*

2.9. Definíció. *$G = (V, E)$ élsúlyozott gráfon az $(u, v) \in E$* könnyű él *az $(S, V \setminus S)$* vágásban, *ha (u, v) keresztezi a vágást, és $\forall (p, q)$, a vágást keresztező élre $w(u, v) \leq w(p, q)$.*

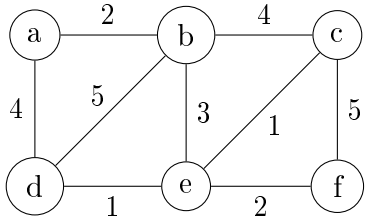
2.10. Definíció. *A $G = (V, E)$ gráfban az $A \subseteq E$ élhalmazt* elkerüli *az $(S, V \setminus S)$* vágás, *ha az A egyetlen éle sem keresztezi a vágást.*

2.11. Tétel. *Ha a $G = (V, E)$ irányítatlan, összefüggő, élsúlyozott gráfon*
(1) A részhalmaza a G valamelyik minimális feszítőfája élhalmazának,
(2) az $(S, V \setminus S)$ vágás elkerüli az A élhalmazt, és
(3) az $(u, v) \in E$ könnyű él *az $(S, V \setminus S)$* vágásban,
$\Longrightarrow$ az (u, v) él biztonságosan hozzávehető az A élhalmazhoz.

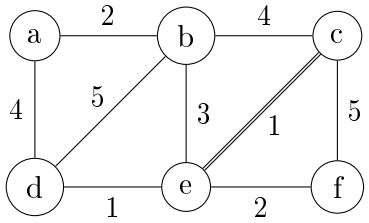
Bizonyítás. $(u,v) \notin A$, ui. (u,v) keresztezi az A-t elkerülő vágást.
Legyen $T = (V, T_E)$ olyan MST, amire $A \subseteq T_E$
a, $(u,v) \in T_E \Rightarrow$ készen vagyunk.
b, $(u,v) \notin T_E$ esetén megjegyezzük, hogy T feszítőfa, tehát T-ben el lehet jutni u-ból v-be, és a T-ben az u-ból a v-be vezető út egyértelmű. Akkor ezen az úton van olyan él, ami keresztezi az $(S, V \setminus S)$ vágást. Legyen (p,q) egy ilyen él! Ekkor $w(p,q) \geq w(u,v) \wedge (p,q) \notin A$. A (p,q) él törlésével a T MST szétesik (azaz két fából álló „feszítő erdő" lesz, az egyik fában az u, a másikban a v csúccsal), de ha az eredményhez az (u,v) élet hozzávesszük, akkor az így adódó T' újra feszítőfa lesz:
$T' := T \setminus (p,q) \cup (u,v)$. Akkor $w(T') = w(T) - w(p,q) + w(u,v) \leq w(T)$ viszont mivel T MST volt, $w(T') \geq w(T)$, azaz $w(T') = w(T)$, és T' is MST kell, hogy legyen. □

Az előbbi tétel segítségével már módszeresen tudunk minimális feszítőfát építeni. Jejöljük a gráfban dupla vonallal az A élhalmaz elemeit!

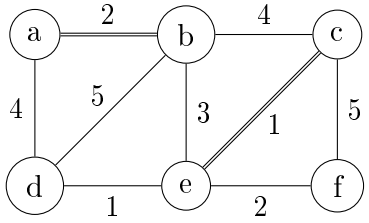
Az alábbi példában minden lépésben választunk egy A-t elkerülő vágást, majd ebben egy könnyű élt, amit hozzáveszünk A-hoz. A hozzávétel eredménye a következő lépés kiinduló pontja.



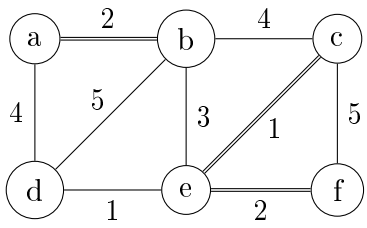
Kezdetben $A = \{\}$, így bármelyik vágás jó. Válasszuk pl. az $(\{a,b,c\}, \{d,e,f\})$ vágást! Ebben (c,e) a könnyű, tehát biztonságos él. Vegyük hozzá A-hoz!



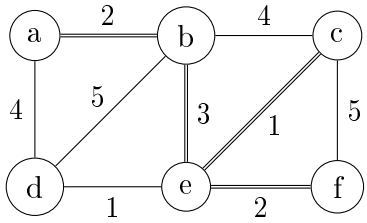
Most $A = \{(c,e)\}$. Ezt elkerüli pl. az $(\{a,f\}, \{b,c,d,e\})$ vágás, amiben (a,b) és (e,f) a könnyű, tehát biztonságos élek. Válasszuk pl. (a,b)-t!



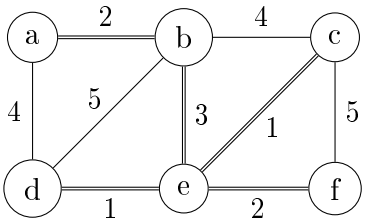
Most $A = \{(a,b),(c,e)\}$. Ezt elkerüli pl. az $(\{a,b,c,d,e\}, \{f\})$ vágás, amiben (e,f) a könnyű, tehát biztonságos él.



Most $A = \{(a,b),(c,e),(e,f)\}$. Ezt elkerüli pl. az $(\{a,b\},\{c,d,e,f\})$ vágás, amiben (b,e) a könnyű, tehát biztonságos él.



$A = \{(a,b),(b,e),(c,e),(e,f)\}$. Ezt egyedül az $(\{a,b,c,e,f\},\{d\})$ vágás kerüli el, amiben (d,e) a könnyű, tehát biztonságos él.



$A = \{(a,b),(b,e),(c,e),(d,e),(e,f)\}$. $|A| = 5 = |G.V|-1$, tehát A egy minimális feszítőfa (MST) élhalmaza.

A fenti módszer még nem tekinthető szigorú értelemben vett algoritmusnak, mert nem adtuk meg, hogyan válasszunk ki az A halmazt elkerülő vágást, és abban könnyű élt. A következő alfejezetekben ismertetendő **Kruskal** és **Prim** algoritmusok éppen ezt teszik, mégpedig

$$O(m * \lg n)\,,$$

tehát nagyon jó maximális műveletigénnyel. (Mint mindig, n a gráf csúcsainak, m pedig az éleinek száma.) Érdekes módon egyik sem adja meg az A-t elkerülő vágást expliciten, a vágásban (az egyik) könnyű élt viszont igen. Ilyen módon, mivel lokálisan optimalizálnak, mindkettő *mohó algoritmus*. A 2.11. tétel szerint viszont a mohóság most kifizetődik, azaz mindkét algoritmus minimális feszítőfát számol.

2.12. Feladat. *Adjon meg olyan 3 csúcsú, összefüggő irányítatlan, élsúlyozott gráfot, amelynek pontosan 2 minimális feszítőfája van. Hány feszítőfája van összesen? Adjon olyan 3 csúcsú gráfot, amelynek 3 minimális feszítőfája van! Létezik olyan 3 csúcsú gráf, aminek háromnál több feszítőfája van? (Gráf alatt, mint mindig, most is egyszerű gráfot értünk.)*

2.2. Kruskal algoritmusa

Kruskal algoritmusa a $G = (V, E)$ gráf éleit a súlyuk (hosszuk) szerint monoton növekvően veszi sorba. Azokat az éleket eldobja, amelyek az A bizonyos éleivel együtt kört képeznének. A többit hozzáveszi A-hoz. Az explicit körkeresés azonban nem lenne hatékony, mert minden egyes e élre bejárást kellene indítani a $(V, A \cup \{e\})$ gráfon. Ehelyett a következő definíción és invariánson alapuló megoldáshoz folyamodunk.

2.13. Definíció. *A $G = (V, E)$ gráf* feszítő erdeje *a (V, A) gráf, ha egymástól diszjunkt fákból, mint komponensekből áll, és $A \subseteq E$. (Két fa egymástól diszjunkt, ha nincs közös csúcsuk [és így közös élük sem].)*

A **Kruskal algoritmus invariánsa,** hogy (V, A) *a $G = (V, E)$ összefüggő, irányítatlan, élsúlyozott gráf feszítő erdeje, és A részhalmaza a G valamelyik minimális feszítőfája élhalmazának.*

Kruskal algoritmusa: $A = \{\}$-val indulunk, ami azt jelenti, hogy a kezdeti feszítő erdő fái a $G = (V, E)$ gráf egycsúcsú fái. A G gráf éleit a súlyuk (hosszuk) szerint monoton növekvően vesszük sorba, és tetszőleges élet pontosan akkor veszünk hozzá A-hoz, ha a (V, A) erdő két fáját köti össze (azaz nem egy fán belül fut, és így zár be egyetlen kört sem). Ezért minden egyes él hozzávételével eggyel csökken az erdő fáinak száma, de továbbra is feszítő erdőt alkotnak. Mivel G összefüggő, bármelyik két fa között van út, így előbb-utóbb összekapcsolódnak és már csak egy T fából áll az erdő, T feszítőfa. A fenti invariáns miatt a T élhalmaza részhalmaza valamelyik M minimális feszítőfa élhalmazának. A G minden feszítőfájának $|V| - 1$ éle van, így a T és M élhalmaza megegyezik. Mindkettő csúcshalmaza V, tehát $T = M$, azaz T minimális feszítőfa.

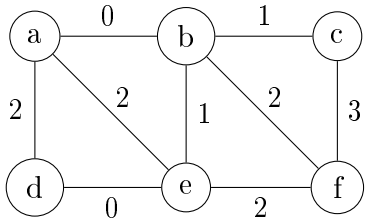
Kérdés még, hogy a fenti invariáns miért igaz. Kezdetben a (V, A) úgy feszítő erdő, hogy $A = \{\}$, tehát A részhalmaza a G bármelyik minimális feszítőfája élhalmazának, azaz az invariáns igaz.

Tekintsünk most az algoritmus futása során egy olyan pillanatot, amikor az invariáns még igaz, és egy újabb e élet készülünk megvizsgálni. Belátjuk, hogy az invariáns az e él feldolgozása után is igaz marad.

Ha e a jelenlegi feszítő erdő valamelyik fáján belül fut, eldobjuk, a feszítő erdő nem változik és az invariáns igaz marad.

Ha e a jelenlegi feszítő erdő két fáját köti össze, legyen S az egyik fa csúcshalmaza, és tekintsük az $(S, V \setminus S)$ vágást, ami nyilván elkerüli A-t. Ekkor e könnyű él a vágásban [mert a G gráf éleit a súlyuk (hosszuk) szerint monoton növekvően vesszük sorba, tehát az aktuális élnél kisebb súlyú éleket már

korábban feldolgoztuk, így ezek vagy benne vannak A-ban, vagy nincsenek benne A-ban, de a (V, A) feszítő erdő egyik fájának két csúcsát kötik össze, ezért eldobtuk őket]. Teljesülnek tehát a 2.11. tétel feltételei. Ennélfogva e biztonságosan hozzávehető A-hoz, és hozzá is vesszük. A fentiek szerint tehát az invariáns ebben az esetben is igaz marad.



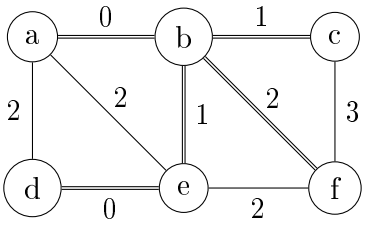
a – b, 0 ; d, 2 ; e, 2.
b – c, 1 ; e, 1 ; f, 2.
c – f, 3.
d – e, 0.
e – f, 2.

7. ábra. Összefüggő, élsúlyozott, irányítatlan gráf

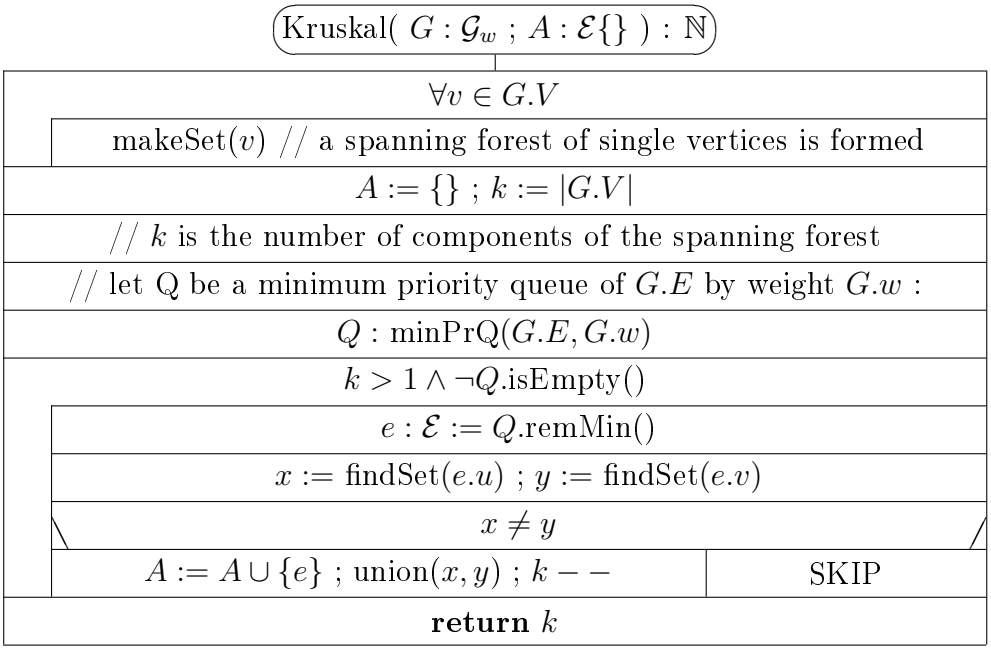
2.14. Példa. *Szemléltessük Kruskal algoritmusát a 7. ábrán látható gráfon!*

Komponensek	él	él súlya	biztonságos?
a, b, c, d, e, f	(a,b)	0	+
ab, c, d, e, f	(d,e)	0	+
ab, c, de, f	(b,c)	1	+
abc, de, f	(b,e)	1	+
abcde, f	(a,d)	2	-
abcde, f	(a,e)	2	-
abcde, f	(b,f)	2	+
abcdef	-	-	-

;



8. ábra. A 7. ábráról ismerős gráfon a minimális feszítőfa éleit dupla vonallal jelöltük.



Mint látható, ez az algoritmus kivételesen „ellenőrzi”, hogy G összefüggő-e. Ha ugyanis összefüggő, $k = 1$ értékkel tér vissza. Ha nem, $k > 1$ lesz.

A műveletigény-számítást azzal a feltételezéssel végezzük el, hogy a makeSet(v) és a union(x, y) eljárások futási ideje $\Theta(1)$, a findSet(v) függvényé pedig $O(\log n)$. E feltételezések jogosságát a 2.2.1. alfejezetben igazoljuk. Feltesszük még, hogy a Q prioritásos sort bináris kupaccal valósítjuk meg. Így, mivel a gráf éleit tároljuk benne, inicializálása $\Theta(m)$, remMin() művelete pedig $O(\log m)$ időt igényel.

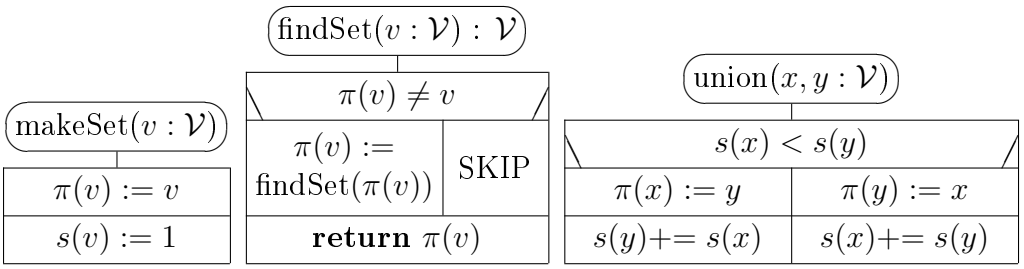
Az első ciklus műveletigénye az előző bekezdés alapján $\Theta(n)$, a két ciklus közti részé pedig $\Theta(m)$. Az $e := Q$.remMin() utasítás $O(\log m)$ futási ideje egyben $O(\log n)$, mivel G összefüggő és irányítatlan, és így $n - 1 \leq m \leq n * (n-1)/2 < n^2$, amiből $(\log n) - 1 < \log(n-1) \leq \log m < \log n^2 = 2 * \log n$, azaz $(\log n) - 1 < \log m < 2 * \log n$, tehát $\log m \in \Theta(\log n)$, ahonnét $\Theta(\log m) = \Theta(\log n)$, következőleg $O(\log m) = O(\log n)$. Az „$x :=$ findSet($e.u$) ; $y :=$ findSet($e.v$)” utasításoké a feltevésünk szerint szintén $O(\log n)$, az elágazásé pedig $\Theta(1)$. A fő ciklus egy iterációja tehát maga is $O(\log n)$ időt igényel, és mivel legfeljebb m-szer iterál, a teljes műveletigénye $O(m * \log n)$. Ezt aszimptotikus értelemben már nem módosítja a fő ciklust megelőző inicializálások nála nagyságrenddel kisebb $\Theta(n + m)$ futási ideje.

2.15. Feladat. *Hogyan tudná kifinomultabban értelmezni azt az esetet, ha a Kruskal algoritmus $k > 1$ értékkel tér vissza? Mit jelent a k értéke általá-*

ban? Tudna-e értelmet tulajdonítani a Kruskal algoritmus által kiszámolt A élhalmaznak, ha végül $k > 1$ értéket ad vissza?

2.16. Feladat. *Implementálja a Kruskal algoritmust az elméleti $O(m * \lg n)$ maximális műveletigény megtartásával!*

2.2.1. A Kruskal algoritmus halmazműveletei



Az irányítatlan feszítőerdő nyilvántartásához egy másik, irányított erdőt is kezelünk. Az irányítatlan feszítőerdő minden egyes (irányítatlan) fájának megfelel az irányított erdő egy (irányított) fája, aminek ugyanaz a csúcshalmaza. Az irányított fákat a gyökerük felé irányítottnak képzelhetjük el: Mindegyik csúcsnak van egy π címkéje, aminek értéke az ő szülője, kivéve az irányított fák gyökércsúcsait; tetszőleges r gyökércsúcsra viszont $\pi(r) = r$, $s(r)$ pedig az r-hez tartozó fa mérete. Így mindegyik irányítatlan fát a neki megfelelő irányított fa gyökércsúcsával azonosítjuk.

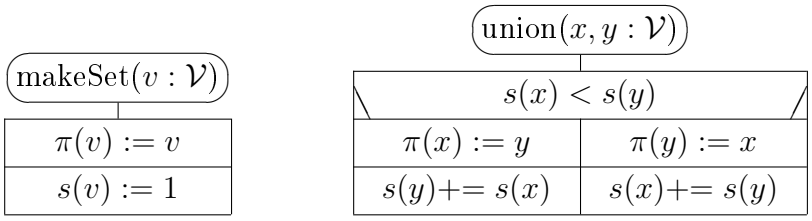
A makeSet(v) művelet a gráf mindegyik v csúcsából egyelemű irányított fát képez.

A findSet(v) függvény megállapítja, hogy a csúcs melyik irányítatlan fában van, azaz megkeresi a csúcsot tartalmazó irányított fa gyökércsúcsát. (Közben a v csúcs és mindegyik őse π mutatóját a gyökércsúcsra állítja, hogy a későbbi findSet-hívások már majd hatékonyabbak legyenek. Arról, hogy ez a heurisztikusnak tűnő „út-rövidítés" tényleges hatékonyságnövekedést hoz magával, az „Új Algoritmusok" [2] könyvben részletes elemzés olvasható. A továbbiakban eltekintünk az „út-rövidítésből" eredő hatékonyság-növekedéstől. Célunkat az így adódó pontatlanabb felső becslésekkel is elérjük.)

Annak megfelelően, hogy a Kruskal algoritmusban az „$x :=$ findSet($e.u$) ; $y :=$ findSet($e.v$)" utasítások az e él két végpontjához tartozó gyökércsúcsokat határozzák meg, a union(x, y) művelet később a megfelelő irányított fák gyökércsúcsait köti össze. A fenti módszerrel a union(x, y) művelet a kisebb méretű irányított fa gyökerét a nagyobb (vagy vele egyenlő méretű) irányított

fa gyökere alá köti be. Belátjuk, hogy ezzel a módszerrel sosem lesz az egyes fák magassága nagyobb, mint a méretük kettes alapú logaritmusa. Ebből pedig közvetlenül következik majd a findSet(v) függvény műveletigényével kapcsolatos állításunk.

2.17. Tulajdonság. *A makeSet(v) és a union(x, y) eljárások hatására létrejövő, gyökércsúcsuk felé irányított fák magassága sosem lesz nagyobb, mint a méretük kettes alapú logaritmusa, azaz, ha r egy ilyen fa gyökércsúcsa, $s(r)$ a mérete és $h(r)$ a magassága, akkor* $\log s(r) \geq h(r)$.



Bizonyítás. A makeSet(v) eljárás egy csúcsú, nulla magasságú fákat hoz létre, és $\log 1 = 0$, tehát a bizonyítandó tulajdonság kezdetben igaz. Elegendő belátni, hogy a union(x, y) eljárás is tartja a fenti egyenlőtlenséget. Legyen r a union(x, y) eljárás hatására létrejövő fa gyökere! Feltehető, hogy az eljáráshívás előtt $\log s(x) \geq h(x) \wedge \log s(y) \geq h(y)$. Azt kell belátnunk, hogy a union(x, y) eljáráshívás után $\log s(r) \geq h(r)$. Feltehető még, hogy $s(x) \geq s(y)$. Ekkor az x gyökércsúcs alá csatoljuk be az y gyökércsúcsot, így $h(r) = max(h(x), h(y) + 1)$. Két eset van.

- $h(x) > h(y) \Rightarrow h(r) = h(x) \wedge s(r) \geq s(x) \Rightarrow \log s(r) \geq \log s(x) \geq h(x) = h(r) \Rightarrow \log s(r) \geq h(r)$.
- $h(x) \leq h(y) \Rightarrow h(r) = h(y) + 1 \wedge s(r) = s(x) + s(y) \geq 2 * s(y) \Rightarrow \log s(r) \geq \log(2 * s(y)) = 1 + \log s(y) \geq 1 + h(y) = h(r) \Rightarrow \log s(r) \geq h(r)$.

□

Most belátjuk a Kruskal algoritmus műveletigény-számításával kapcsolatos feltételezések jogosságát, azaz, hogy a makeSet(v) és a union(x, y) eljárások futási ideje $\Theta(1)$, a findSet(v) függvényé pedig $O(\log n)$.

Jelölje ez utóbbihoz h a v csúcsot tartalmazó irányított fa magasságát, s pedig a méretét! A findSet(v) függvény műveletigénye nyilván $O(h)$. Másrészt a 2.17. tulajdonság alapján $h \leq \log s$. Ezeket az $s \leq n$ egyenlőtlenséggel összevetve a findSet(v) függvény futási ideje durva felső becsléssel $O(\log n)$. A makeSet(v) és a union(x, y) műveleteké pedig $\Theta(1)$, hiszen sem ciklust, sem eljáráshívást nem tartalmaznak. Ezzel a Kruskal algoritmus műveletigény-számításával kapcsolatos feltételezések jogosságát beláttuk.

2.3. Prim algoritmusa

Kijelölünk a $G = (V, E)$ összefüggő, élsúlyozott, irányítatlan gráfban egy tetszőleges $r \in V$ csúcsot. A $T = (\{r\}, \{\})$, egyetlen csúcsból álló fából kiindulva építünk fel egy minimális (V, F) feszítőfát: Minden lépésben a $T = (N, A)$ fához egy újabb biztonságos élet és hozzá tartozó csúcsot adunk.

Ez azt jelenti, hogy a $T = (N, A)$ fa végig ennek a minimális feszítőfának a része marad, azaz végig igaz az $N \subseteq V \wedge A \subseteq F$ invariáns. Ehhez mindegyik lépésben egy könnyű élet választunk ki az $(N, V \setminus N)$ vágásban. (Ld. a 2.11. tételt!)

A megfelelő könnyű él hatékony meghatározása céljából egy Q min-prioritásos sorban tartjuk nyilván a $V \setminus N$ csúcshalmazt. Mindegyik $u \in V \setminus N$ csúcshoz tartozik egy $c(u)$ és egy $p(u)$ címke. Ha van él az u csúcs és a T fa N csúcshalmaza között, azaz az $(N, V \setminus N)$ vágásban, akkor a $(p(u), u)$ a gráf egyik éle a vágásban, $c(u) = w(p(u), u)$, és minden olyan x csúcsra, amelyre (x, u) vágásbeli él, $w(x, u) \geq w(p(u), u)$.

Ez azt jelenti, hogy $(p(u), u)$ minimális súlyú él azok között, amelyek az u csúcsot a T fához kapcsolják. Ha nincs él a u csúcs és a T fa között, akkor $p(u) = \oslash \wedge c(u) = \infty$.

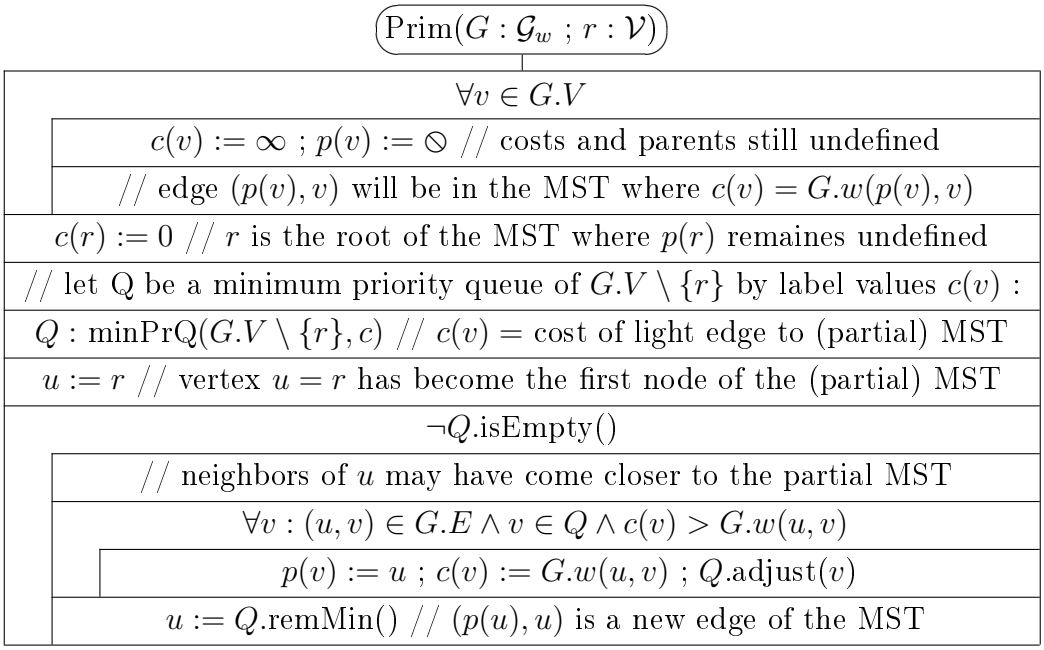
A Q min-prioritásos sorban a csúcsokat a $c(u)$ értékeik szerint tartjuk nyilván. Az $u := Q$.remMin() utasítás tehát egy olyan u csúcsot vesz ki Q-ból, amire a $(p(u), u)$ könnyű él az $(N, V \setminus N)$ vágásban. Ezt az élt így a 2.11. tétel szerint biztonságosan adjuk hozzá a T fához, azaz T továbbra is kiegészíthető minimális feszítőfává, ha még nem az.

Így az eredetileg egyetlen csúcsból álló T fa, $n-1$ db ilyen $(p(u), u)$ él hozzáadásával MST (minimális feszítőfa) lesz.

Amikor u bekerül a T fába, a Q-beli v szomszédai közelebb kerülhetnek a fához, így ezeket meg kell vizsgálni, és ha némelyikre $w(u, v) < c(v)$, akkor a $c(v) := w(u, v)$ és a $p(v) := u$ utasításokkal aktualizálni kell a v csúcs címkéit.

Ilyen módon a $c(v)$ érték csökkenése miatt a Q helyreállítása is szükségessé válhat. Ha például Q reprezentációja egy minimum kupac, a v csúcsot lehet, hogy meg kell cserélni a szülőjével, esetleg újra és újra, mígnem a szülőjének c értéke $\leq c(v)$ lesz, vagy v fölér kupac gyökerébe.

Az alábbi algoritmusban valaki hiányolhatja a T fa explicit reprezentációját. Világos, hogy implicite $N = V \setminus Q$, T éleit pedig az $N \setminus \{r\}$-beli x csúcsok és $p(x)$ címkéik segítségével $(p(x), x)$ alakban definiálhatjuk.



A Prim algoritmus műveletigénye: Feltételezzük, hogy Q reprezentációja bináris minimum kupac, $n = |G.V| \wedge m = |G.E|$.

Az első ciklus futási ideje $\Theta(n)$. Az első és a második ciklus közti rész műveletigényét a Q kupac inicializálása határorozza meg, amire így szintén $\Theta(n)$ adódik.

A második ciklus mindegyik iterációja kiveszi a Q legkisebb c-értékű elemét. Ez tehát $n-1$-szer fut le, és a belső ciklustól eltekintve mindegyik iterációja $O(\log n)$ időt igényel, ami összesen $O(n * \log n)$.

Ehhez hozá kell még adni a belső ciklus futási idejét, ami a gráf mindegyik élére legfeljebb kétszer fog lefutni, hiszen mindegyik élet mindkét végpontja felől megtaláljuk, kivéve azokat, amelyek egyik vépontja a Q-ban utoljára megmaradt csúcs. (A ciklusfejben a $\forall v : (u, v) \in G.E$ rész után következő $v \in Q \wedge c(v) > G.w(u, v)$ szűrő feltétel valójában egy implicit, beágyazott elágazás feltétel része.) Itt a Q.adjust(v) utasítás $O(\log n)$ műveletigénye aszimptotikusan domináns a többi $\Theta(1)$-es futási idő felett. A belső ciklus tehát, összes végrehajtását tekintve is biztosan lefut $O(m * \log n)$ idő alatt.

A fenti részeredményeket összeadva a Prim algoritmus teljes futási idejére $\Theta(n) + \Theta(n) + O(n * \log n) + O(m * \log n) = O((n+m) * \log n)$ adódik. Mivel a gráf összefüggő, $m \geq n-1$, így végül

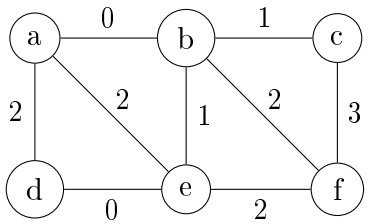
$$MT_{Prim}(n, m) \in O(m * \log n).$$

2.18. Feladat. *Implementálja a Prim algoritmust szomszédossági mátrixos gráfábrázolás esetén! A prioritásos sort rendezetlen tömbbel reprezentálja!*

Mekkora lesz a műveletigény? Lehet-e az aszimptotikus műveletigényen javítani a prioritásos sor kifinomultabb megvalósításával?

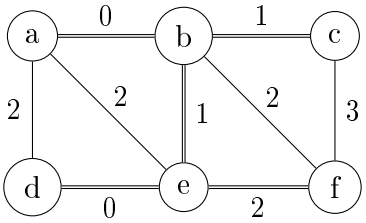
2.19. Feladat. *Implementálja a Prim algoritmust szomszédossági listás gráfábrázolás esetén! Ügyeljen arra, hogy a prioritásos sor megfelelő megvalósításával biztosítani kell az elméleti $O(m * \lg n)$ műveletigényt!*

Az alábbi, már ismerős gráfon szemléltetjük a Prim algoritmus működését, a **d** csúcsból indítva.



a – b, 0 ; d, 2 ; e, 2.
b – c, 1 ; e, 1 ; f, 2.
c – f, 3.
d – e, 0.
e – f, 2.

c értékek Q-ban						minimális feszítő-fába:	p címkék változásai					
a	b	c	d	e	f		a	b	c	d	e	f
∞	∞	∞	0	∞	∞		$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$
2	∞	∞		0	∞	d	d				d	
2	1	∞			2	e		e				e
0		1			2	b	b		b			
		1			2	a						
					2	c						
						f						
0	1	1	0	0	2	eredmény	b	e	b	$\oslash$	d	e



A Kruskal algoritmus eredményéhez képest most a gráf másik minimális feszítőfáját kaptuk meg.

3. Legrövidebb utak egy forrásból([2] 24 ; [5, 8])

Feladat: A $G : \mathcal{G}_w$ gráf tetszőlegesen rögzített s start csúcsából (source vertex) legrövidebb, azaz optimális utat keresünk mindegyik, az s-ből G-ben elérhető csúcsba.

A most következő algoritmusokban az irányítatlan gráfokat olyan irányított gráfoknak tekintjük, ahol a gráf tetszőleges (u, v) élével együtt (v, u) is éle a gráfnak, és $w(u, v) = w(v, u)$.

A feladat pontosan akkor oldható meg, ha G-ben nem létezik s-ből elérhető negatív kör, azaz olyan kör, amely mentén az élsúlyok összege negatív.

Mivel az irányítatlan gráfokat ebben a fejezetben speciális irányított gráfoknak tekintjük, ez a feltétel irányítatlan gráfok esetében azt jelenti, hogy a feladatot akkor tudjuk megoldani, ha nincs a gráfban s-ből elérhető negatív él. (Ha pl. az irányítatlan gráfban (u, v) s-ből elérhető negatív él, akkor a fenti egyszerűsítés miatt $\langle u, v, u\rangle$ egy s-ből elérhető negatív kör.)

Amennyiben a feladat megoldható, mindegyik $v \in G.V\backslash\{s\}$ csúcsra két lehetőség van:
- Ha létezik út s-ből v-be, $d(v)$-ben az optimális út hosszát szeretnénk megkapni, $\pi(v)$-ben pedig egy ilyen optimális úton a v csúcs közvetlen megelőzőjét, azaz szülőjét.
- Ha nem létezik út s-ből v-be, a $d(v) = \infty$ és $\pi(v) = \oslash$ értékeket szeretnénk kapni.

Az s csúcsra a helyes eredmény $d(s) = 0$ és $\pi(s) = \oslash$, mivel az optimális út csak az s csúcsból áll. (Ez már az alábbi algoritmusok inicializálása után teljesül, és végig igaz is marad.)

A legrövidebb utakat kereső algoritmusok futása során az optimális utakat fokozatosan közelítjük. Jelölje $s \rightsquigarrow v$ az adott pillanatig kiszámolt s-ből v-be vezető legrövidebb utat! Az alábbi algoritmusok közös, az inicializálásuk után már teljesülő invariánsa, hogy $d(v)$ ennek a hossza, $\pi(v)$ pedig ($v \neq s$ esetén) ezen az úton a v csúcs közvetlen megelőzője. Ha még nem számoltunk ki $s \rightsquigarrow v$ utat, akkor végtelen hosszúnak tekintjük, azaz $d(v) = \infty$ és $\pi(v) = \oslash$.

Az optimális utak fokozatos közelítéséhez a gráf (u, v) éleit szisztematikusan vizsgáljuk, és ha $s \rightsquigarrow u \to v$ rövidebb mint $s \rightsquigarrow v$, akkor az előbbi lesz az új $s \rightsquigarrow v$. Kód szinten ez azt jelenti, hogy végrehajtjuk az alábbi elágazást, amit *közelítésnek (relaxation)* nevezünk. (Az egyes algoritmusok specialitásai miatt a közelítés egyéb utasításokat is tartalmazhat.)

$d(v) > d(u) + G.w(u, v)$	
$\pi(v) := u$; $d(v) := d(u) + G.w(u, v)$	SKIP

Az alábbi algoritmusok közös vonása még, hogy ismételten kiválasztanak és ki is vesznek az ún. *feldolgozandó* csúcsok halmazából egy csúcsot, és a csúcsból kimenő összes élre végeznek közelítést. Ezeket a közelítéseket együtt a csúcs *kiterjesztésének* nevezzük, míg a csúcs kiválasztását (és kivételét a feldolgozandók közül) a kiterjesztésével együtt a *feldolgozásának* nevezzük.

Főbb különbségeik a következők:

- A Dijkstra algoritmus kezdetben kiválasztja kiterjesztésre s-et, a feldolgozandó csúcsok halmazát pedig $G.V \setminus \{s\}$-sel inicializálja. Először tehát s-et terjeszti ki, majd a feldolgozandók közül mindig egy olyan csúcsot dolgoz fel, amibe az adott időpontig a többihez képest legrövidebb utat talált. Így ez egy *mohó algoritmus*, mert lokálisan optimalizál, azaz mindig a legígéretesebb csúcsot dolgozza fel. Kiderül, hogy így mindig olyan csúcsot terjeszt ki, amibe már eddig optimális utat talált, és egy csúcsot sem kell kétszer kiterjesztenie.

- A DAGshP algoritmus először meghatározza az s-ből elérhető csúcsokat, és egyúttal sorbarendezi ezeket olyan módon, ami garantálja, hogy az adott sorrend szerint feldolgozva őket, mindegyik kiválasztásának az időpontjában már megvan bele az optimális út. Ilyen módon ez is csak egyszer terjeszti ki, és csak az s-ből elérhető csúcsokat.

- Az előbbi két algoritmusnak speciális előfeltételei vannak. Egyik sem tudja a lehető legáltalánosabb esetben megoldani a *legrövidebb utak egy forrásból* problémát. A Dijkstra algoritmus elégséges előfeltétele, hogy a gráfban ne legyen s-ből elérhető negatív súlyú él, míg a DAGshP-é, hogy a gráf s-ből elérhető része DAG legyen. Cserébe mindkét algoritmus a legrosszabb esetben is meglehetősen hatékony. (DAGshP: $\Theta(n+m)$, Dijkstra: $O(n+m) * \log n$.)

- Ezekkel szemben a QBF algoritmus a lehető legáltalánosabb esetben oldja meg a feladatot. Szükséges és elégséges előfeltétele tehát, hogy a gráfban ne legyen az s-ből elérhető negatív kör. Cserébe csak $O(n*m)$ műveletigényt tudunk garantálni, mert ugyanazt a csúcsot többször is feldolgozhatja. Szerencsére átlagos esetben ennél sokkal hatékonyabb, implementációikat összehasonlítva sok nemnegatív élsúlyú gráfon a Dijkstra algoritmusnál is gyorsabb (amihez hozzá kell tenni, hogy talán még gyakrabban ez fordítva van).

- Mivel a DAGshP és a QBF algoritmusok előfeltétele nemtriviális, ezeknél így az algoritmus része az előfeltételének az ellenőrzése, sőt, a

nem megfelelő bemenetek visszautasítása is oly módon, hogy mindkettő megad egy olyan kört a gráfban, ami miatt nem tudja a feladatot megoldani. (A QBF ebben az esetben negatív kört ad meg.)

3.1. Dijkstra algoritmusa

Előfeltétel: A $G : \mathcal{G}_w$ gráfban $\forall (u,v) \in G.E$-re $G.w(u,v) \geq 0$, azaz a gráf mindegyik élének élsúlya nemnegatív.

Ebben az esetben nem lehet a gráfban negatív kör, tehát a *legrövidebb utak egy forrásból* feladat megoldható. Ez is

Dijkstra($G : \mathcal{G}_w$; $s : \mathcal{V}$)

- $\forall v \in G.V$
  - $d(v) := \infty$; $\pi(v) := \oslash$ // distances are still ∞, parents undefined
  - // $\pi(v)$ = parent of v on $s \rightsquigarrow v$ where $d(v) = w(s \rightsquigarrow v)$
- $d(s) := 0$ // s is the root of the shortest-path tree
- // let Q be a minimum priority queue of $G.V \backslash \{s\}$ by label values $d(v)$:
- Q : minPrQ($G.V \backslash \{s\}, d$)
- $u := s$ // going to calculate shortest paths form s to other vertices
- $d(u) < \infty \wedge \neg Q$.isEmpty()
  - // check, if $s \rightsquigarrow u \rightarrow v$ is shorter than $s \rightsquigarrow v$ before
  - $\forall v : (u,v) \in G.E \wedge d(v) > d(u) + G.w(u,v)$
    - $\pi(v) := u$; $d(v) := d(u) + G.w(u,v)$; Q.adjust(v)
  - $u := Q$.remMin() // $s \rightsquigarrow u$ is optimal now, if it exists

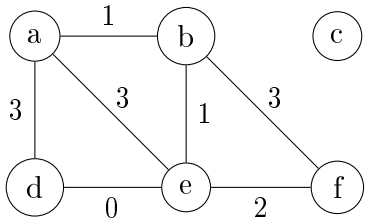
$MT_{Dijkstra}(n,m) \in O((n+m) * \lg n)$ a Prim algoritmusnál már bemutatott módon adódik, tekintettel a két algoritmus közötti meglepő hasonlóságra.

Másrészt $mT_{Dijkstra}(n,m) \in \Theta(n)$, mivel a 2. ciklus előtti rész műveletigénye már $\Theta(n)$, amihez csak a második ciklus egyetlen iterációja és a belső ciklus nullaszori iterációja adódik hozzá, amennyiben nincs a gráfban s-nek rákövetkezője. Ebben az esetben a második ciklus műveletigénye $O(\log n)$ (pontosabban $\Theta(1)$), ami nagysárenddel kisebb, mint $\Theta(n)$.

3.1. Feladat. *Implementálja a Dijkstra algoritmust szomszédossági mátrixos gráfábrázolás esetén! A prioritásos sort rendezetlen tömbbel reprezentálja! Mekkora lesz a műveletigény? Lehet-e az aszimptotikus műveletigényen javítani a prioritásos sor kifinomultabb megvalósításával?*

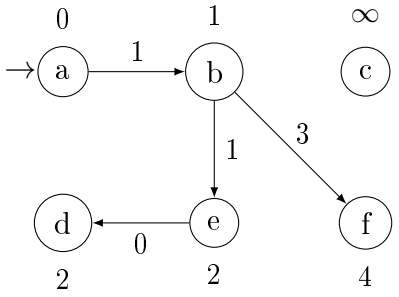
3.2. Feladat. *Implementálja a Dijkstra algoritmust szomszédossági listás gráfábrázolás esetén! Ügyeljen arra, hogy a prioritásos sor megfelelő megvalósításával biztosítani lehet az elméleti $O((m+n) * \lg n)$ műveletigényt!*

Az alábbi gráfon szemléltetjük a Dijkstra algoritmus működését, az **a** csúcsból indítva. A Dijkstra algoritmusnál a *ki nem terjesztett* csúcsokat röviden *nyílt* csúcsoknak nevezzük.



a – b, 1 ; d, 3 ; e, 3.
b – e, 1 ; f, 3.
c.
d – e, 0.
e – f, 2.

nyílt csúcsok d értékei						kiter-	π címkék változásai					
a	b	c	d	e	f	jesztett	a	b	c	d	e	f
0	∞	∞	∞	∞	∞	csúcs : d	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$
	1	∞	3	3	∞	a : 0		a		a	a	
		∞	3	2	4	b : 1					b	b
		∞	2		4	e : 2				e		
		∞			4	d : 2						
		∞				f : 4						
0	1	∞	2	2	4	eredmény	$\oslash$	a	$\oslash$	e	b	b



A legrövidebb utak fája $s =$ a esetén. Az s-ből elérhetetlen csúcsot vagy csúcsokat is feltüntetjük, ∞ d értékkel.

3.3. Tétel. *Amikor az u csúcsot kiválasztjuk kiterjesztésre (először az $u := s$, később az $u := Q.remMin()$ utasítással), akkor u-ba már optimális utat találtunk, feltéve, hogy $d(u) < \infty$.*

Bizonyítás. $u = s$-re nyilván igaz az állítás. Tegyük fel indirekt módon, hogy nem mindegyik csúcsra igaz!

Eszerint van egy olyan v csúcs, amire az algoritmus futása során először lesz igaz, hogy kiválasztjuk kiterjesztésre, de $w(s \overset{opt}{\rightsquigarrow} v) < d(v) < \infty$, ahol $s \overset{opt}{\rightsquigarrow} v$ egy s-ből v-be vezető optimális út, $w(s \overset{opt}{\rightsquigarrow} v)$ pedig ennek a hossza. Világos, hogy $v \neq s$. Továbbá, az $s \overset{opt}{\rightsquigarrow} v$ úton az s csúcsot már kiterjesztettük, a v csúcsot viszont még nem. Van tehát az $s \overset{opt}{\rightsquigarrow} v$ úton egy első csúcs, amit még nem terjesztettünk ki.

Legyen t az $s \overset{opt}{\rightsquigarrow} v$ úton az első csúcs, amit még nem terjesztettünk ki! Szelméletesen: $s \overset{opt}{\rightsquigarrow} t \overset{opt}{\rightsquigarrow} v$. Mivel s-et már kiterjesztettük, $t \neq s$. Továbbá, az $s \overset{opt}{\rightsquigarrow} t$ úton t közvetlen x megelőzőjét is kiterjesztettük már, és az x kiterjesztésekor még teljesült, hogy $d(x) = w(s \overset{opt}{\rightsquigarrow} x)$. Akkor viszont az $x \to t$ él feldolgozása óta már $d(t) = w(s \overset{opt}{\rightsquigarrow} t)$ is teljesül. Mivel a $t \overset{opt}{\rightsquigarrow} v$ úton minden él súlya (azaz hossza) nemnegatív, és t az $s \overset{opt}{\rightsquigarrow} v$ úton van, azaz $s \overset{opt}{\rightsquigarrow} t \overset{opt}{\rightsquigarrow} v$, szükségszerűen $w(s \overset{opt}{\rightsquigarrow} t) \leq w(s \overset{opt}{\rightsquigarrow} v)$.

A fentieket összegezve $d(t) = w(s \overset{opt}{\rightsquigarrow} t) \leq w(s \overset{opt}{\rightsquigarrow} v) < d(v) < \infty$, azaz $d(t) < d(v)$, ami ellentmond az indirekt feltételezésnek, ami szerint a v csúcsot választottuk kiterjesztésre. □

3.4. Tétel. *Ha a kiterjesztésre kiválasztott u csúcsra $d(u) = \infty$, akkor $Q \cup \{u\}$ egyetlen eleme sem érhető már el az s csúcsból.*

Bizonyítás. Nyilván u az első és egyetlen olyan csúcs, amit $d(u) = \infty$ értékkel választottunk ki, mert ezzel megáll a fő ciklus. A korábban kiterjesztésre kiválasztott x csúcsokra tehát $d(x) < \infty$, és ezekbe az előző tétel szerint már optimális utat találtunk.

Tegyük fel indirekt módon, hogy $Q \cup \{u\}$-nak van olyan v eleme, amely elérhető s-ből! Ekkor létezik $s \overset{opt}{\rightsquigarrow} v$ is, amin kell lennie egy első t csúcsnak, amely eleme $Q \cup \{u\}$-nak, de az előző tétel bizonyítása szerint erre már $d(t) = w(s \overset{opt}{\rightsquigarrow} t) < \infty = d(u)$, tehát $d(t) < d(u)$, ami ellentmond annak, hogy az u csúcsot választottuk ki kiterjesztésre. □

3.5. Következmény. *Amíg tehát a kiterjesztésre kiválasztott u csúcsra $d(u) < \infty$, addig olyan csúcsot választunk ki, amelyikbe már optimális utat találtunk.*

Ha $d(u) < \infty$ mindegyik kiválasztott csúcsra igaz, akkor és csak akkor a fenti algoritmus 2. ciklusa $n-1$ iterációval kiüríti Q-t, és megáll, miután a gráf mindegyik csúcsába optimális utat talált.

Ha pedig egyszer a kiterjesztésre kiválasztott u csúcsra már $d(u) = \infty$, akkor $Q \cup \{u\}$ egyetlen eleme sem érhető már el az s csúcsból, és megállhatunk: mindegyik d-értéke végtelen, π-érteke pedig $\oslash$, míg a korábban, véges

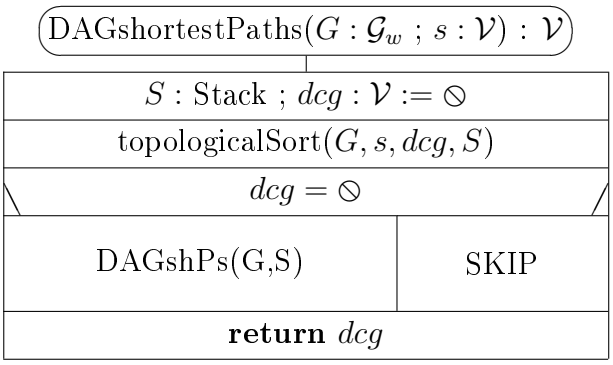
d-értékkel kiterjesztett csúcsok éppen azok, amelyek elérhetők s-ből, és ezekbe találtunk is optimális utat.

3.2. DAG legrövidebb utak egy forrásból (DAGshP)

Előfeltétel: A $G : \mathcal{G}_w$ gráf irányított, és a gráfban nincs s-ből elérhető irányított kör.

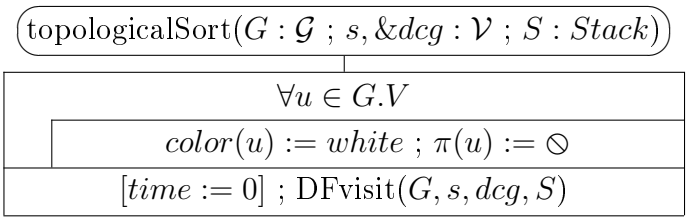
Ebben az esetben nincs a gráfban s-ből elérhető negatív kör sem, így a *legrövidebb utak egy forrásból* feladat megoldható.

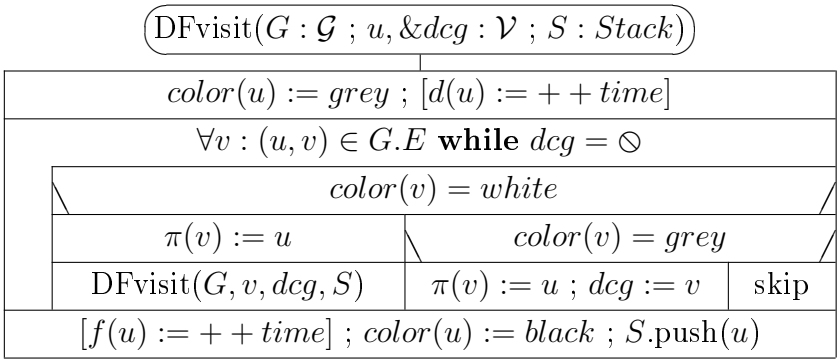
Ez az algoritmus ellenőrzi az előfeltételét. Ha teljesül, a DAGshortestPaths() függvény $\oslash$ értékkel tér vissza. Különben megtalál egy irányított kört, aminek egyik csúcsával tér vissza. Ebből indulva a π címkék mentén a kör fordított irányban bejárható.

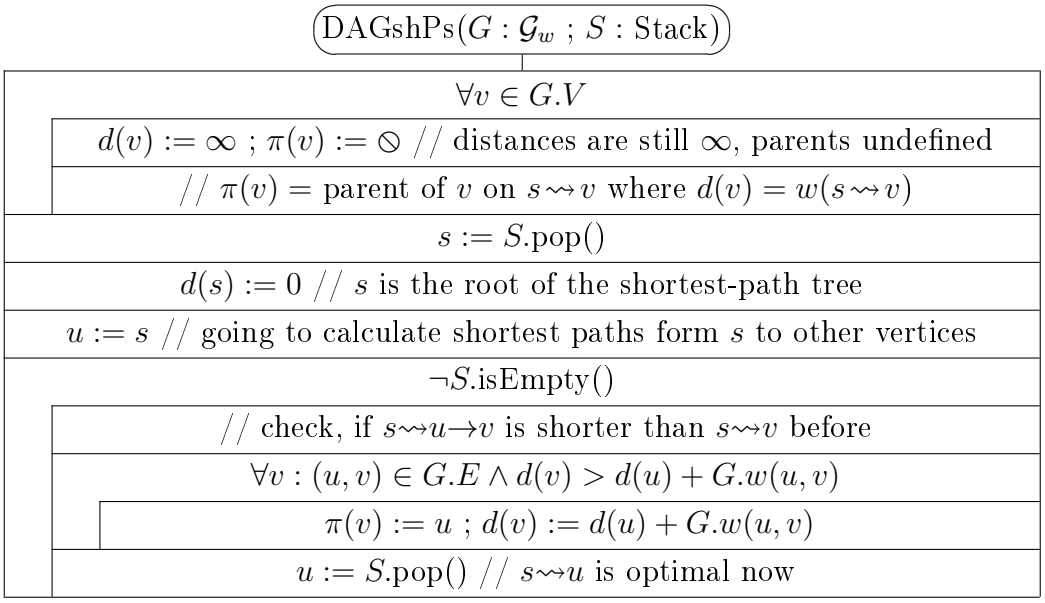


A topologikus rendezés csak az s-ből elérhető részgráfot próbálja topologikusan rendezni.

A *time* nevű változóra – amit az egyszerűség kedvéért globálisnak képzelünk – és a hozzá kapcsolódó utasításokra csak a topologikus rendezés szemléltetésének megkönnyítése végett van szükségünk. Ezek az implmentációkból elhagyhatók, ezért a vonatkozó utasításokat szögletes zárójelbe tettük.







$$MT(n, m) \in \Theta(n + m) \quad \wedge \quad mT(n, m) \in \Theta(n)$$

3.6. Feladat. *Mikor lesz a legkisebb az algoritmus futási ideje? Mikor lesz a legnagyobb? Miért?*

3.7. Tétel. *A legrövidebb utak egy forrásból algoritmus az s-ből elérhető csúcsokba optimális utat talál. Az s-ből el nem érhető x csúcsokra $d(x) = \infty$ és $\pi(x) = \oslash$ lesz.*

Bizonyítás. A kis s-ből indított részleges topologikus rendezés pontosan az s-ből elérhető csúcsokat teszi a nagy S verembe, a megfelelő sorrendben, és így pontosan ezeket a csúcsokat terjesztjük ki, a topologikus sorrendnek megfelelően. A többi x csúcsra az inicalizálás eredményeként $d(x) = \infty$ és

$\pi(x) = \oslash$ marad, ui. ha egy csúcs nem érhető el s-ből, akkor egyetlen G-beli megelőzője sem.

Elegendő belátni, hogy amikor egy u csúcsot kiválasztjuk kiterjesztésre (először az $u := s$, később az $u := S.\text{pop}()$ utasítással), akkor u-ba már optimális utat találtunk.

Az előbbi állítás $u = s$ esetben nyilván igaz. Tegyük fel, hogy adott $v \neq s$ csúcs kiválasztása előtt mindegyik kiterjesztésre kiválasztott csúcsra igaz volt. Mivel a v csúcsnak topologikus sorrend szerinti és így G-beli megelőzőit is a v kiválasztásának pillanatában már kiterjesztettük, a v csúcsnak adott $s \overset{opt}{\rightsquigarrow} v$ úton vett közvetlen u megelőzőjét is, mégpedig úgy, hogy u-ba a kiterjesztésekor már optimális utat találtunk, és így legkésőbb az $u \to v$ él feldolgozásakor v-be is optimális utat találtunk már. □

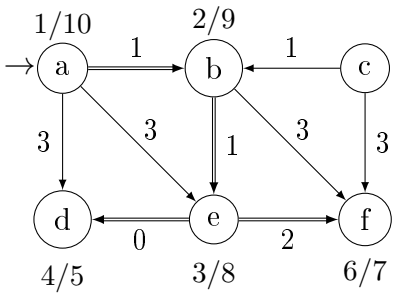
3.8. Feladat. *Implementálja a* DAG legrövidebb utak egy forrásból *algoritmust szomszédossági mátrixos gráfábrázolás esetén!*

Mekkora lesz a műveletigény? Tartható-e az elméleti maximális, illetve minimális műveletigény ezzel az ábrázolással?

3.9. Feladat. *Implementálja a* DAG legrövidebb utak egy forrásból *algoritmust szomszédossági listás gráfábrázolás esetén!*

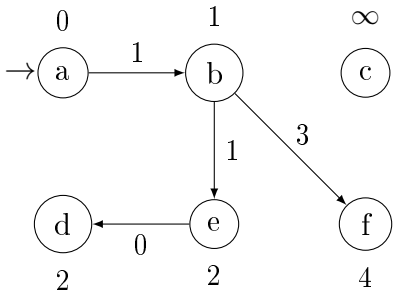
Mekkora lesz a műveletigény? Tartható-e az elméleti maximális, illetve minimális műveletigény ezzel az ábrázolással?

A következő oldalon látható gráfon szemléltetjük a *DAG legrövidebb utak egy forrásból* algoritmust, ahol $s = \text{a}$. Először topologikusan rendezzük az s-ből elérhető részgráfot. (A dupla nyilak a mélységi fa *fa-éleit* jelölik.) Ezután – a szokásos inicializálásokat követően – a csúcsokat a topologikus sorrendjüknek (S) megfelelően terjesztjük ki.



a → b, 1 ; d, 3 ; e, 3.
b → e, 1 ; f, 3.
c → b, 1 ; f, 3.
e → d, 0 ; f, 2.
$S = \langle$ a, b, e, f, d $\rangle$

d értékek változásai						kiter-	π címkék változásai					
a	b	c	d	e	f	jesztett	a	b	c	d	e	f
0	∞	∞	∞	∞	∞	csúcs : d	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$	$\oslash$
	1		3	3		a : 0		a		a	a	
				2	4	b : 1					b	b
			2			e : 2				e		
						f : 4						
0	1	∞	2	2	4	eredmény	$\oslash$	a	$\oslash$	e	b	b



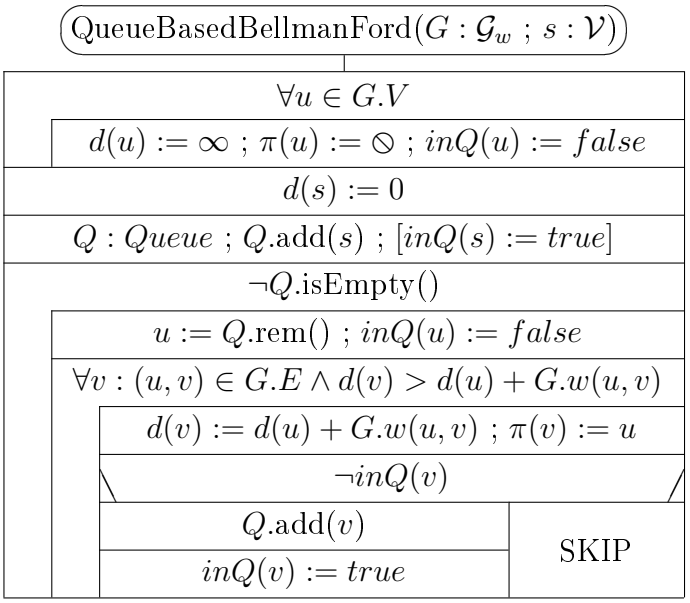
A legrövidebb utak fája $s =$ a esetén. Az s-ből elérhetetlen csúcsot vagy csúcsokat is feltüntetjük, ∞ d értékkel.

3.3. Sor-alapú (Queue-based) Bellman-Ford algoritmus (QBF)

Előfeltétel: Nincs az s-ből elérhető negatív kör. (Irányított gráf esetén lehetnek negatív élsúlyok.)

A *Sor-alapú Bellman-Ford algoritmus (QBF)* (tudományos elemzés és negatív kör vizsgálat nélkül) az informatikai folklórból származik. Legjobb tudásunk szerint először Tarján Róbert Endre amerikai matematikus[1] elemezte és publikálta, *Breadth-first Scanning* néven [5].

A QBF hasonlít a szélességi kereséshez, de az utak hosszát a Dijkstra, a DAGshP (és a Bellman-Ford [2, 3]) algoritmushoz hasonlóan, mint az út menti élsúlyok összegét határozza meg. Kezdetben csak az s start (forrás) csúcsot teszi a sorba. A BFS-hez hasonlóan a szokásos inicializálások után, a fő ciklusban mindig kiveszi a sor első elemét és kiterjeszti, de most mindegyik közvetlen rákövetkezőjére meg is vizsgálja, nem talált-e bele rövidebb utat mint eddig, és ha igen, megfelelően módosítja a d és a π értékét. Ha a módosított d és π értékű csúcs pillanatnyilag nincs benne a sorban, beteszi a sor végére.



Mivel a sor (Queue) adattípusra nincs „$\in$” műveletünk, vagy *átlátszó sor* típust kellene bevezetnünk és megvalósítanunk, vagy – mint itt is – minden v csúcsra értelmezünk egy $inQ(v)$ logikai értékű címkét, ami pontosan akkor

[1] Tarján professzor úr szülei magyar származásúak.

lesz igaz, amikor v eleme a sornak. (Implementációs szinten – feltéve, hogy a csúcsokat 1-től n-ig sorszámoztuk – az $inQ(v)$ címkéket reprezentálhatjuk pl. az $inQ/1 : \mathbb{B}[n]$ tömbbel. Ez a tömb persze az *átlátszó sor* típus része is lehetne.)

Ha nincs a gráfban s-ből elérhető negatív kör, az előző bekezdésben ismertetett algoritmus, miután minden s-ből elérhető v csúcsra meghatározott egy $s \overset{opt}{\rightsquigarrow} v$ utat, üres sorral megáll. Ha a gráf tartalmaz az s-ből elérhető negatív kört, akkor a kiterjesztések egy ilyen mentén „körbejárnak", a sor sosem ürül ki, és az előbbi algoritmus végtelen ciklusba kerül.

3.3.1. A negatív körök kezelése

Arra az esetre, ha nem eleve kizárt, hogy a gráf s-ből elérhető negatív kört tartalmaz, egy ilyen meghatározására Tarján több módszert is kidolgozott [5]. Itt e jegyzet szerzőjének egy viszonylag egyszerű, de könnyen ellenőrizhető kritériumát fogjuk használni [8].

Bevezetjük a gráf v csúcsaira az $e(v)$ címkét: ennyi élt tartalmaz $s \rightsquigarrow v$. Belátható, hogy ha nincs a gráfban s-ből elérhető negatív kör $\iff$ a QBF futása során minden, a fő ciklusban kezelt v csúcsra $e(v) < n$.

A fenti állítást fordítva megfogalmazva, ha van a gráfban s-ből elérhető negatív kör $\iff$ a QBF futása során van olyan, a fő ciklusban kezelt v csúcs, amelyre $e(v) \geq n$. Ilyenkor ez a v csúcs vagy része egy negatív körnek, amit a csúcsok π címkéi mutatnak meg, vagy a π címkéken keresztül el lehet belőle jutni egy ilyen negatív körbe, ami gráf csúcsainak n száma miatt összesen legfeljebb n lépésben azonosítható.

A továbbiakban tehát az s-ből elért v csúcsokra a $d(v)$, a $\pi(v)$ és az $inQ(v)$ mellett az $e(v)$ címkét is nyilvántartjuk.

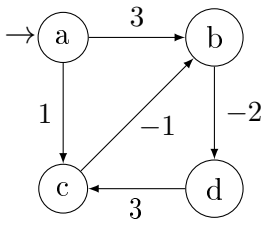
Ha valamely (u, v) él mentén végzett közelítés során úgy találjuk, hogy $e(v) \geq n$, akkor a fentiek szerint meghatározzuk valamelyik negatív kör egyik csúcsát, és az algoritmus ezzel tér vissza.

Ha a QBF futása során mindegyik közelítés után $e(v) < n$, akkor üres sorral állunk meg, és a $\oslash$ extremális hivatkozással térünk vissza.

Belátható, hogy az így kegészített QBF $O(n*m)$ idő alatt minden esetben befejeződik.

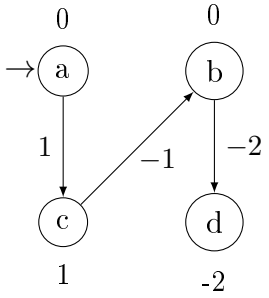
3.3.2. A legrövidebb utak fája meghatározásának szemléltetése

Mindegyik s-ből elérhető v csúcsra kiszámítunk egy $s \overset{opt}{\rightsquigarrow} v$ utat.



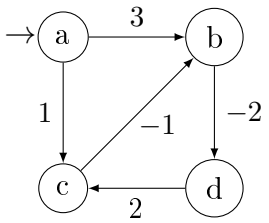
a → b, 3 ; c, 1.
b → d, −2.
c → b, −1.
d → c, 3.

$d; e$ változásai				kiter-	Q :	π változásai			
a	b	c	d	jesztett	Queue	a	b	c	d
$0; 0$	∞	∞	∞	csúcs: $d; e$	⟨ a ⟩	⊘	⊘	⊘	⊘
	$3; 1$	$1; 1$		a: $0; 0$	⟨ b, c ⟩		a	a	
			$1; 2$	b: $3; 1$	⟨ c, d ⟩				b
	$0; 2$			c: $1; 1$	⟨ d, b ⟩		c		
				d: $1; 2$	⟨ b ⟩				
			$-2; 3$	b: $0; 2$	⟨ d ⟩				b
				d: $-2; 3$	⟨ ⟩				
0	0	1	-2	végső d és π értékek		⊘	c	a	b



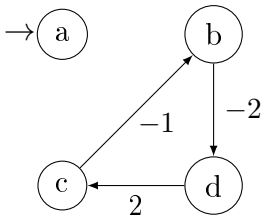
A legrövidebb utak fája s = a esetén. A csúcsoknál csak a d értékeket tüntettük föl.

3.3.3. A negatív körök kezelésének szemléltetése



a → b, 3 ; c, 1.
b → d, −2.
c → b, −1.
d → c, 2.

$d; e$ változásai				kiter-	Q :	π változásai			
a	b	c	d	jesztett	Queue	a	b	c	d
$0; 0$	∞	∞	∞	csúcs: $d; e$	⟨ a ⟩	⊘	⊘	⊘	⊘
	$3; 1$	$1; 1$		a: $0; 0$	⟨ b, c ⟩		a	a	
			$1; 2$	b: $3; 1$	⟨ c, d ⟩				b
	$0; 2$			c: $1; 1$	⟨ d, b ⟩		c		
				d: $1; 2$	⟨ b ⟩				
			$-2; 3$	b: $0; 2$	⟨ d ⟩				b
		$0; 4$		d: $-2; 3$	⟨ ⟩			d	

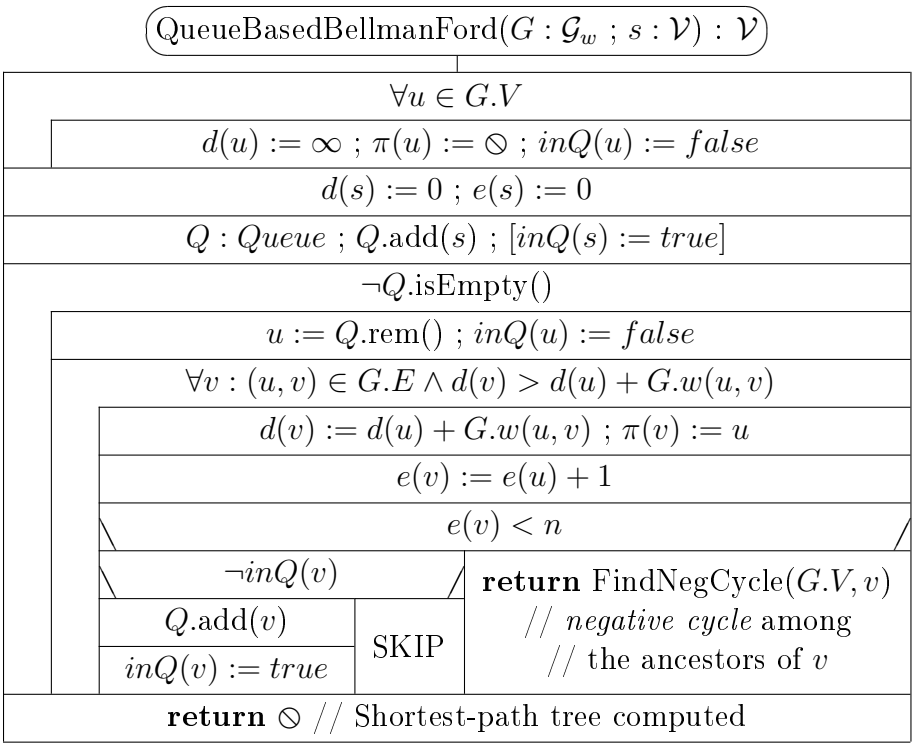


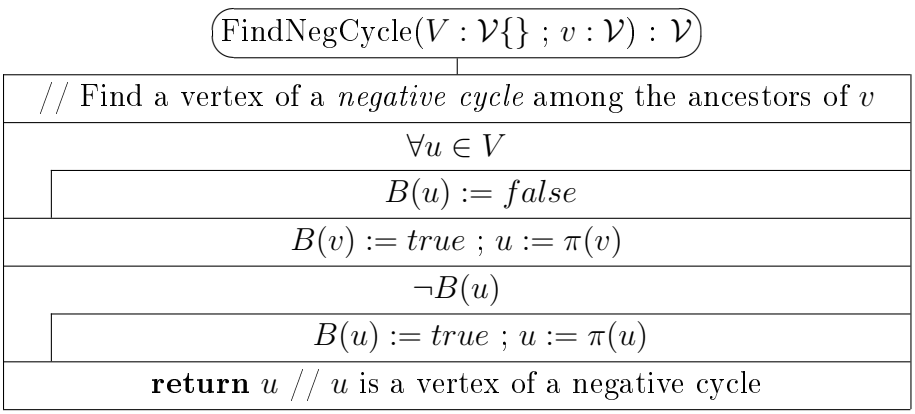
Itt megállunk, mert $e(\mathrm{c}) = 4 = n$, tehát negatív kör található a „c” csúcs ősei közt. A π értékek szerint visszafelé haladva megtalálhatjuk a negatív kört.

3.3.4. A negatív körök kezelésével kiegészített struktogramok

Vegyük figyelembe, hogy egy tetszőleges v csúcs d, e és π címkéjének módosítása után a v csúcsot akkor és csak akkor tesszük a sor végére, ha $e(v) < n$, és v pillanatnyilag nincs benne a sorban!

Ha biztosan nincs a gráfban s-ből elérhető negatív kör, akkor az algoritmus fentebbi változata alkalmazható.





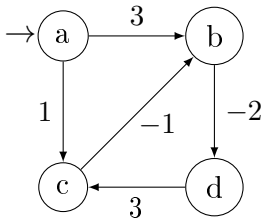
3.3.5. A sor-alapú Bellman-Ford algoritmus (QBF) elemzése

Az alábbi elemzésben arra az esetre szorítkozunk, amikor nincs a gráfban s-ből elérhető negatív kör.

Az algoritmus helyességének bizonyítása és az algoritmus hatékonyságának vizsgálata szempontjából is alapvető a QBF futásának menetekre bontása.

3.10. Definíció. *Menet rekurzív definíciója:*
– 0. menet: a start csúcs (s) feldolgozása.
– (i+1). menet: az i. menet végén a sorban levő csúcsok feldolgozása.

Vegyük pl. az első gráfunkon az algoritmus szemléltetését menetszámlálással együtt! **(Az $s \overset{opt}{\rightsquigarrow} v$ utak megkeresésének szemléltetése.)**



a → b, 3 ; c, 1.
b → d, −2.
c → b, −1.
d → c, 3.

$d; e$ változásai				kiter-	Q :	π változásai				
a	b	c	d	jesztett	Queue	a	b	c	d	Menet
0; 0	∞	∞	∞	csúcs: $d; e$	⟨ a ⟩	⊘	⊘	⊘	⊘	
	3; 1	1; 1		a: 0; 0	⟨ b, c ⟩		a	a		0.
			1; 2	b: 3; 1	⟨ c, d ⟩				b	1.
	0; 2			c: 1; 1	⟨ d, b ⟩		c			1.
				d: 1; 2	⟨ b ⟩					2.
			−2; 3	b: 0; 2	⟨ d ⟩				b	2.
				d: −2; 3	⟨ ⟩					3.
0	0	1	−2	végső d és π értékek		⊘	c	a	b	-

3.11. Tulajdonság. *A gráf tetszőleges u csúcsára, ha s-ből nem érhető el negatív kör, és az u csúcsba vezet k élből álló $s \overset{opt}{\rightsquigarrow} u$ út, akkor a k. menet elején már $d(u) = w(s \overset{opt}{\rightsquigarrow} u)$ és $\langle s, \ldots, \pi^2(u), \pi(u), u\rangle$ egy optimális út.*

Bizonyítás. k szerinti teljes indukcióval. □

3.12. Lemma. *Ha s-ből nem érhető el negatív kör, akkor tetszőleges u elérhető csúcsra van olyan $s \overset{opt}{\rightsquigarrow} u$ út, ami legfeljebb $n-1$ élből áll.*

Bizonyítás. Ebben az esetben az s-ből induló körmentes utak biztosan nem hosszabbak, mint a kört is tartalmazók. Tetszőleges körmentes útnak viszont legfeljebb $n-1$ éle van. Ebből azonnal adódik, hogy véges sok körmentes út van a gráfban. Akkor s-ből tetszőleges v csúcsba is véges sok körmentes út van, és így létezik ezek között optimális. □

3.13. Tétel. *(A fenti Lemma és Tulajdonság következménye)*
Ha s-ből nem érhető el negatív kör $\Rightarrow$ tetszőleges s-ből elérhető u csúcsra van olyan $s \overset{opt}{\rightsquigarrow} u$ út, amit az $n-1$. menet elejére már a QBF kiszámolt. $\Rightarrow$ az $n-1$. menet végére kiürül a sor, az algoritmus pedig $O(n*m)$ időben megáll, azaz

$$MT(n,m) \in O(n*m).$$

Az elméleti műveletigény becslés meglehetősen rossz hatékonyságot sugall. Gyakorlati tesztek pozitítív élsúlyú, véletlenszerűen generált, nagyméretű, s-ből összefüggő ritka gráfokra[2] ($m \in O(n)$) statisztikusan $\Theta(n)$ átlagos műveletigényt adtak, szomszédossági listás gráfábrázolás esetén. Ez viszont aszimptotikusan az elméleti minimummal egyezik. (A hálózatok jelentős része nagyméretű, ritka gráffal modellezhető. Nem véletlen, hogy hálózatokon való útkeresésnél alkalmazzák is ezt a viszonylag egyszerű, de általános algoritmust.)

[2]Egy gráf egy *adott csúcsból összefüggő*, ha ebből a csúcsból a gráf mindegyik csúcsa elérhető. A *ritka gráfokra* az $m \leq k*n$, ahol k előre adott konstans, gyakran $k \leq 4$.

4. Utak minden csúcspárra ([2] 25)

Ebben a fejezetben két algoritmust ismertetünk. Mindkettő a gráfok csúcsmátrixos reprezentációjára épül, műveletigényük $\Theta(n^3)$ és hasonló elven is működnek.[3] Floyd ún. Floyd-Warshall algoritmusa általánosabb feladatot old meg és egyben későbbi, mint Roy, illetve Warshall, tetszőleges gráf tranzitív lezártját meghatározó algoritmusa [2]. Az irányítatlan gráfokat mindkét algoritmus olyan irányított gráfnak tekinti, amelyben minden (u, v) élnek megvan az ellentétes irányú (v, u) élpárja és $w(u, v) = w(v, u)$.

4.1. Legrövidebb utak minden csúcspárra: a Floyd-Warshall algoritmus (FW)

4.1. Jelölés. $\mathbb{R}_\infty = \mathbb{R} \cup \{\infty\}$

Az $A/1 : \mathbb{R}_\infty[n, n]$ csúcsmátrixszal adott élsúlyozott gráf alapján meghatározza a $D/1 : \mathbb{R}_\infty[n, n]$ mátrixot, ahol $D[i, j]$ az i indexű csúcsból a j indexű csúcsba vezető optimális út hossza, vagy ∞, ha nincs út i-ből j-be. Megadja még a $\pi/1 : \mathbb{N}[n, n]$ mátrixot is, ahol $\pi[i, j]$ az algoritmus által kiszámolt, az i indexű csúcsból a j indexű csúcsba vezető optimális úton a j csúcs közvetlen megelőzője, ha $i \neq j$ és létezik út i-ból j-be. Különben $\pi[i, j] = 0$.

Előfeltétel: A gráfban nincs negatív kör. (Ezt az algoritmus ellenőrzi.)

A továbbiakban gráf csúcsait 1-től n-ig indexeljük és a csúcsokat az indexükkel azonosítjuk.

Az algoritmusunk a $\langle (D^{(0)}, \pi^{(0)}), (D^{(1)}, \pi^{(1)}), \ldots, (D^{(n)}, \pi^{(n)}) \rangle$ mátrixpársorozatot állítja elő, amit a következőképpen definiálunk.

4.2. Jelölés. *$i \overset{k}{\rightsquigarrow} j$ az i csúcsból a j csúcsba vezető út, azzal a megszorítással, hogy az úton a közbenső csúcsok indexe $\leq k$ ($i > k$ és $j > k$ megengedett, továbbá $i, j \in 1..n$ és $k \in 0..n$).*

4.3. Jelölés. *$i \underset{opt}{\overset{k}{\rightsquigarrow}} j$ az i csúcsból a j csúcsba vezető* legrövidebb *körmentes út, azzal a megszorítással, hogy az úton a közbenső csúcsok indexe $\leq k$. Ha több ilyen út lenne, azt vesszük, ahol a közbenső csúcsok indexeinek maximuma a lehető legkisebb.*

[3] Mindkettő az ún. *dinamikus programozás* iskolapéldája [2].

4.4. Mj. *k szerinti indukcióval adódik, hogy amennyiben létezik $i \overset{k}{\rightsquigarrow} j$ út $\Rightarrow$ az $i \underset{opt}{\overset{k}{\rightsquigarrow}} j$ út egyértelműen definiált, ui. a negatív körök hiánya miatt $i = j$ esetén $i \underset{opt}{\overset{k}{\rightsquigarrow}} j = \langle i \rangle$, $i \neq j$ esetén pedig*

- $\exists\, i \underset{opt}{\overset{0}{\rightsquigarrow}} j = \langle i, j \rangle \iff (i, j) \in G.E.$
- *$k \in 1..n$ esetén pedig egy $i \underset{opt}{\overset{k}{\rightsquigarrow}} j$ úttal kapcsolatban két lehetőség van:* $i \underset{opt}{\overset{k}{\rightsquigarrow}} j = i \underset{opt}{\overset{k-1}{\rightsquigarrow}} j \quad \vee \quad i \underset{opt}{\overset{k}{\rightsquigarrow}} j = i \underset{opt}{\overset{k-1}{\rightsquigarrow}} k \underset{opt}{\overset{k-1}{\rightsquigarrow}} j.$

4.5. Definíció. $D_{ij}^{(k)} = \begin{cases} w(i \underset{opt}{\overset{k}{\rightsquigarrow}} j) & \text{ha létezik } i \overset{k}{\rightsquigarrow} j \\ \infty & \text{ha nem létezik } i \overset{k}{\rightsquigarrow} j \end{cases}$

4.6. Definíció.
$\pi_{ij}^{(k)} = \begin{cases} \text{az } i \underset{opt}{\overset{k}{\rightsquigarrow}} j \text{ úton a } j \text{ csúcs közvetlen megelőzője,} & \\ & \text{ha } i \neq j \text{ és létezik } i \overset{k}{\rightsquigarrow} j \\ 0 & \text{ha } i = j \text{ vagy nem létezik } i \overset{k}{\rightsquigarrow} j \end{cases}$

4.7. Tulajdonság. *A $D^{(0)}$ mátrix megegyezik a gráf A csúcsmátrixával, a $D^{(n)}$ mátrix pedig éppen a kiszámítandó D mátrix.*

4.8. Tulajdonság.
$\pi_{ij}^{(0)} = \begin{cases} i & \text{ha } i \neq j \wedge (i, j) \text{ a gráf éle} \\ 0 & \text{ha } i = j \vee (i, j) \text{ nem éle a gráfnak} \end{cases}$
A $\pi^{(n)}$ mátrix pedig megegyezik a kiszámítandó π mátrixszal.

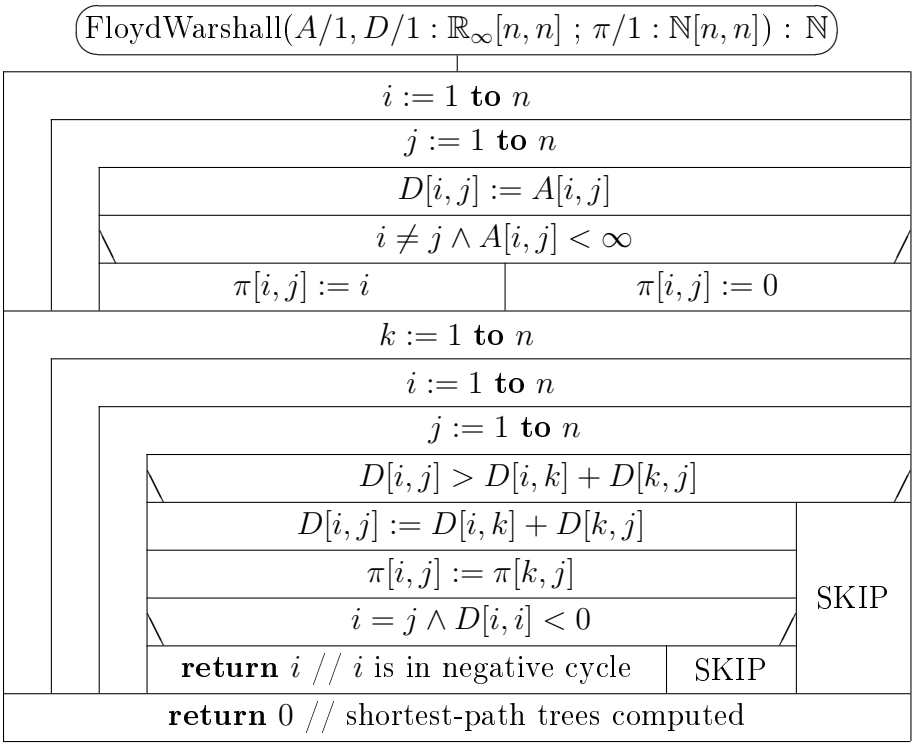
4.9. Tulajdonság. *A 4.4. megjegyzés alapján, $k \in 1..n$ esetén:*
Ha $D_{ij}^{(k-1)} > D_{ik}^{(k-1)} + D_{kj}^{(k-1)}$
akkor $D_{ij}^{(k)} = D_{ik}^{(k-1)} + D_{kj}^{(k-1)} \wedge \pi_{ij}^{(k)} = \pi_{kj}^{(k-1)}$
különben $D_{ij}^{(k)} = D_{ij}^{(k-1)} \wedge \pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$

4.10. Következmény. *$k \in 1..n$ esetén:*
$D_{ik}^{(k)} = D_{ik}^{(k-1)} \wedge \pi_{ik}^{(k)} = \pi_{ik}^{(k-1)} \quad \wedge \quad D_{kj}^{(k)} = D_{kj}^{(k-1)} \wedge \pi_{kj}^{(k)} = \pi_{kj}^{(k-1)}$

Ez azt jelenti, hogy a $D^{(k)}$ mátrix k-adik sora és oszlopa ugyanaz, mint $D^{(k-1)}$-é, és a $\pi^{(k)}$ mátrix k-adik sora és oszlopa is ugyanaz, mint $\pi^{(k-1)}$-é.

Mivel $D_{ij}^{(k)}$ csak a $D_{ij}^{(k-1)}$, a $D_{ik}^{(k-1)}$ és a $D_{kj}^{(k-1)}$ értékektől függ, valamint $D_{ik}^{(k)} = D_{ik}^{(k-1)} \wedge D_{kj}^{(k)} = D_{kj}^{(k-1)}$, a D mátrix kiszámításához nem kell segédmátrixokat tárolni. A fizikailag is létező D mátrixot az elméleti $D^{(0)}$ mátrix elemeivel inicializáljuk, az alábbi függvény első, kettős ciklusával, majd $k = 1$-től n-ig kiszámoljuk $D^{(k-1)}$-ből $D^{(k)}$-t, az alábbi függvény háromszorosan beágyazott ciklusával, fizikailag végig ugyanazt a D mátrixot használva. Ugyanígy, a π mátrixból is elég egy példány.

Ha a D mátrix főátlójában negatív érték jelenik meg, akkor negatív kört találtunk (ennek meggondolását az Olvasóra bízzuk).



Az algorimus műveletigényéhez elég meggondolni, hogy az első külső ciklus n-szer, a belső n^2-szer fut le, így az inicializálás műveletigénye $\Theta(n^2)$. Ugyanígy, ha nincs a gráfban negatív kör, akkor a második külső ciklus mindegyik iterációjának futási ideje $\Theta(n^2)$, így az összes iterációjáé $\Theta(n^3)$, ami egyben a teljes algoritmus maximális műveletigénye is, azaz $MT(n) \in \Theta(n^3)$. Ha van a gráfban negatív kör, ez akár a második külső ciklus első iterációja alatt kiderülhet, így $mT(n) \in \Theta(n^2)$.

4.1.1. A legrövidebb utak minden csúcspárra probléma *visszavezetése* a legrövidebb utak egy forrásból feladatra

Vegyük észre, hogy a Floyd-Warshall (FW) algoritmus által D és π mátrixok i-edik sora a *legrövidebb utak egy forrásból* feladat megoldása $s = i$ esetére. Megfordítva, ha minden $s \in 1..n$ értékre megoldjuk a *legrövidebb utak egy forrásból* feladatot, megkapjuk a *legrövidebb utak minden csúcspárra* probléma megoldását is.

Az alábbiakban – a teljesség igénye nélkül– megvizsgálunk néhány esetet.

Ha például a gráfunk DAG, ilyen módon a *legrövidebb utak minden csúcspárra* probléma megoldása a legrosszabb esetben is $\Theta(n * (n + m))$ idő alatt számolható ki, ami ritka gráfokon $\Theta(n^2)$, tehát nagysárenddel gyorsabb, mint az FW algoritmus. Sűrű gráfokon viszont – feltéve, hogy a csúcsok többségéből a gráfunk nagy része elérhető – $\Theta(n^3)$ a műveletigény, ami a gyakorlatban a nagyobb rejtett konstansok miatt lassúbb futást eredményez.

Hasonló eredményeket kaphatunk, ha élsúlyozatlan gráfokra szélességi keresést alkalmazunk.

Ha a gráfunk élsúlyai nemnegatívak, a Dijkstra algoritmust minden $s \in 1..n$ értékre végrehajtva $MT(n, m) \in O(n * (n + m) * \log n)$. Ritka gráfokra ez még mindig csak $O(n^2 * \log n)$, ami aszimptotikusan lényegesen jobb, mint az FW algoritmus $\Theta(n^3)$ műveletigénye, tehát nemnegatív élsúlyú, nagyméretű, ritka gráfokon ez lesz a jobb megoldás. Sűrű gráfokra a *felső becslés* most $MT(n, m) \in O(n^3 * \log n)$, ami rosszabb, mint az FW esetén.

Ha csak annyit tudunk, hogy nincs a gráfunkban negatív kör, a QBF algoritmust minden $s \in 1..n$ értékre végrehajtva $MT(n, m) \in O(n * n * m)$, ami – legalábbis a legrosszabb esetet tekintve, és feltételezve, hogy nincs lényegesen kevesebb él, mint csúcs a gráfban – sosem ad jobb *felső becslést*, mint amit az FW algoritmus esetén kaptunk.

4.2. Gráf tranzitív lezártja (TC)

Ebben az alfejezetben csak azzal foglalkozunk, hogy egy hálózatban (gráfban) honnét hova lehet eljutni. Az eljutás módja és költsége most nem érdekel bennünket. Ebből célból vezetjük be a címben jelzett fogalmat.

4.11. Definíció. *A $G = (V, E)$ gráf tranzitív lezártja alatt a $T \subseteq V \times V$ relációt értjük, ahol*

$(u, v) \in T \iff$ *ha G-ben az u csúcsból elérhető a v csúcs.*

4.12. Jelölés. $\mathbb{B} = \{0; 1\}$

Amennyiben a gráfunk csúcsai az $1..n$ indexekkel azonosíthatók, a gráfot ábrázolhatjuk az $A/1 : \mathbb{B}[n, n]$ csúcsmátrixszal (az esetleges élsúlyoktól eltekintve), tranzitív lezártját pedig a $T/1 : \mathbb{B}[n, n]$ mátrix segítségével, ahol

$T[i, j] \iff$ ha az A mátrixszal ábrázolt gráfban van út az i csúcsból a j csúcsba.

A T mátrix kiszámításához definiáljuk a $\langle T^{(0)}, T^{(1)}, T^{(n)} \rangle$ mátrixsorozatot, aminek utolsó tagja magával a T mátrixszal lesz egyenlő.

4.13. Definíció. $T_{ij}^{(k)} \iff \exists\, i \overset{k}{\rightsquigarrow} j \quad (k \in 0..n \ \wedge \ i, j \in 1..n)$

4.14. Tulajdonság. *(A T mátrixok rekurzív megadása.)*

$T_{ij}^{(0)} = A[i, j] \vee (i = j) \qquad (i, j \in 1..n)$

$T_{ij}^{(k)} = T_{ij}^{(k-1)} \vee T_{ik}^{(k-1)} \wedge T_{kj}^{(k-1)} \quad (k \in 1..n \ \wedge \ i, j \in 1..n)$

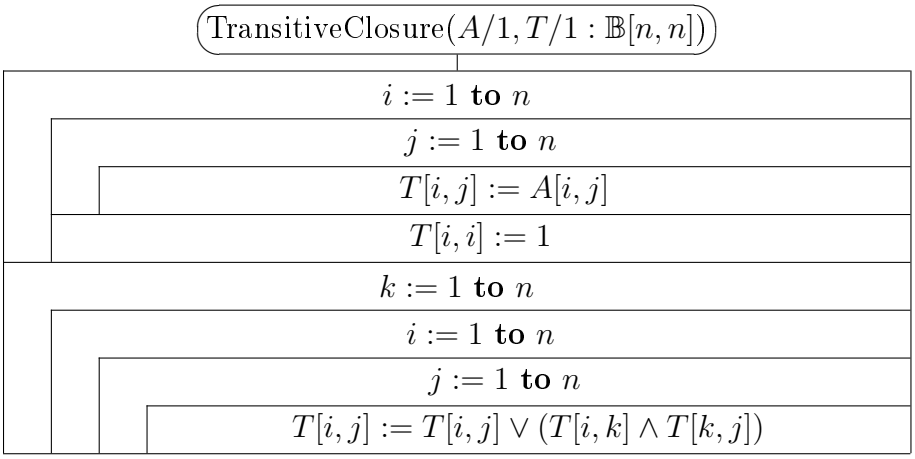
4.15. Következmény.

$T_{ik}^{(k)} = T_{ik}^{(k-1)} \quad \wedge \quad T_{kj}^{(k)} = T_{kj}^{(k-1)} \quad (k \in 1..n \ \wedge \ i, j \in 1..n)$

Ez azt jelenti, hogy a $T^{(k)}$ mátrix k-adik oszlopa és sora ugyanaz, mint a $T^{(k-1)}$ mátrixé. $\quad (k \in 1..n)$

Mivel $T_{ij}^{(k)}$ csak a $T_{ij}^{(k-1)}$, a $T_{ik}^{(k-1)}$ és a $T_{kj}^{(k-1)}$ értékektől függ, valamint $T_{ik}^{(k)} = T_{ik}^{(k-1)} \wedge T_{kj}^{(k)} = T_{kj}^{(k-1)}$, a T mátrix kiszámításához nem kell segédmátrixokat tárolni. A fizikailag is létező T mátrixot az elméleti $T^{(0)}$ mátrix elemeivel inicializáljuk, az alábbi eljárás első, kettős ciklusával, majd $k = 1$-től n-ig kiszámoljuk $T^{(k-1)}$-ből $T^{(k)}$-t, az alábbi eljárás háromszorosan beágyazott ciklusával, fizikailag végig ugyanazt a T mátrixot használva:

Amikor a háromszorosan beágyazott ciklus végrehajtása során $T[i, j]$ új értékét számolom ki, az elméleti mátrixsorozat $T_{ij}^{(k)}$ elemét határozom meg. Az értékadás végrehajtása előtt még egyrészt $T[i, j] = T_{ij}^{(k-1)}$, másrészt pedig $T[i, k] = T_{ik}^{(k)} = T_{ik}^{(k-1)}$ és $T[k, j] = T_{kj}^{(k)} = T_{kj}^{(k-1)}$, tehát mindegy, hogy a második külső ciklus adott iterációján belül a $T[i, k]$, illetve a $T[k, j]$ már értéket kapott-e.



A fenti algoritmusra $T(n) \in \Theta(n^3)$ az FW algoritmusnál már alkalmazott gondolatmenethez teljesen hasonlóan adódik.

Az is világos, hogy a tranzitív lezárt eredményképpen $T[i,j] \iff D[i,j] < \infty$ az FW algoritmus végrehajtása után ($i,j \in 1..n$).

Ez az algoritmus ezt az egyszerűbb feladatot gyakorlatilag mégis hatékonyabban oldja meg, mint az FW a magáét, az egyszerűbb ciklusmagok miatt.

4.2.1. A tranzitív lezárt kiszámításának *visszavezetése* szélességi keresésre

A szélességi keresés után, $s = i$ esetben éppen azok a csúcsok lesznek feketék, amelyek elérhetők i-ből.

Ha tehát lefuttatjuk a szélességi keresés egyszerűsített változatát, ahol $T[i,j] \iff \mathit{color}(j) \neq \mathit{white}$, éppen a T mátrix megfelelő sorát kapjuk meg, $MT(n,m) \in \Theta(n+m)$ műveletigénnyel.

A mátrix összes sorát így $\Theta(n*(n+m))$ maximális futási idővel kapjuk meg, ami ritka gráfok esetén $\Theta(n^2)$, azaz aszimptotikusan jobb, mint a TC algoritmus, a gráf sűrűségétől független műveletigénye.

Sűrű gráfok esetében viszont a szélességi keresésekkel is $\Theta(n^3)$ lesz a műveletigény, ami a gyakorlatban hosszabb futási időt eredményez, a nagyon egyszerű TC algoritmussal összehasonlítva.