Algoritmusok és adatszerkezetek 2

2015/16 tavaszi félév

Előadó: Dr. Ásványi Tibor

Készítették:

Koruhely Gábor (Koru)

Szalay Richárd (Whisperity)

Lektorálta:

Dr. Ásványi Tibor

Frissítve: 2019. 06. 07.

A tárgy követelményei:

- Legalább elégséges gyakorlati jegy kell a vizsgázáshoz,
- vizsga
    - 3szor lehet maximum vizsgázni egy vizsgaidőszakban,
    - vizsga ugyanolyan felépítésű, mint előző félévben az algo 1

Tematika:

- A tárgy honlapján megtalálható a tárgy tematikája: http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet.pdf
- AVL fák (ad2jegyzet.pdf)
- általános fák (ad2jegyzet.pdf)
- B+ fák (http://aszt.inf.elte.hu/~asvanyi/ad/B+ fa.pdf)
- gráf algoritmusok, pl.
    - mélységi bejárás és alkalmazásai
    - legrövidebb út
    - minimális feszítőfa
- veszteségmentes tömörítés
- mintaillesztés

Gráf algoritmusok

Gráfok ábrázolásai:

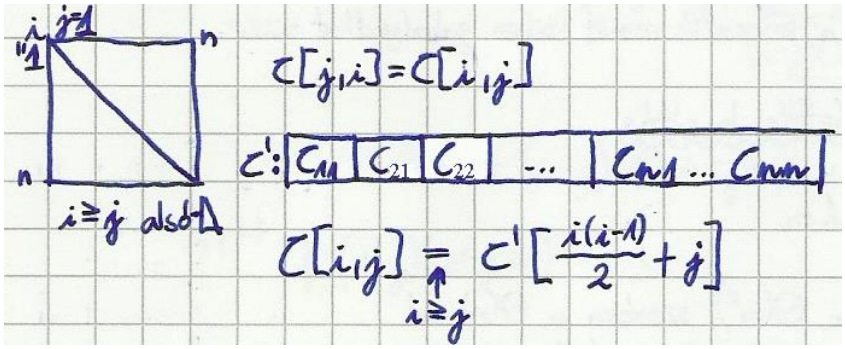
Szomszédosági mátrix (C)
$G= (V,E)$ $E \subseteq V \times V$
$V = 1..n$
$C[1..n,1..n]$

$$C[i,j] = \begin{cases} 1 \Leftrightarrow (i,j) \in E \\ 0 \Leftrightarrow (i,j) \notin E \end{cases}$$

ha számít az él költsége $w: E \rightarrow \mathbb{R}$

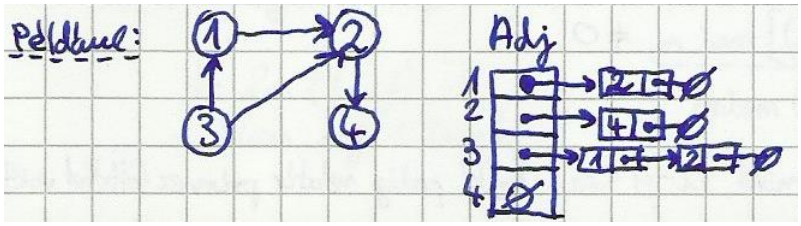
$$C[i,j] = \begin{cases} w(i,j) \Leftrightarrow (i,j) \in E \quad (i \neq j) \\ 0 \Leftrightarrow i = j \\ +\infty \text{ különben} \end{cases}$$

Azonban a mátrix tárolás miatt a memóriaigény $\Theta(n^2)$, ami az irányítatlan esetben a szimmetria kihasználásával javítható (majdnem felezhető), de $\Theta(n^2)$ marad (csak alsó háromszög mátrixot tároljuk).



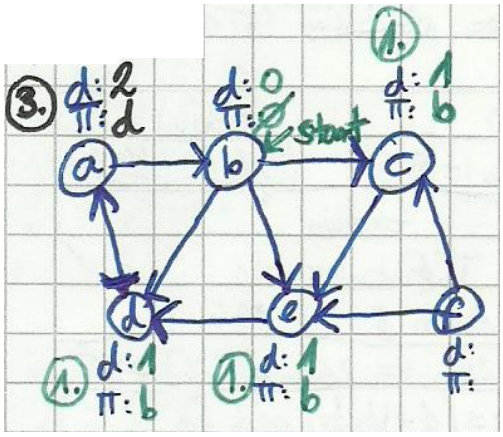
Feladat: Ha a gráfban nincsenek hurokélek, a szomszédossági mátrix fő átlójában csupa 0 érték van. Hogyan lehetne ezt az irányítatlan gráfok ábrázolásánál kihasználni?

Szomszédsági éllistás reprezentáció (élsúlyozott esetben az élsúlyokat is a listaelemekben tároljuk, irányítatlan gráfoknál minden (u,v) élt (v,u) élként is tárolunk):
Memóriaigény n csúcs és e él esetén: $\Theta(n + e)$

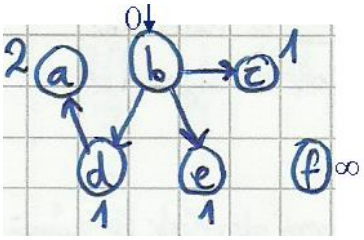

Példa:
1
2
3
4
Adj
1
2
3
4

Elemi gráf algoritmusok

Szélességi keresés (Breadth-first Search = BFS)
Meghatározzuk a start csúcsból a további csúcsokba a legkevesebb élet tartalmazó utat.
d – hány élen keresztül jutunk a csúcsba
π – melyik csúcsból jutunk a csúcsba

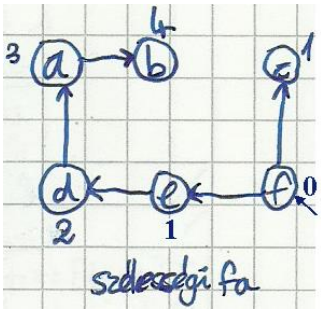


Q = <b>
1. lépés: Q= <c, d, e> /*b szomszédai*/
2. lépés: Q= <d, e> /*c rákövetkezője e, de ott már voltunk*/
3. lépés: Q= <e, a> /*„a” d szomszédjai*/
4. lépés: Q= <a> /*e szomszédja csak a d, de ott már voltunk*/
5. lépés: Q=<> /*a sor kiürült, a bejárás véget ért



f-ből indítva, másik szemléltetési móddal:

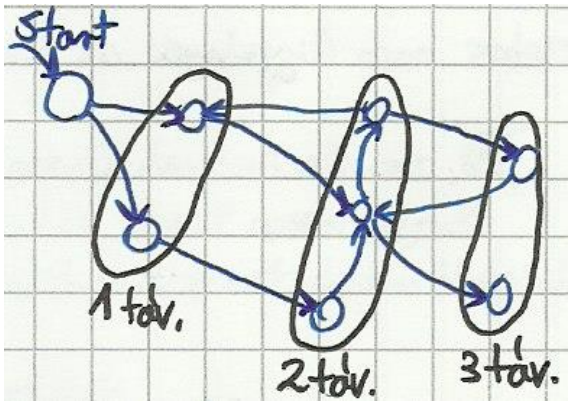
d	a	b	c	d	e	f	Q π	a	b	c	d	e	f
	∞	∞	∞	∞	∞	0	<f>	∅	∅	∅	∅	∅	∅
f			1		1		<c, e>			f		f	
c							<e>						
e				2			<d>				e		
d	3						<a>	d					
a		4					<b>		a				
b	3	4	1	2	1	0	< >	d	a	f	e	f	∅



Az irányított gráfon a „szomszédsági” kapcsolat nem szimmetrikus.
A fenti gráfban pl.: a szomszédja b, de b-nek nem szomszédja a.

Jelölés:
Ha G = (V, E), E ⊆ V × V gráf,
G.V a G csúcsainak halmaza
G.E a G éleinek halmaza
u ∈ G.V esetén
G.Adj[u] $\stackrel{\text{def}}{=} \{ v \in G.V \mid (u,v) \in G.E \}$
Feltesszük, hogy $(u,u) \notin G.E$, azaz nincs hurokél.
Feltesszük, hogy nincsenek párhuzamos élek sem.
} G egyszerű gráf

```
BFS(G,s)
    ∀ u ∈G.V
        u.d := ∞
        u. π := ∅
        u.szín := fehér
s.d := 0
s.szín := szürke
Q := <s>
    Q ≠<>
        u:= Q.sorból()
            ∀ v ∈ G.Adj[u]
                v.szín = fehér
                    v.d := u.d + 1
                    v.π := u
                    v.szín := szürke
                    Q.sorba(v)
        u.szín := fekete
```



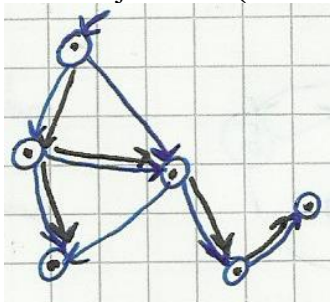
BFS műveletigénye: O(n + e), ahol n = |V| és e=|E| (MT(n,e) ∈ Θ(n + e), mT(n,e) ∈ Θ(n))
Az első ciklus n-szer iterál, a fő ciklus legfeljebb n-szer, a belső ciklus pedig legfeljebb e-szer
(a start csúcsból nem elérhető csúcsok és élek maradnak ki.).

DFS (Mélységi keresés = Depth-first Search)

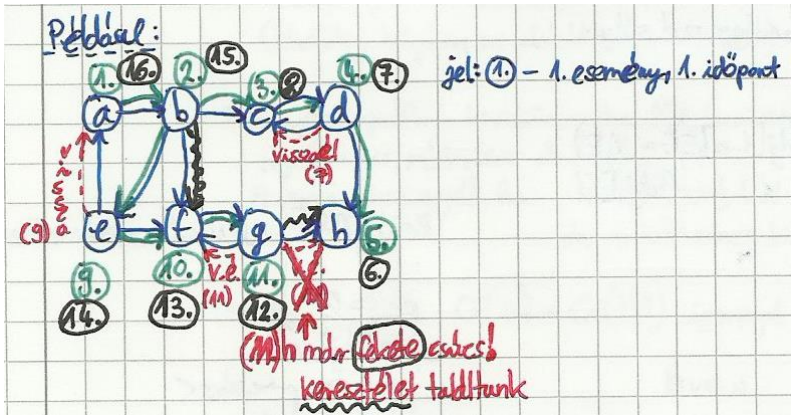
```
DFS(G)
    ∀ u ∈ G.V
        u. π := ∅
        u.szín := fehér
ido := 0
    ∀ u ∈ G.V
        u.szín = fehér
            DFS_VISIT(u,G, ido)
```

```
DFS_VISIT(u,G, &ido)
ido := ido + 1
u.d := ido
u.szín := szürke
    ∀ v ∈ G.Adj[u]
        v.szín = fehér
            v.π := u
            DFS_VISIT(v,G, ido)
        v.szín = szürke
            viszael(u,v)
u.szín := fekete
ido := ido + 1
u.f := ido
```

u.d = elérési idő (discovery time)
u.f = befejezési idő (finishing time)



Mélységi bejárás példa:



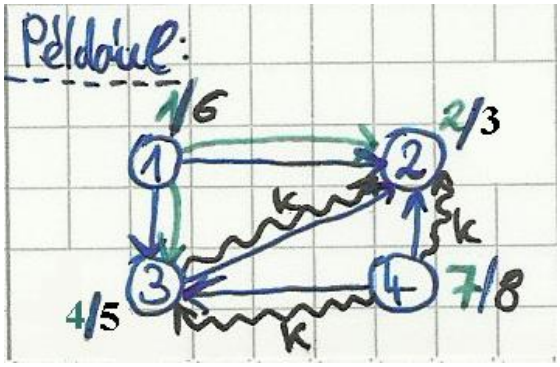
Ha az él már fekete csúcsba mutat, keresztélet vagy előreélet találtunk.

Élek osztályozása

Def:
a ⟶ b faél: ez kerül a mélységi fába, e mentén járjuk be a gráfot
f ⟵ g visszaél: (f a g őse egy mélységi fában.)
b ⟶ f előreél: b-ből kettő vagy több faélből álló út vezet f-be.
g ⟶ h keresztél: g és h két olyan csúcs, amelyek más ágon vannak, vagy másik mélységi fában

(*)

Példa:



Azaz a keresés eredménye egy mélységi erdő

DFS műveletigénye: $T(n,e) \in \Theta(n+e)$ Igazolás: Az első és a második ciklus is n-szer iterál (végig mennek a csúcsokon). A rekurzív DFS_VISIT eljárást is összesen n-szer: minden csúcsra egyszer hívjuk meg, amikor a csúcs még fehér. A DFS_VISIT eljárás ciklusa minden élet egyszer dolgoz fel, tehát összesen e-szer iterál.

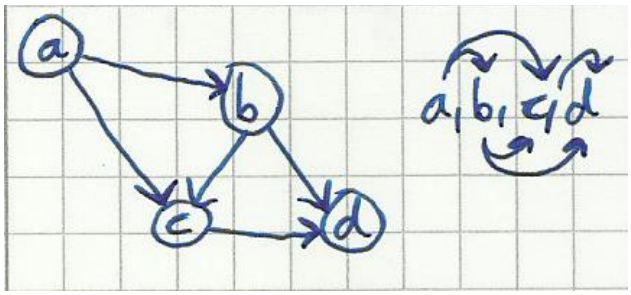
Tétel:
Ha a mélységi bejárás során egy (u,v) élet találunk:
(u,v) faél ⇔ v.szín = fehér
(u,v) visszaél ⇔ v.szín = szürke
(u,v) előreél ⇔ v.szín = fekete ∧ u.d < v.d
(u,v) keresztél ⇔ v.szín = fekete ∧ u.d > v.d

Tétel:
A mélységi bejárás akkor és csak akkor talál visszaélet, ha irányított kör van a gráfban.
Ha a mélységi bejárás során egy (u,v) visszaélet találunk, akkor az u és v csúcsokat tartalmazó mélységi fában u a v leszármazottja, így az (u,v) visszaéllel együtt irányított kört találtunk.

Megjegyzés: A mélységi bejárást mi csak egyszerű, irányított gráfokra értelmezzük.
Azt, hogy a gráf egyszerű (nem tartalmaz sem hurokélet sem párhuzamos éleket), mindegyik gráfalgoritmusunknál feltesszük, de tetszőleges irányított gráf tartalmazhatja az (u,v) él mellett a (v,u) élet is. Irányítatlan gráfok esetén pedig tetszőleges (u,v) élre definíció szerint (u,v) = (v,u).

Irányított gráfok csúcsainak topologikus rendezése

Topologikus rendezés: A csúcsok olyan sorrendje, amelyben minden él egy-egy később jövő csúcsba (szemléletesen: balról jobbra) mutat.



Tétel: Pontosan akkor ∃ topologikus rendezés, ha nincs irányított kör a gráfban.
Bizonyítás:

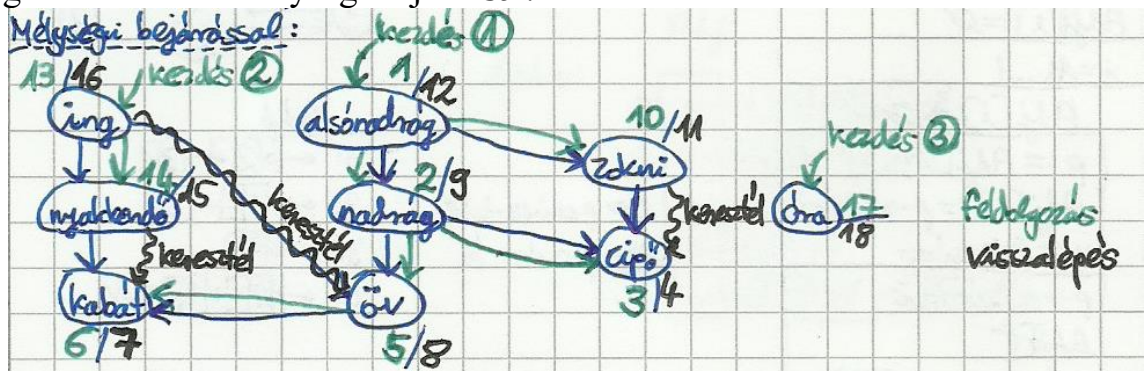
a) Ha van irányított kör a gráfban, jelölje $\langle u_1, u_2, \dots, u_k, u_1 \rangle$!
Ekkor egy tetszőleges topologikus rendezésben u_1 után jön valahol u_2, az után valahol u_3, és így tovább, végül is u_1 után jön u_k, és u_k után u_1, ami ↯, tehát ekkor nincs topologikus rendezés (a gráf csúcsain).
b) Ha nincs irányított kör a gráfban, akkor nyilván van olyan csúcs, aminek nincs megelőzője.
Ha veszünk egy megelőzővel nem rendelkező csúcsot és töröljük a gráfból, akkor a maradék gráfban nem keletkezik irányított kör, lesz megint legalább egy olyan, amelyiknek nincs megelőzője.
Sorban a törölt csúcsok adják a topologikus rendezést.

Def.: Legyen G=(V,E) gráf! Ekkor tetszőleges u∈V csúcsra definíció szerint: Adj[u] = {v ∈ V |(u, v) ∈ E} ⊆ V

```
fv.TR(G, TR[1..n])
BEFOKOK(G,BE[1..n])                          Θ(n + e)
i := 0
H:= {v ∈ V | BE[v] = 0}                      Θ(n)
   H ≠ {}
      u 'from' H                             'from' ← kivesz H-ből egy elemet
      i := i+1                               a kör csúcsai nem kerülnek H-ba, így az eredménybe (TR[1..n]) sem.
      TR[i] := u
         ∀ v ∈ G.Adj[u]                      O(n+e)
            BE[v] := BE[v] - 1
               BE[v] = 0
               H:=H ∪{v}
return i                                     ha i=n, minden csúcsot feldolgoztunk
```

Műveletigény: $O(n + e)$.

Topologikus rendezés mélységi bejárással:

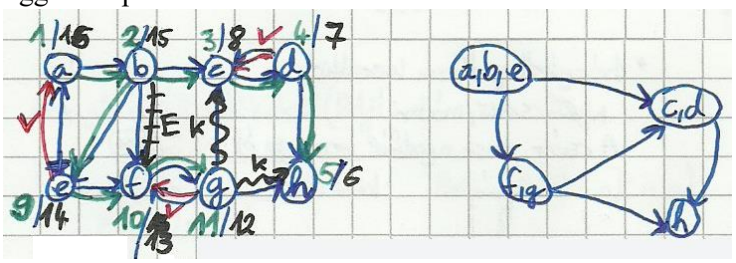


A sorrend: <óra, ing, nyakkendő, alsónadrág, zokni, nadrág, öv, kabát, cipő>
Tetszőleges u csúcs, akkor kerül a TR-be, a hátulról első szabad helyre, amikor befejeztük.
Ha a mélységi bejárás visszaélt talál ⇔ irányított kör van a gráfban ⇔ nincs topologikus rendezés.

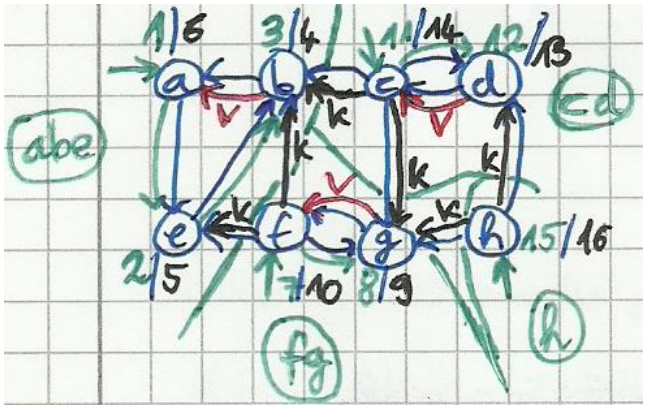
Erősen összefüggő komponensek

/*Minden csúcsból minden csúcsba vezet irányított út a komponensen belül*/
A gráf erősen összefüggő komponensei:

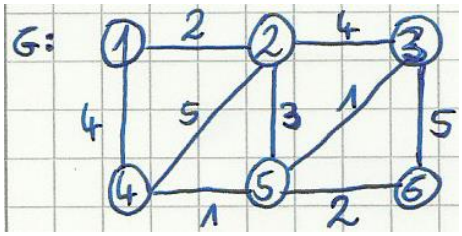
- a,b,e
- f,g
- c,d
- h



Előállítás: mélységi bejárással mintha topologikus rendezés lenne, de nem kezeljük a visszaéleket.
A sorozat: <a, b, e, f, g, c, d, h> a csúcsok befejezés szerinti sorrendben monoton csökkenően.
Transzponáljuk a gráfot, ezen mélységi bejárás a sorrend szerint, az így talált mélységi fák a komponensek.



Minimális feszítőfák



Kössünk össze minden várost, hogy minden pontba eljusson az áram, de a lehető legolcsóbban (és még sok más megfogalmazású feladat)

A lényeg minimális feszítőfa keresése.

G=(V,E) E ⊆ V x V (u,v) ∈ E ⇔ (v,u) ∈ E /*u ≠ v, a hurokéleknek nincs értelme*/
w: E→ ℝ élsúlyok
w(u,v) = w(v,u)

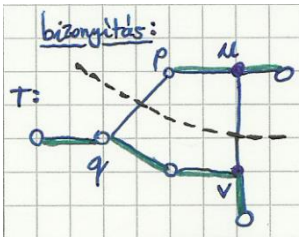
```
GEN_MST(G,A)
A:={}
s:=0;   n := |G.V|
    s < n−1
        (u,v) legyen olyan él a G.E \ A-ból,
        ami A-hoz biztonságosan hozzávehető
        A:= A ∪ {(u,v)}
        s := s + 1
```

azaz A ∪ {(u,v)} minimális feszítőfa része marad
P invariáns = $\exists\, T = (V, T_E)$ MST, hogy $A \subseteq T_E$

Def1: G=(V,E), ∅ ⊊ S ⊊ V esetben (S, V\S) vágás a G gráfban.
Def2: (u,v) él keresztezi az (S, V\S) vágást ⇔ az u ∈ S és v ∈ V\S (vagy fordítva u ∈ V\S és v ∈ S).
Def3: A ⊆ E és (S, V\S) vágás esetén a vágás elkerüli az A-t ⇔ A egyetlen éle sem keresztezi a vágást.
Def4: (u,v) könnyű él az (S, V\S) vágásban ⇔ (u,v) keresztezi a vágást, és tetszőleges, a vágást keresztező (x,y) élre w(u,v) ≤ w(x,y).

Tétel: G = (V,E) irányítatlan w: E → ℝ-rel élsúlyozott gráf
(S, V\S) vágás a gráfban elkerüli A− t, ahol $\exists T = (V, T_E)$ $MST, hogy\ A \subseteq T_E$
$(u, v) \in E$ egy könnyű él (legkisebb költségű) a vágásban
Ekkor (u,v) biztonságos A-ra nézve

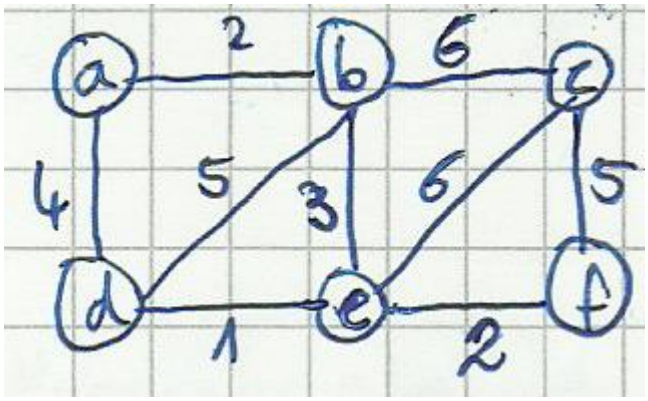
Bizonyítás: (u,v) nem eleme A-nak, ui. keresztezi az A-t elkerülő vágást.



a) $(u, v) \in T_E$ ✓
b) $(u,v) \notin T_E \Rightarrow \exists (p, q) \in T_E$, ami keresztezi az (S, V\S) vágást, ui: T feszítőfa, tehát T-ben el lehet jutni u-ból v-be.
⇒ w(p,q) ≥ w(u,v) és (p.q) nem eleme az A élhalmaznak
(p,q) törlésével az MST szétesik, de ha ehhez (u,v)-t hozzávesszük, akkor újra feszítőfa lesz.
T' = T \ {(p,q)} ∪ {(u,v)}
w(T') = w(T) – w(p,q) + w(u,v) ≤ w(T)
viszont mivel T MST volt, w(T') ≥ w(T) ⟹ azaz w(T') = w(T) és T' is MST kell, hogy legyen.

110_2 EA

Minimális feszítőfa ismétlése példán keresztül



G=(V,E)
w: E → ℝ
egy vágás:

S	V\S
{a,b,d}	{c,e,f}

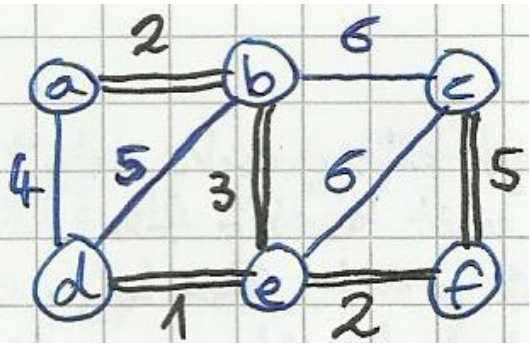
egy vágást keresztező él az $E \cap (S \times (V \setminus S))$ egy eleme, ennek a halmaznak a legkisebb súlyú tagja a könnyű él

A	S	V\S
∅	{a, b, d}	{c, e, f}
d-e	{a, b, d, e, f}	{c}
d-e c | f	{a}	{b, c, d, e, f}
a-b d-e c | f	{a, b, d, e}	{c, f}
a-b d-e-f | c	{a, b}	{c, d, e, f}

Az új vágás a két élt kerülje el, ne menjen keresztül rajtuk, majd mindig választjuk a könnyű élt.

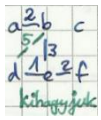
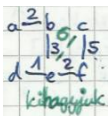
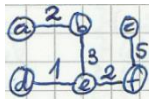
Az eddig kiszámolt feszítőfa-részlet (A) élhalmaza diszjunk kell, hogy legyen a vágás éleivel.

a-b-e-f a végeredmény:
d' c'

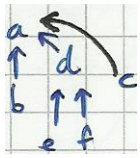


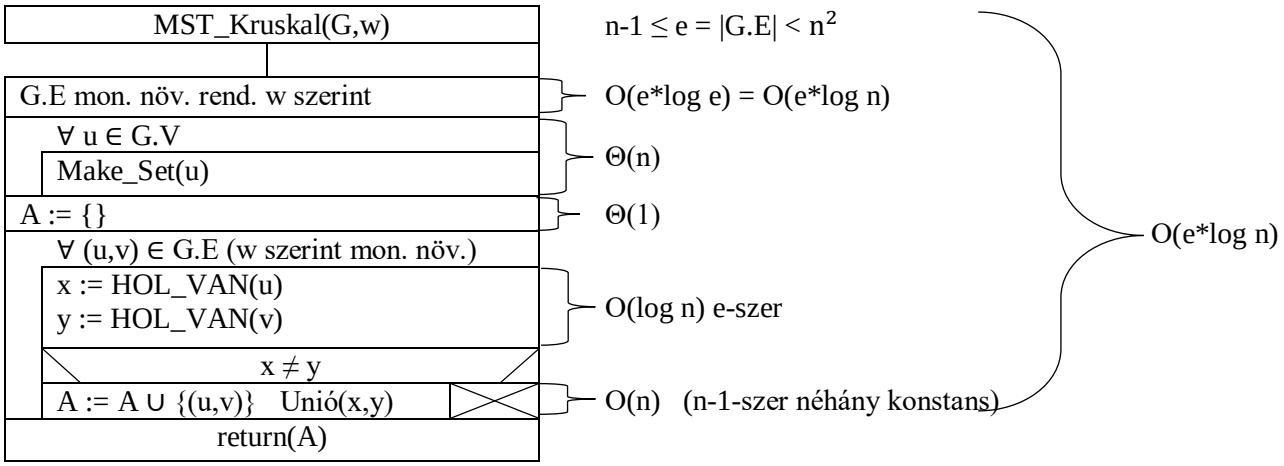
Kruskal algoritmus

- Elsőként súly szerint sorba rendezzük a csúcsokat! Minden csúcs egy egyelemű fát képezzen!
- Majd minden lépésben hozzávesszük a minimális élt, ha külön komponenseket kötnek össze (ha nem, kihagyjuk)

0. lépés	1	2	3	4	5	6	7	8	9	végeredmény:
	d¹e	a²b	e²f	b |3 e	a |4 d		c |5 f	b⁶c		
a b c d e f	a b c d¹e f	a²b c d¹e f	a²b c d¹e²f	a²b c |3 d¹e²f	a²b c |4 |3 d¹e²f kihagyjuk		a²b c |3 |5 d¹e²f	a²b⁶c |3 |5 d¹e²f kihagyjuk		

A komponensek nyilvántartásához menetközben egy másik, irányított erdőt is kezelünk, amiben a fáknak ugyanaz a csúcshalmaza, mint a nekik megfelelő (gyökerük felé) irányított fáknak. A komponenseket az irányított fák a gyökércsúcsai azonosítják:

0	1 (d-e)	2 (a-b)	3 (f-d)	4 (b-e)	5	6	7 (c-f)	8	9
a b c d e f	a b c d f ↑ e	a d c ↑ ↑ b e f	a d c ↑ ↑ ↖ b e f	a←d c ↑ ↑ ↖ b e f	(kihagyjuk)	(kihagyjuk)		(kihagyjuk)	(kihagyjuk)



A Make_Set(u) művelet a gráf mindegyik csúcsából egyelemű irányított fát képez.

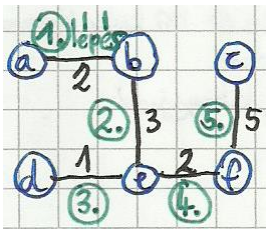
A HOL_VAN() megállapítja, a csúcs melyik ilyen köztes fában van (megkeresi a csúcsot tartalmazó irányított fa gyökércsúcsát). Ennek megfelelően az unió később az irányított fák gyökércsúcsait köti össze.

A fenti példában az unió művelet az alacsonyabb irányított fa gyökerét a magasabb irányított fa gyökere alá köti be, így a magasabb fa h magassága nem változik, csak mérete nő. (Mindegyik irányított fa gyökerében nyilvántartjuk a magasságát. Így az unió eljárás konstans műveletigényű.) Közös magasság esetén az egyesített fa magassága h+1 lesz (h közös magasság volt). Könnyen belátható, hogy ezzel a módszerrel az egyes fák magassága sosem lesz nagyobb, mint a méretük kettes alapú logaritmusa. Így a HOL_VAN() függvény logaritmikus műveletigényű. Egyszerű felső becsléssel az irányított fák magassága legfeljebb $\log_2 n$ $(n = |G.V|)$.

Másik módszer: Az unió művelet a kisebb méretű irányított fa gyökerét a nagyobb (vagy vele egyenlő méretű) irányított fa gyökere alá köti be. (Mindegyik irányított fa gyökerében nyilvántartjuk a méretét, azaz csúcsainak számát. Az unió eljárás így is konstans műveletigényű.) Könnyen belátható, hogy ezzel a módszerrel sem lesz az egyes fák magassága nagyobb, mint a méretük kettes alapú logaritmusa. Így a HOL_VAN() függvény most is logaritmikus műveletigényű.
Egyszerű felső becsléssel az irányított fák magassága most is legfeljebb $\log_2 n$ $(n = |G.V|)$.

Prim algoritmus

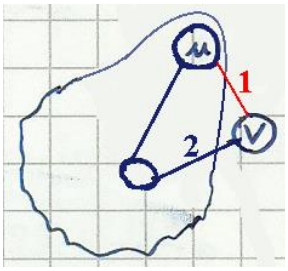
- Kiindulva egy csúcsból és a (csúcs, maradék) vágás közé választja a minimális élt.
- Ezt hozzáveszi az eredményhez és az (eddigi eredmény, maradék) vágásban keresi a könnyű élt.
- Addig folytatja az előző két lépést, míg <u>minimális feszítőfát</u> nem kapunk.



(A jobboldali minimális feszítőfa [MST] az előző példa gráfra vonatkozik.)

Meg kell adni, hogy hol kezdi (s), de mindegy, hogy hol kezdi.

```
MST_PRIM(G, s, w)
    ∀ u ∈ G.V                                   } Θ(n)
        u.c := ∞
        u.π := ∅
    s.c := 0                                    } Θ(1)
    Q : PrSor(G.V, kulcs alapján)               } Θ(n)
    Q ≠ <>                                      } Θ(n)
        u := Q.MIN_KIVESZ()                     } n*O(lg(n) )
        ∀ v ∈ G.Adj[u]                          } Θ(n+e)
            v ∈ Q ∧
            v.c > w(u,v)                        } Θ(e)
            v.c := w(u, v)                      } O(e) + O((e)*lg(n)) a kupac
            v. π := u                             adminisztrációja, a kulcs váltás miatt
```



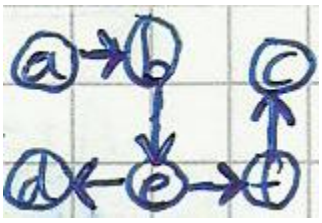
Lehet, hogy az u hozzávétele után v közelebb kerül a részleges feszítőfához.

Az egész algoritmus műveletigénye: O((n+e)*lg(n)), de a gráf összefüggő (e ≥ n-1), azaz O(e*lg(n))

kezdő csúcs: a (csak a Q-beli csúcsok c értékeit írjuk ki)

lépés	Q, c						π					
	a	b	c	d	e	f	a	b	c	d	e	f
start	0	∞	∞	∞	∞	∞	∅	∅	∅	∅	∅	∅
a		2	∞	4	∞	∞		a		a		
b	-		6	4	3	∞			b		b	
e		-	6	1		2				e		e
d	-	-	6		-	2						
f			5		-				f			
c	nincs már több lehetséges szomszédja											
vége:	0	2	5	1	3	2	∅	a	f	e	b	e

-: a szomszéd már korábban belekerült a fába
1: új, jobb távolság



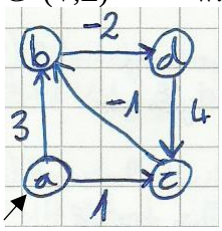
Az MST, amit kaptunk, irányított faként reprezentált IRÁNYÍTATLAN FA

prim optimalizációja:
Fibonacci kupacokkal le lehet csökkenteni a műveleti igényt: O(e+n*lg(n))

111_2 EA

Legrövidebb út probléma

Adott egy tetszőleges hálózat, amiben az utaknak költségeik vannak, ezt a hálózatot egyszerű gráffal ábrázoljuk.
G=(V,E) w: E → ℝ



Negatív költség is lehetséges. Adott pontból az összes többi csúcsba meg szeretnénk határozni a legrövidebb utat a költségekkel, nem biztos, hogy a legkevesebb élből álló út a legrövidebb.

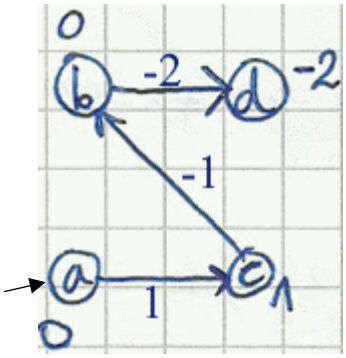
Mikor értelmes a kérdés:

- ha létezik a start csúcsból elérhető negatív kör, nincs legrövidebb út egyetlen a körből elérhető csúcsba sem.

Ha a startcsúcsból csak nem negatív kör érhető el, az az útból elhagyható.
Optimális útnak tetszőleges részútja is optimális út.

Általános algoritmus: (Robert Endre Tarjan: Data Structures and Network Algorithms könyvben: "Breadth-first Scanning" algoritmus)
„Queue-based BELLMAN-FORD” (Sor alapú Bellman-Ford algo)

d	a	b	c	d	Q π	a	b	c	d	menet
init	0	∞	∞	∞	<a>	∅	∅	∅	∅	∅
a		3	1		<b,c>		a	a		0.
b				1	<c,d>				b	1.
c		0			<d,b>		c			
d					<b>					2.
b				-2	<d>				b	
d					<>					3.
	0	0	1	-2		∅	c	a	b	



SZÉLESSÉGI_BELLMAN_FORD(G,s)

GKI(G,s)
Q := <s>
Q ≠ <>
u := Q.sorból()
∀v: (u,v) ∈ G.E
u.d + w(u,v) < v.d
v.d := u.d + w(u,v)
v. π := u
v ∉ Q
Q.sorba(v)

GKI(G,s)
∀u ∈ G.V
u.d := ∞
u. π := ∅
s.d := 0

a sorban levés ellenőrizhető pl. egy adattaggal, mindig bejárni a sort nem lenne hatékony.

Egy csúcs feldolgozása a fő ciklus magjának egyszeri végrehajtása, azaz, hogy kivesszük a sorból, és kiterjesztjük (feldolgozzuk a belőle kimenő éleket).

Menet rekurzív definíciója:

0. menet: a start csúcs (s) feldolgozása.
(i+1). menet: az i. menet végén a sorban levő csúcsok feldolgozása.

Tulajdonság:
∀u ∈ G.V, hogy ha az u csúcsba vezet k élből álló s → u optimális út ⇒ a k. menet elején már u.d = w(s → u) és (s, …, u.π.π, u.π, u) egy optimális út

Lemma: s-ből nem érhető el negatív kör (amely összköltsége negatív) ⇒ tetszőleges u elérhető csúcsra létezik s → u optimális út, ami legfeljebb n-1 élből áll.

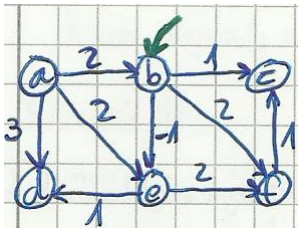
T(Lemma és Tulajdonság következménye):
Ha s-ből nem érhető el negatív kör
⇒ tetszőleges u elérhető (s-ből) csúcsra létezik s → u optimális út, és egy optimális út az n-1. menet elején már rendelkezésünkre áll (ki van számolva).
⇒ az n-1. menet végére kiürül a sor, az algoritmus O(n*e) időben megáll.

Következmény: Ha az (n-1). menet végén van a Q sorban elem ⇔ van s-ből elérhető negatív kör.

Megjegyzés: vegyünk egy ilyen w elemet. Ebből a π pointereken visszafele haladva megtalálható egy ismétlődő v elem, amire ∃ k ∈ 1..n $\pi^k(v) = v$, és $(v, \pi^{k-1}(v), \pi^{k-2}(v), \dots, \pi(v), v)$ egy negatív kör.

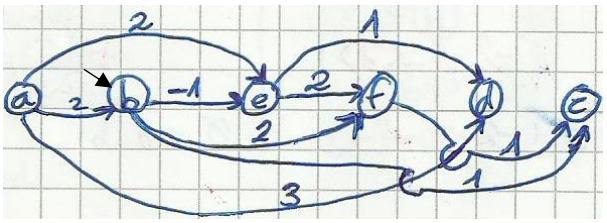
Jelölés: $\pi(v) = v.\pi$, valamint $\pi^k(v) = \pi(\dots(\pi(v)\dots)$, a π-t k-szor alkalmazva.

Legrövidebb utak egy forrásból, körmentes irányított gráfokon



feltétel: nincs irányított kör (a gráf DAG),
ekkor lehet topologikus rendezéssel optimális utakat keresni.

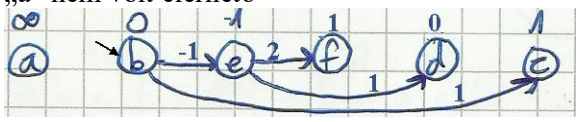
topologikus rendezés:



	d						π					
	a	b	c	d	e	f	a	b	c	d	e	f
init	∞	0	∞	∞	∞	∞	Ø	Ø	Ø	Ø	Ø	Ø
a	----------						----------					
b			1		-1	2			b		b	b
e				0		1				e		e
f												
d												
c												
vége:	∞	0	1	0	-1	1	Ø	Ø	b	e	b	e

nem történik semmi, el kell jutni a start csúcsba

„a” nem volt elérhető



DAG legrövidebb utak egy forrásból algoritmus:
1. a gráf csúcsainak topologikus rendezése,
2. a csúcsok kiterjesztése a start csúcstól (a forrástól) kezdve a topologikus sorrend szerint.

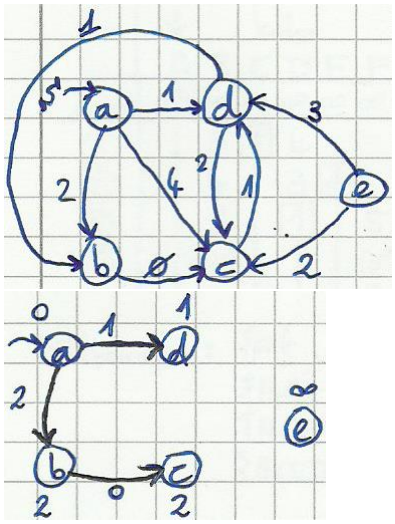
Műveletigény: topologikus rendezés: Θ(n+e) műveletigény,
a csúcsok kiterjesztése: O(n+e) műveletigény.
Teljes műveletigénye: Θ(n+e)

Megjegyzés: Ha a kiterjesztendő u csúcsra u.d=∞, akkor a kiterjesztés nem csinál semmit.
Ha u.d véges, akkor feldolgozza a kimenő éleket, azaz
minden (u,v), az u-ból kimenő élre,
ha u.d+w(u,v)<v.d akkor { v.d := u.d+w(u,v) ; v.π:=u }

1000_2 EA

Legyen G=(V,E) akár irányított, akár irányítatlan,
w: E → $\mathbb{R}_0$ nem negatív valós számok

DIJKSTRA algoritmus



Q, d	a	b	c	d	e	kiterjesztés π	a	b	c	d	e
init:	0	∞	∞	∞	∞		∅	∅	∅	∅	∅
		2	4	1	∞	a		a	a	a	
		2	3		∞	d			d		
			2		∞	b			b		
					∞	c					
						e					
vége:	0	2	2	1	∞		∅	a	b	a	∅

```
Dijkstra(G,w,s)
    ∀u ∈ G.V                                   } Θ(n)
        u.d := ∞
        u. π := ∅
    s.d := 0                                   } Θ(1)
    Q := G.V                                   } Θ(n)
    Q ≠ <>                                     } d szerint n-szer Θ(n)
        u:= Q.MINKIVESZ_d()                    } e < n^2
        ∀v: (u,v) ∈ G.E                        } e*O(1)
            v.d > u.d + w(u,v)
                v.d := u.d + w(u,v)
                v. π := u
                Q.HELYREÁLLÍT_d(v)
```

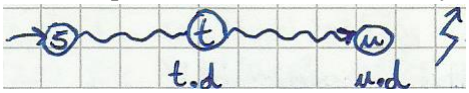
T(n) ∈ Θ(n^2) ha a Q rendezetlen

alsó index d azt jelenti, hogy a minkivesz a d értékei szerinti szomszédsági éllista + bin. kupac esetén MT(n,e) ∈ O((n+e)*log n)

- Q.MINKIVESZ$_d$(): n* O(lg n)
- Q. HELYREÁLLÍT$_d$(v): O(e * lg n)

Áll: Amikor a csúcsot kivesszük a sorból, oda már optimális út vezet.
Bizonyítás:

start csúcsra ✓, a költség optimális. Tegyük fel, hogy nem igaz az Állítás! Legyen u az első ilyen csúcs, t pedig az s→u optimális út első csúcsa amely ∈ Q sornak.



u-ra nem igaz, u-t válasszuk kiterjesztésre, ekkor nincs talált optimális út (de attól még létezik)

u.d > w(s $\overset{opt}{\longrightarrow}$ u), mivel egyébként megtaláltuk volna az optimális utat.

t.d = w(s $\overset{opt}{\longrightarrow}$ t) ≤ w(s→u) < u.d

az s→ t optimális út, mivel optimális út részútja.

=: hiszen „t”-t megelőző csúcs kiterjesztésekor beállítottuk, de mivel t.d < u.d, ezért a következő 1épés „t”-t választja kiterjesztésre ↯

Minden csúcsból minden csúcsba legrövidebb út minden csúcsra futtatott Dijkstrával:

- ritka gráfra O(n*(n+e)*log n) = O(n^2*log n)
- sűrű gráfra O(n^3*log n), nem jó választás…
    - ehelyett a vektorpáros Dijkstra stabil O(n^3)-öt (n * O(n^2)) fut.

Floyd–Warshall - algoritmus

Engedjük meg a negatív élsúlyt, de ne lehessen negatív kör.
G=(V, E)
G: V = 1..n
csúcsmátrixos ábrázolás

D_{ij} := i→ j optimális út költsége (végtelen, ha nincs optimális út)
π_{ij} := i→ j optimális úton j szülője
$D_{ij}^{(k)}$ = az i→ j úton az [1..k] indexű csúcsok lehetnek csak közbenső csúcsok.

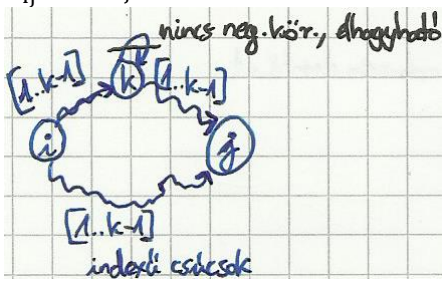
$\min\{w(i \xrightarrow[V|]{k} j)\}$

$\pi_{ij}^{(k)}$ = egy ilyen [1..k] közbensős úton a j csúcsok szülője

$$D_{ij}^{(0)} = \begin{cases} 0, \text{ ha } i = j & \text{(közvetlen út szülője)} \\ w(i,j), \text{ ha } (i,j) \in G.E \wedge i \neq j \\ \infty, \text{ ha } i \neq j \wedge (i,j) \notin G.E \end{cases}$$

$$\pi_{ij}^{(0)} = \begin{cases} \emptyset & \text{, ha } i = j \\ i & \text{, ha } i \neq j \text{ és } (i,j) \in G.E \\ \emptyset & \text{, különben} \end{cases}$$

$D_{ij}^{(n)} = D_{ij}$
$\pi_{ij}^{(n)} = \pi_{ij}$



ha $D_{ij}^{(k-1)} > D_{ik}^{(k-1)} + D_{kj}^{(k-1)}$, akkor $D_{ij}^{(k)} = D_{ik}^{(k-1)} + D_{kj}^{(k-1)}$

különben: $D_{ij}^{(k)} = D_{ij}^{(k-1)}$

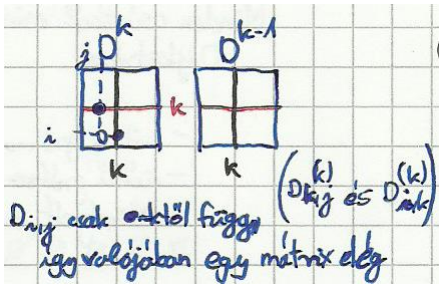
hasonlóan $\pi_{ij}^{(k)}$ is ha nem tudtuk az utat javítani = $\pi_{ij}^{(k-1)}$

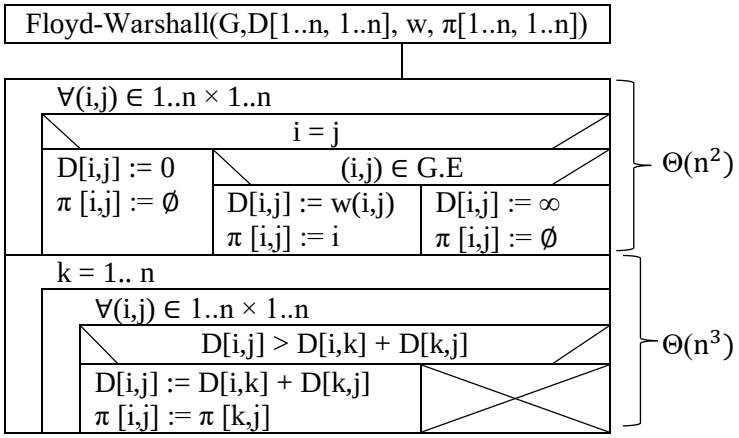
ha tudtuk = $\pi_{kj}^{(k-1)}$

$D_{ik}^{(k)} = D_{ik}^{(k-1)}$
$D_{kj}^{(k)} = D_{kj}^{(k-1)}$

Az algoritmus lépésenként előállítja mátrixpárok sorozatát.
$D^{(k)} = \left(D_{ij}^{(k)}\right)_{i,j=1}^{n}$ k = 0,.., n-ig, n+1 db mátrix.

D és π k-adik sora és oszlopa a (k-1) és k. lépésben egyenlő:
D i.-edik sora: i. csúcsból optimális utak hossza.





$MT_{FW}(n) \in \Theta(n^3)$

Összehasonlítás n*Dijkstra-val (azaz a Dijkstra algoritmust n-szer végrehajtjuk, mindig más start csúcsot választva, és az eredményeket összegezve):
Sűrű gráfokon ($e \in \Theta(n^2)$) az aszimptotikus műveletigény, mint n*Dijkstra (a prioritásos sort rendezetlen tömbbel megvalósítva), de ritka gráfokon ($e \in O(n)$) az n*Dijkstra (a prioritásos sort minimum kupaccal megvalósítva) aszimptotikusan jobb (feltéve hogy $\nexists$ negatív él, és így a Dijkstra is alkalmazható):
n*Dijkstra-ra ritka gráf esetén: MT(n,e) ∈ O(n*(n+e)*log n) = O(n^2*log n).

Megjegyzés: A Floyd-Warshall algoritmusnál negatív kör/körök esetén a D mátrix fő-átlójában negatív számok jelennek meg.

Tranzitív lezárt algoritmus (Warshall algoritmus)

Definíció: G=(V,E) gráf tranzitív lezártja T ⊆ V x V, ahol (u,v) ∈ T ⇔ G-ben van út u-ból v-be.

Feltesszük, hogy G csúcsait az 1..n számokkal címkéztük (V=1..n). Így T tekinthető egy n*n-es logikai mátrixnak. Ezt a mátrixot számítja ki a Warshall algoritmus.

T_{ij}: van-e út i-ből j-be (megállapítja, hogy van-e a két csúcs között út (csak az számít, hogy van-e, hogy milyen költségű az nem számít))

$T_{ij}^{(k)} \Leftrightarrow i \xrightarrow{k\vee|} j$ megint csak legfeljebb az 1..k csúcsokat érintő út.

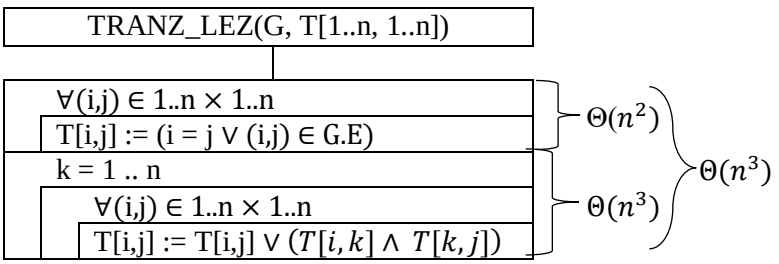
$T_{ij}^{(0)} =$ ↑ ha i = j ∨ (i,j) ∈ G.E
↓ különben

$T_{ij}^{(k)} = T_{ij}^{(k-1)} \vee \left(T_{ik}^{(k-1)} \wedge T_{kj}^{(k-1)}\right)$

$T_{ij}^{(n)} = T_{ij}$

$T_{ik}^{(k)} = T_{ik}^{(k-1)}$

$T_{kj}^{(k)} = T_{kj}^{(k-1)}$

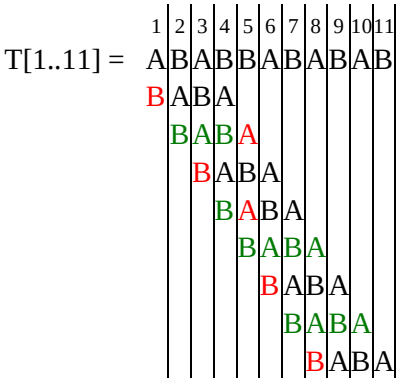


Sűrű gráfokra sokkal jobb, mint n db szélességi bejárás.
Ritka gráfokra (e ∈ O(n)) n darab szélességi bejárás max. műveletigénye: $\Theta[n * (n + e)] = \Theta(n^2)$ ($e \in O(n)$)
aszimptotikusan kisebb, mint a $\Theta(n^3)$ Warshall algo-nál.

Sűrű gráfra: $MT_{n*BFS}(n) \in \Theta\big(n * (n + e)\big) = \Theta(n^3)$. ($e \in \Theta(n^2)$)

Ez hasonló a Warshall-hoz,
de annak kisebb a konstans szorzója.

Mintaillesztési feladat



P[1..4] = BABA

piros betű: annál a betűnél található az első nem egyező karakter

zöld betű: egyező karakter

zöld szó: a keresett minta megtalálható a szóban

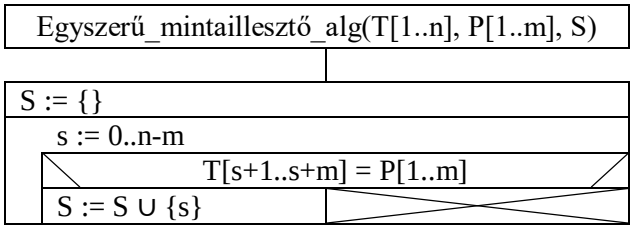
s = 4 eltolás

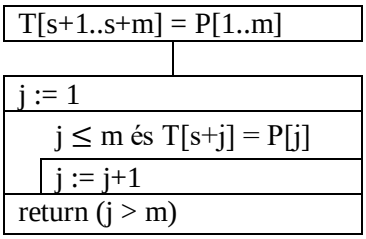
két találat

s = 6 eltolás

Fa: T[1..n]: Σ; P[1..m]: Σ $0 < m \leq n$

S := { s ∈ 0..n-m | T[s+1..s+m] = P[1..m]} programozása





O(m)

MT(m) ∈ Θ(m)

mT(m) ∈ Θ(1)

Az egész eljárás műveletigénye:

MT(n,m) ∈ Θ((n-m+1)*m) = Θ((n-m)*m)

mT(n,m) ∈ Θ(n-m+1) = Θ(n-m) n > m esetén

Megjegyzés:

Gyakran feltehető: n ≫ m

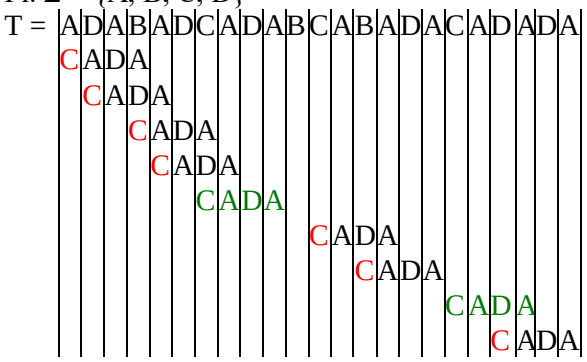
Ekkor: MT(n,m) ∈ Θ(n*m)

mT(n) ∈ Θ(n)

Elég jónak tűnhet, de nagy inputra hamar elszáll, kell ennél jobb algoritmus.

QUICK SEARCH

Alapfilozófiája példán illusztrálva:
Pl: Σ = {A, B, C, D}



P = CADA
piros betű: a keresett minta nem található
zöld szó: a keresett minta megtalálható a szóban

⊛

eltoljuk annyival, hogy a szövegnek a mintán túli első karaktere (jel.X) illeszkedjen a mintában az (X) karakter jobb szélső előfordulására.
ha mintánkon nem szereplő input karakter jön, teljes hosszon átugorjuk, hiszen úgysem illeszkedne

ABCD
shift: 1 5 4 2 adott betű esetén annyit kell lépni, hogy a minta illeszkedhessen, ezek a mintához specifikusak

```
Init(shift[Σ] : 1..n+1, P[1..m] : Σ)
    ∀δ ∈ Σ                          } Θ(d)
        shift[δ] :=m + 1
    j = 1 .. m                      } Θ(m)
        shift[P[j]] := m+1 - j
```

először feltesszük, hogy Σ ábécé összes betűjét át kell ugrani.

j	shift[P[j]]
1	m
2	m-1
3	m-2
	…

⊛

hogy a P[1] kerüljön X=T[s+m+1] alá, a mintát a teljes hosszával el kell tolni, hogy P[2] kerüljön az X alá, csak egyel, kevesebbet kell tolni, stb…

```
QuickSearch(T[1..n]: Σ; P[1..n]: Σ; S)
    Init(shift[Σ], P[1..m])          } Θ(m+ |Σ|) = θ(m+d) = Θ(m)   (d konstans)
    s := 0                           } Θ(1)
    S := {}
    s + m ≤ n                        } mT(n, m) ∈ Θ(n/(m+1)) ← pl: T és P diszjunkt
        T[s+1..s+m] = P[1..m]        } MT(n,m) ∈ Θ((n-m+1)*m) ← pl: T= A…A és p= A…A
        S :=S ∪ {s}
        s+m < n
        s := s+shift[T[s+m+1]] | s := s + 1
```

RABIN-KARP

$\Sigma = \{\sigma_1, \dots, \sigma_d\}$
$\varphi: \Sigma \rightarrow 0..d-1$
$\varphi(\sigma_i) = i-1$

$P[1..m] \sim \overbrace{\sum_{j=1}^{m} \varphi(P[j]) d^{m-j}}^{P_0}$: d számrendszerbeli szám, Horner elrendezéssel: Θ(m) időben számítható.
$T[s+1..s+m] \sim T_s = \sum_{j=1}^{m} \varphi(T[s+j]) d^{m-j}$: Horner elrendezéssel: Θ(m) időben számítható.

A keresett eltolásokat $S := \{s \in 0..n-m \mid T_s = P_0\}$ alakban oldjuk meg, a d-számrendszerű számok egyenlőségét vizsgáljuk.
A számítás a Horner mellett tovább egyszerűsödik:

$$T_{s+1} = \sum_{j=1}^{m} \varphi(T[s+1+j])d^{m-1} = \left(T_s - \varphi(T[s+1])d^{m-j}\right)d + \varphi(T[s+m+1]) : \Theta(1) \text{ időben,}$$

ha d^{m-1}-et előre kiszámoltuk.

T_s=|1 2 8 25|
T_{s+1} = |2 8 2514|

T_0 és P_0 számítása $\Theta(m)$ és $\Theta(m)$ idejű a rekurzió $\Theta(1)$ idejű.
A teljes műveletigény így $\Theta(m) + (n-m) * \Theta(1)$, ami $\Theta(n)$ igényű.
ha a rekurzió tényleg $\Theta(1)$ és a T_i, P_0 számokat a gép tudja hatékonyan ábrázolni.

Mi lesz a nagy mintákkal?
$p := P_0 \bmod q$ (q „nagy” prím)
$t_s := T_s \bmod q$
$h := d^{m-1} \bmod q$

$$t_{s+1} = (t_s - \varphi(T[s+1])h)d + \varphi(T[s+m+1]) \bmod q$$

a különbség átcsaphat negatívba…

$$t_{s+1} = \left(\left(\underbrace{(t_s + dq - \varphi(T[s+1])h)}_{<(d+1)*q} \bmod q\right) * d + \varphi(T[s+m+1])\right) \bmod q$$

$$\Longrightarrow t_s = \left(\left((t_{s-1} + dq - \varphi(T[s])h) \bmod q\right) * d + \varphi(T[s+m])\right) \bmod q =: \Delta$$

hogy jól működjön (d+1)q ≤ legnagyobb ábrázolható szám kell legyen (size_t, Integer’Last’…)
Lehet hamis találat (fals pozitív), ekkor ellenőrizni kell, hogy valódi találat-e, eredeti karakterekkel.

1010_2 EA

RABIN-KARP folytatása

$q = \frac{\max\mathbb{N}}{d+1}$ ← legnagyobb ábrázolható $\mathbb{N}$ szám, prímszám használatával a mod q számítások nem fognak túlcsordulni

$p \neq t_s \Rightarrow P_0 \neq T_s$

$p = t_s \not\Rightarrow P_0 = T_s$ így potenciális találat esetén a valódi találatot ellenőrizni kell.

a modulo számítás miatt elveszik az információ

```
RABIN_KARP(T[1..n], P[1..m], S)
t0 := φ(T[1])                                   ┐
p := φ(P[1])                                    │
h := 1                                          │
  j = 2..m                                      │ Θ(m)
    t0 := (t0 * d + φ(T[j])) mod q              │
    p := (p * d + φ(P[j])) mod q                │
    h := (h * d) mod q                          ┘
p = t0 ∧ T[1..n] = P[1..m]                      ┐ Θ(1)
S := {0}          S := {}                       ┘
  s := 1.. n - m                                ┐ mT(n,m) ∈ Θ(n − m)
    Δ                                           │   ha t_s = p sosem teljesül
    t_s = p ∧ P[1..m] = T[s+1..s+m]             │ MT(n,m) ∈ Θ((n − m) * m) ← pl: T = AA ... A p = A ... A
    S := S ∪{s}                                 ┘   ha a minta mindig megtalálható
```

teljes algoritmus:

$mT_{RK} \in \Theta(n)$

$MT_{RK} \in \Theta((n - m + 1) * m)$

A gyakorlatban nincs túl sok valódi találat, nagy q esetén a hamis találatok száma se túl sok.

Így $AT_{RK} \sim mT_{RK}$ (nem biz)

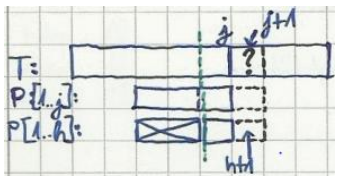
Mintaillesztés lineáris időben

Példa:

P[1..8] = BABABBAB

```
T[1..18]=ABABABABBABABABBAB
         B
          BABABBAB
            BABABBAB
              BABABBAB
                BABABBAB
                  BAB…(vége a szövegnek)
```

eltoljuk a mintát annyival, hogy a minta illeszkedő része az eltolás után illeszkedjen részben a szövegre



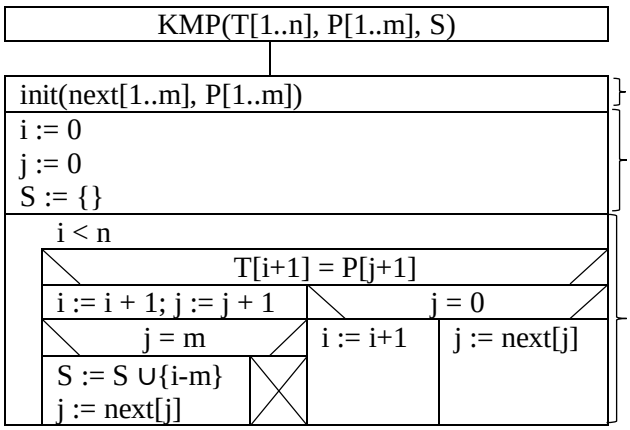
Keressük a legnagyobb olyan h-t, hogy P[1..h] suffixe T[1..i] és h < j

$T[1..i]_{\text{j hosszú suffixe}} = P[1..j] \Rightarrow T[1..i]_{\text{h hosszú suffixe}} = P[1..j]_{\text{h hosszú suffixe}} = P[1..h]$

ilyen feltételt kielégítő h-t keresünk.

Ez az információ csak a minta ismeretében megkapható.

Def: next(j) = max{h ∈ [0..j-1] | P[1..h] ⊐ P[1..j]}
next[1..m] := next(1..m)
Ezt kiszámolva úgy toljuk el a mintát, hogy p[1..next(j)] legyen a T[1..i] hosszú suffixe alatt, mivel korábban biztos nem lehet illeszkedés.



$0 \le i \le n \wedge 0 \le j < m \wedge i \ge j \wedge P[1..j] \sqsupset T[1..i] \wedge S = \{s \in 0..i-m \mid P[1..m] = T[s+1..s+m]\}$

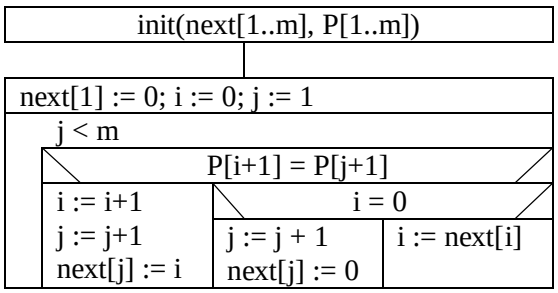
$next^k(j) = P[1..j]$ k-adik leghosszabb illeszkedő prefixe

A fő ciklus $mT(n) \in \Omega(n)$, mivel i növekedni egyesével tud, és n-ig nő $\Longrightarrow$ biztos van n lefutás
$t(i,j) = 2i - j \in [0..2n]$ szig mon növő mind a négy ágon, de így a főciklus legfeljebb 2n-szer fut le $\Longrightarrow$
$MT(n) \in O(n)$

$0 \le j \le i \le n$

$\Downarrow$

$MT_{fő}(n), mT_{fő}(n) \in \theta(n)$



$P[1..i] \sqsupset P[1..j]$

most $2j-i \in [2..2m]$, az ágakon szigorúan monoton növő, így a ciklus legfeljebb (2m-2)-ször iterál;
j=1-ről indul, és legfeljebb egyesével m-ig nőhet, így a ciklus legalább (m-1)-et iterál
$\Rightarrow$ a műveletigény $\Theta(m)$

Innét a KMP algoritmusra

$MT(n,m),\ mT(n,m) \in \theta(n+m)$

Továbbá $0 < m \le n$ miatt

$MT(n),\ mT(n) \in \theta(n)$

1011_2 EA

Folytatás init stuki:
Def: next: [1..m] → [0..m-1] ($m \in \mathbb{N}_+$)

next(j) = max{h ∈ [0..j-1] | P[1..h] ⊐ P[1..j]}

$x \sqsubset y \Leftrightarrow xz = y\ (\exists\ z)$
$x \sqsupset y \Leftrightarrow zx = y\ (\exists\ z)$
x ps-párosa y-nak ⇔ $x \neq y \wedge x \sqsubset y \wedge x \sqsupset y$ /*ps: prefix-suffix*/

Tulajdonságok:

1. $P[1..h]$ ps − párosa $P[1..j]$ − nek $\Leftrightarrow\ 0 \leq h < j \wedge P[1..h] \sqsupset P[1..j]$
2. $P[1..i+1] \sqsupset P[1..j+1] \Leftrightarrow P[1..i] \sqsupset P[1..j] \wedge\ P[i+1] = P[j+1]$
3. $0 \leq next(j) < j$ Köv: $j > next(j) > next^2(j) > \cdots > next^k(j) = 0$
4. $0 \leq next(j+1) \leq\ next(j) + 1$
5. valamely k > 0-ra
   a. $next^k(j)$ értelmezve van ⇔ ∃ a P[1..j]-nek a k. leghosszabb ps-párosa
   b. $next^k(j)$ értelmezve van ⇒ P[1.. $next^k(j)$] a P[1..j] k. leghosszabb ps-párosa

Q: m > 0 /*előfeltétel*/
R: next[1..m] = next(1..m) /*utófeltétel*/
Inv: $0 \leq i < j \leq m \wedge$ next[1..j] = next(1..j) ∧ P[1..i] ⊐ P[1..j] ∧ (a P[1..j] tetszőleges i-nél hosszabb P[1..l] ps-párosára: P[l+1] ≠ P[j+1] (0≤ i < j))

2j-i ∈ 2..2m
2-ről indul, szig. mon. nő
⇒ a ciklus legfeljebb (2n-2)-szer hajtódik végre
⇒ a ciklus legalább (n-1)-szer végrehajtódik

Példa:

j	1	2	3	4	5	6	7	8
P[j]	A	B	A	B	B	A	B	A
next[j]	0	0	1	2	0	1	2	3

Az algoritmus működése lépésről-lépésre:

			1	2	3	4	5	6	7	8
i	j	next[j]	A	B	A	B	B	A	B	A
0	1	0		A						
0	2	0			A					
1	3	1			A	B				
2	4	2			A	B	A			
0	4	üres mező					A			
0	5	0						A		
1	6	1						A	B	
2	7	2						A	B	A
3	8	3								

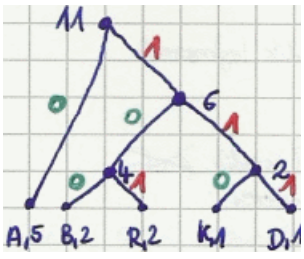
Tömörítés

Naiv módszer:
T[1..n]: Σ $\Sigma = \{\sigma_1, \ldots, \sigma_d\}$ ⌈lg d⌉ bittel kódolható a karakter
n * ⌈lg d⌉ bittel kódolható a szöveg

Huffman kód: egyfajra prefix kód

a szöveg: ABRAKADABRA

jel	előfordulás	kód
A	5	0
B	2	100
R	2	101
K	1	110
D	1	111



Megjegyzés: A kerekterenként kódoló tömörítések között a Huffman kód hossza minimális.

Tömörített kód: $\underbrace{0}_{A}\ \underbrace{100}_{B}\ \underbrace{101}_{R}\ \underbrace{0}_{A}\ \underbrace{110}_{K}\ \underbrace{0}_{A}\ \underbrace{111}_{D}\ \underbrace{0}_{A}\ \underbrace{100}_{B}\ \underbrace{101}_{R}\ \underbrace{0}_{A}$

Kitömörítés kódfa alapján

LEMPEZ-ZIV-WELCH (LZW) $\Sigma = \{a, b, c\}$

IN:	a	b	ab	c	ba	bab	a	aa	aaa	a
OUT	1	2	4	3	5	8	1	10	11	1

szó	kód
a	1
b	2
c	3
ab	4
ba	5
abc	6
cb	7
bab	8
baba	9
aa	10
aaa	11
aaaa	12

Rekonstruálás:

IN'	1	2	4	3	5	8	1	10	11	1
OUT'	a	b	ab	c	ba	bab	a	aa	aaa	a

8-as kód nincs a szótárban!
generáláskor a 8-as kód a ba… szövegből jött létre

$\underbrace{ba}_{5}\ bab \rightarrow \underbrace{bab}_{8}a..?$

(hasonlóan járunk el a 10-es, 11-es esetben is)

szó	kód
a	1
b	2
c	3
ab	4
ba	5
abc	6
cb	7
bab	8
baba	9
aa	10
aaa	11
aaaa	12

az utolsó sor dekódoláskor legtöbbször felesleges már

LZW_COMPRESS (In, Out, Σ)

D := $\{\underline{\sigma_1}: 1, \underline{\sigma_2}: 2, \ldots, \underline{\sigma_d}: d\}$
kód := d + 1
s := get_char(In)
¬eof(In)
  c := get_char(In)
  ($\widehat{sc}$:_) ∈ D
  | s := $\widehat{sc}$ | write(Out, kód(D,s))
D := D ∪{$\widehat{sc}$:kód} ⊙
s := c
kód := kód + 1 |
write(Out, kód(D, s))

$\Sigma = \{\sigma_1, \ldots, \sigma_d\}$ ábécé
s: string
c: char
D: szótár, „string: kód” alakú rendezett párosok halmaza
$\widehat{sc}$: az s string és a c char konkatenáltja
$\underline{\sigma_i}$: a σ_i betűből álló string (egybetűs string)
kód(D,s) = a D szótárban az s string kódja
eof(In) ⟺ az In inputról ∀ karaktert beolvastunk
„s:_” olyan rendezett páros, aminek az első komponense s
⊙ Az így jelzett utasítások csak akkor hajtandók végre, ha még kód ≤ MAXKÓD, ahol MAXKÓD a kódok előre rögzített hosszától függ.
Pl: ha ez 12bit, akkor MAXKÓD = 2^{12}-1 = 4095.

LZW_DECOMPRESS(In, Out, Σ)

D := $\{\underline{\sigma_1}: 1, \underline{\sigma_2}: 2, \ldots, \underline{\sigma_d}: d\}$
kód := d + 1
k := get_code(In)
s := string(D, k)
write(Out, s)
¬eof(In)
  k := get_code(In)
  (_:k) ∈ D
  | t := string(D, k)
write(Out, t)
D := D ∪ {$\widehat{st_1}$:kód}⊙ | t := $\widehat{ss_1}$
write(Out, t)
D := D ∪ {t:kód}⊗ |
  s := t
  kód := kód + 1

s,t: string
k, kód: kódok
t_1: a t string első betűje
string(D,k)= a D szótárban k kódnak megfelelő string
eof(In) ⟺ az In inputról ∀ kódot beolvastunk
„_:k” olyan rendezett páros, aminek a 2. komponense k
⊗: Itt k = kód teljesül

Megjegyzés: Balra, az elágazásban az „(_:k) ∈ D” feltétel helyén „k < kód” is állhatna (így egy kicsit gyorsabb lenne az algoritmus)

Felhasznált segédanyagok:

- Az előadáson készített jegyzetem (Koru),
- Whisperity előadás jegyzete (a jegyzetben található összes kép és néhány szöveges kiegészítés).
- http://aszt.inf.elte.hu/~asvanyi/ad/sor%20alapu%20Bellman-Ford%20alg.pdf
- http://aszt.inf.elte.hu/~asvanyi/ad/Lempel-Ziv-Welch%20alg.pdf

Külön köszönet Dr. Ásványi Tibor tanár úrnak a jegyzet alapos átolvasásáért, a talált hibák jelzéséért és az anyag megértését segítő megjegyzéseiért.

A jegyzetet átolvasták és a talált hibákat jelezték, amiért köszönet:

- Bán Róbert
- Fekete Anett
- László Tamás
- Nagy Vendel
- Szabó Gergő
- Szécsi Péter
- Tőkés Anna

FAQ:

>>
1. A Szélességi gráfkeresés a gráf mely csúcsaiba talál optimális utat, és a végrehajtás során mikor?
2. Mondja ki a biztonságos élekről és a minimális feszítőfákról szóló tételt! Definiálja a tételben szereplő vágás és könnyű él fogalmakat! a. Hogyan következik a Kruskal algoritmus helyessége ebből a tételből? b. Hogyan következik a Prim algoritmus helyessége ebből a tételből?
<<

1.
A "Szélességi gráfkeresés" a gráfnak a start csúcsból elérhető csúcsaiba talál optimális utat, amikor a végrehajtás során először eléri őket.

2.
Adott a G=(V,E) irányítatlan, összefüggő, w: E -> ℝ-rel élsúlyozott gráf.

Def1: G=(V,E), $\emptyset \subsetneq S \subsetneq V$ esetben (S, V\S) vágás a G gráfban
Def2: (u,v) él keresztezi az (S, V\S) vágást $\Leftrightarrow$ az $u \in S$ és $v \in V\S$ (vagy fordítva $u \in V\S$ és $v \in S$)

Def3: $A \subseteq E$ és (S, V\S) vágás esetén a vágás elkerüli az A-t $\Leftrightarrow$ A egyetlen éle sem keresztezi a vágást

Def4: (u,v) könnyű él az (S, V\S) vágásban, ha keresztezi a vágást, és
minden más, a vágást keresztező (x,y) élre w(x,y) >= w(u,v).

Def5: (u,v) él biztonságos az A-ra nézve $\Leftrightarrow$ $A \subseteq E$ valamely MST élhalmazának részhalmaza, (u,v) nem eleme A-nak, és $A \cup \{(u,v)\}$ is valamely MST élhalmazának részhalmaza.

Def6: Ha egy F fa élhalmaza valamely MST élhalmazának részhalmaza, akkor F-et MST részfának hívjuk.

Tétel: G = (V,E) irányítatlan w: E → ℝ-rel élsúlyozott gráf.

(S, V\S) vágás a gráfban elkerüli A-t, ahol A valamely MST élhalmazának részhalmaza, és

(u,v) egy könnyű él (legkisebb költségű) a vágásban.

Ekkor (u,v) biztonságos A-ra nézve, azaz A U {(u,v)} is valamely MST élhalmazának részhalmaza.

2.a, Prim algoritmus helyessége:

A Prim algoritmus helyessége ebből azért következik, mert az adott időpillanatig felépített MST részfa és a maradék gráf csúcshalmaza közti vágásban mindig könnyű él az, amit az MST részfához hozzávesz.

2.b, Kruskal algoritmus helyessége:

A Kruskal algoritmus fő ciklusának invariánsa, hogy (V,A) olyan erdő, aminek a fái MST részfák, és az adott időpontig feldolgozott élek mindegyike valamelyik fa két csúcsa között megy (vagy úgy, hogy a fa egy éle, vagy úgy, hogy kört képezne a fában, és ezért nem vettük hozzá A-hoz).

Először a fenti tétel segítségével belátjuk, hogy a fenti állítás valóban a fő ciklus invariánsa. Utána majd ebből következtetünk az algoritmus helyességére.

Ez az invariáns az első ciklusiteráció előtt nyilván igaz, mert mindegyik fa pontosan egy csúcsot tartalmaz, és még egyetlen élet sem dolgoztunk fel.

Ha egy adott időpontban igaz a fenti invariáns, és a következő feldolgozandó él az (u,v), akkor két eset van:

1. Az (u,v) él a (V,A) erdő valamelyik fájának két csúcsa között megy. Ekkor kört képezne a fában, és eldobjuk: így a fenti invariáns nyilván igaz marad.
2. Az (u,v) él a (V,A) erdő két fája között megy. Ekkor vegyük a (V,A) erdőben az u csúcsot tartalmazó MST részfát! Jelölje S ennek a fának a csúcshalmazát, és vegyük az (S,V\S) vágást! Ez a vágás elkerüli A-t, hiszen az egyik fa csúcshalmaza S, a többi fák csúcshalmazai pedig V\S részhalmazai. Az (u,v) él könnyű él a vágásban, ui. a vágást keresztező élek mindegyike a feldolgozatlan élek közül való, és az E rendezettsége miatt a feldolgozatlan élek között (u,v) minimális súlyú él. Így a fenti tétel szerint (u,v) biztonságos A-ra nézve, azaz a (V,AU{(u,v)}) is olyan erdő, aminek a fái MST részfák, csak az (u,v) él összekapcsolja a (V,A) erdő két fáját.

Ezzel a fenti invariánst beláttuk. A Kruskal algoritmus helyességéhez azt kell még igazolnunk, hogy legkésőbb az E utolsó élének feldolgozása után a (V,A) erdőnek már csak egy fája lesz.

A bizonyításhoz tegyük fel indirekt módon, hogy az E feldolgozása után a (V,A) erdőnek még legalább két fája van! Legyen az egyik ilyen fa csúcshalmaza S! Akkor az (S,V\S) vágást az invariáns szerint E egyetlen éle sem keresztezi, azaz a G=(V,E) gráf nem összefüggő, és ez ellentmondás. Q.E.D.