

Hálózattervezés Alapjai 2007

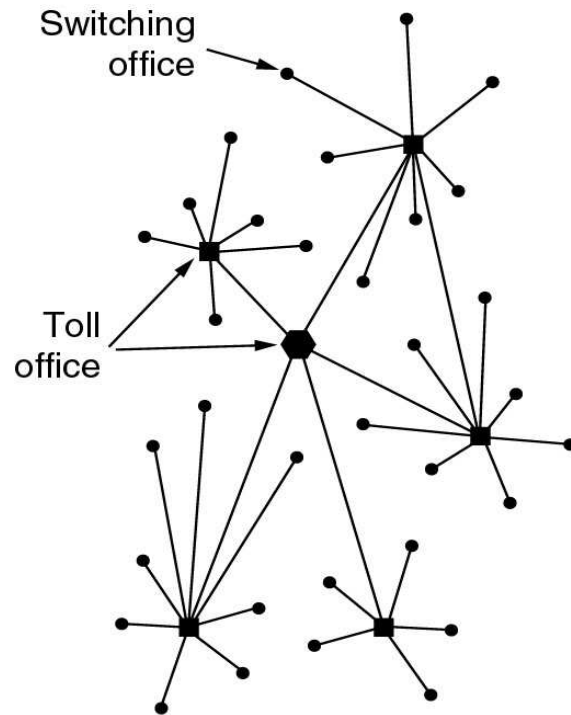
1: Alapok, minimális feszítőfák, legrövidebb utak

Az előadáshoz

- Előadás: Szerda 15:05-16:35 óra
- Vizsga írásbeli
- Honlap: <http://people.inf.elte.hu/lukovszki/Courses/G/07NWT>
- Irodalom:
 - J. Cheriyan, R. Ravi: Approximation Algorithms for Network Problems, Lecture Notes.
 - T. H. Cormen, C. E. Leiserson, R. L. Rivest: Introduction to Algorithms. MIT Press, 1990.
 - Aktuális publikációk

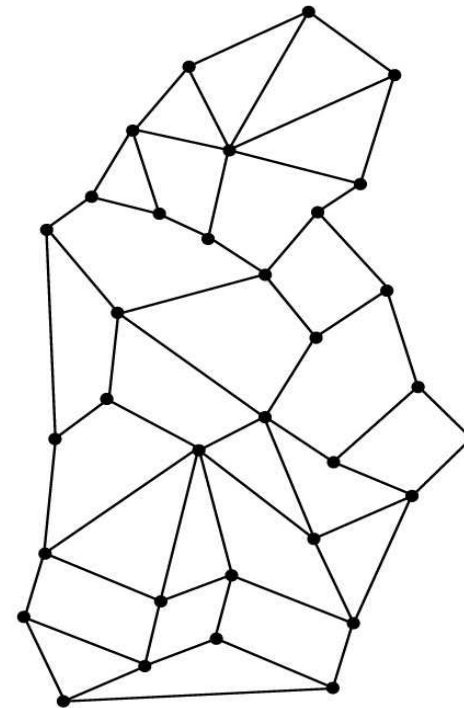
Hálózatok struktúrájának összehasonlítása

Hierarchikus telefon-hálózat



(a)

Az Internet



(b)

Tartalom

- Alapok: algoritmus elemzés, gráfok, minimális feszítőfák
- Multicasting – Steiner fák
- Access hálózat tervezés – könnyű, közelítően legrövidebb utak fája
- Általános hálózattervezési probléma – spanner gráfok
- Hálózatok hibatoleranciája – többszörös összefüggés, minimális vágás
- WDM optikai hálózatok – útvonalszinezés és routing
- Megfigyelési problémák – művészeti galéria probléma

Aszimptótika

Legyen $f, g : \mathbf{N}_0 \rightarrow \mathbf{R}$ két függvény.

- f majdnem mindig nem nagyobb g -nél, $f \leq_{\text{mm}} g$, ha $\exists n_0 \in \mathbf{N}_0 \ \forall n \geq n_0, n \in \mathbf{N}_0 : f(n) \leq g(n)$.
- $f(n) = O(g(n))$, ha $\exists c \in \mathbf{R} : f \leq_{\text{mm}} c g$. (c konstans, n -től független)
- $f(n) = \Omega(g(n))$, ha $\exists c \in \mathbf{R} : c g \leq_{\text{mm}} f$.
- $f(n) = \Theta(g(n))$, ha $f = O(g)$ és $f = \Omega(g)$.
- $f(n) = o(g(n))$, ha $\lim_{n \rightarrow \infty} f(n) / g(n) = 0$.
- $f(n) = \omega(g(n))$, ha $\lim_{n \rightarrow \infty} g(n) / f(n) = 0$.

Aszimptótika

Egy algoritmus komplexitása:

- Legyen P : probléma,
 I : input,
 A : algoritmus, ami megoldja P -t
- $T_A(I)$: az A algoritmus futási ideje I inputtal
- $S_A(I)$: az A algoritmus tárigénye I inputtal
- $T_A(n) := \sup \{ T_A(I) : |I| \leq n \}$
- $S_A(n) := \sup \{ S_A(I) : |I| \leq n \}$

Aszimptótika

P probléma komplexitása:

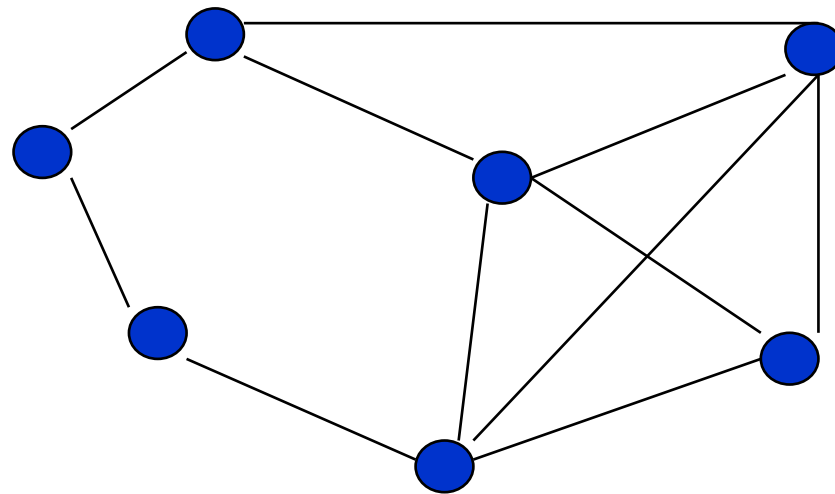
- $T(P) = O(f(n))$, ha $\exists A$ algoritmus, hogy A megoldja P -t és $T_A(n) = O(f(n))$
- $T(P) = \Omega(f(n))$, ha $\forall A$ algoritmus ami megoldja P -t : $T_A(n) = \Omega(f(n))$
- $T(P) = \Theta(f(n))$, ha $T(P) = O(f(n))$ és $T(P) = \Omega(f(n))$
- $T(P) = o(f(n))$, ha $\exists A$ algoritmus, hogy A megoldja P -t és $T_A(n) = o(f(n))$
- $T(P) = \omega(f(n))$, ha $\forall A$ algoritmus, ami megoldja P -t : $T_A(n) = \omega(f(n))$

Gráfok

- Kommunikációs hálózatok reprezentációja: Gráfok
- Gráf $G=(V,E)$
 - V : Csomópontok (csúcsok) halmaza
 - E : Élek halmaza
 - Amikor több gráffal dolgozunk, akkor egy G gráf csomópontjainak halmazát $V(G)$ –vel, éleinek halmazát $E(G)$ –vel jelöljük.
- Csomópont – Router, Switch, Host...
- Él – fizikai összeköttetés (Link)

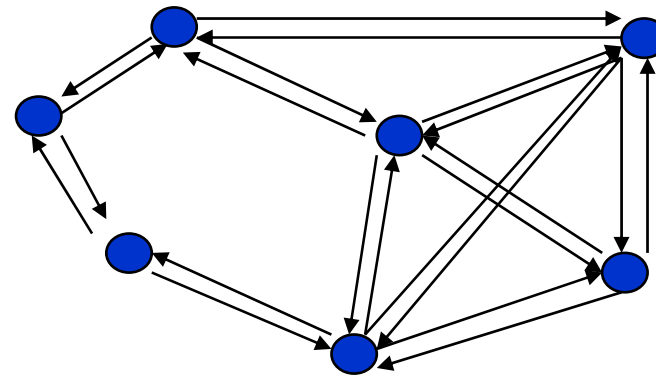
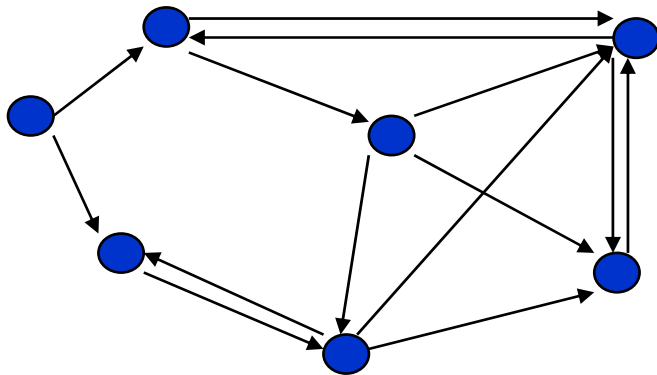
Írányítatlan gráfok

- Az élekhez nincs irány hozzárendelve
- Az éleket a csomópontok kételemű részhalmazai reprezentálják: $e=\{u,v\}$
- Azt mondjuk, hogy az él $e=\{u,v\}$ a u és v csomóponthoz **incidens**.
- Ha a csomópont u és v egy éllel össze van kötve, akkor azt mondjuk, hogy u és v **adjacens**.



Írányított gráfok (Digráfok)

- Minden élnek van egy kezdőpontja és egy végpontja
- Az éleket csomópontok rendezett párjával reprezentáljuk $e=(u,v)$.
- Egy v csomópont **ki-foka**: $|\{ e \in E: e=(v,w) \}|$
- Egy v csomópont **be-foka**: $|\{ e \in E: e=(u,v) \}|$
- Egy speciális eset: szimmetrikusan irányított gráfok:
 $(u,v) \in E \Leftrightarrow (v,u) \in E$.

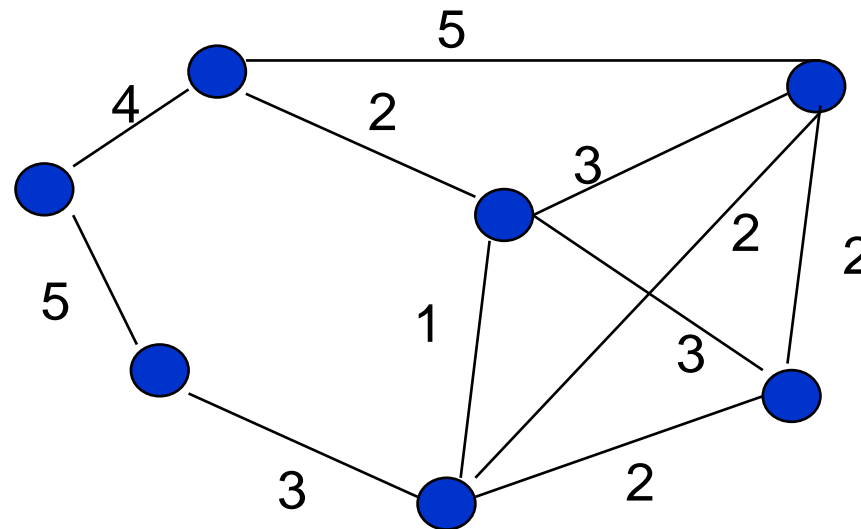


Gráfok reprezentációja

- A továbbiakban mindig $n=|V|$ a csomópontok száma és $m=|E|$ az élek száma a gráfban.
- G gráf szokásos reprezentációja:
 - A csomópontok listája
 - Az élek listája
 - Minden csomóponthoz egy lista a hozzá incidens élekről (irányított gráfoknál egy külön lista a bemenő és a kimenő élekről)
 - Tárigény: $O(n+m)$
- Alternatív: Adjazecia mátrix
 - $n \times n$ mátrix A
 - $a_{ij} = 1$, ha csomópont i és j össze vannak kötve egy éllel, egyebként 0.
 - Tárigény: $O(n^2)$

Súlyozott gráfok

- A csomópontokhoz ill. élekhez gyakran súlyok vannak rendelve.
Pl. minden élhez rendelhetünk egy kapacitást $c : E \rightarrow \mathbb{R}^+$ függvény által, ami a link sávszélességét adja meg.

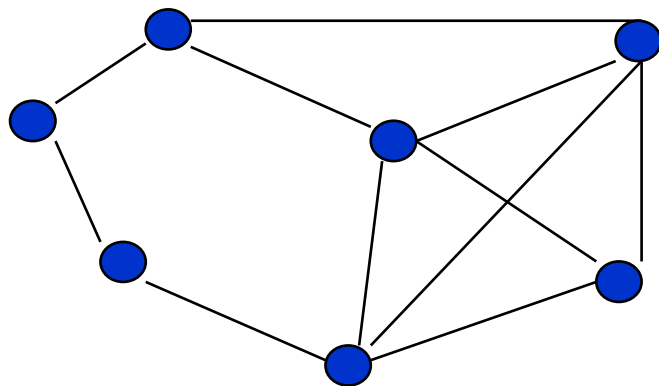


Út, kör

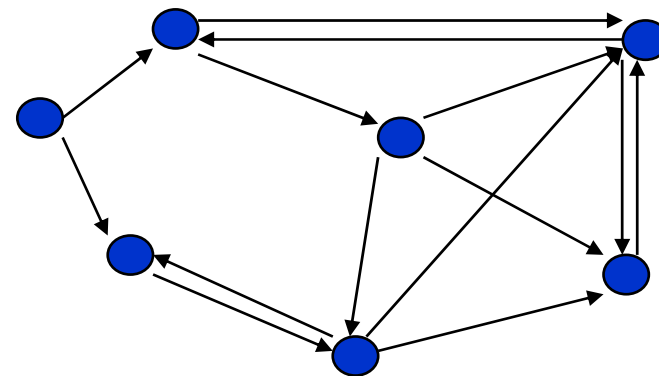
- Egy **út** u -tól v -hez egy $G=(V,E)$ gráfban ($u,v \in V$) egymást követő élek egy olyan sorozata $(x_0,x_1),(x_1,x_2),\dots,(x_{k-1},x_k)$, ahol $u = x_0$, $v = x_k$.
- Egy **egyszerű út** u -tól v -hez egy olyan út u -tól v -hez, ahol minden csomópont csak egyszer fordul elő.
- Egy P **út hossza** egy súlyozatlan gráfban az élek száma P -ben.
- Egy P **út hossza** egy súlyozott gráfban az élek súlyainak összege P -ben.
- Egy **kör (ciklus)** egy $G=(V,E)$ gráfban egymást követő élek egy olyan sorozata $(x_0,x_1),(x_1,x_2),\dots,(x_{k-1},x_k)$, ahol $x_0 = x_k$.
- Egy **egyszerű kör** egy olyan kör, amelyben minden csomópont csak egyszer fordul elő.

Összefüggőség, erős összefüggőség

- Egy irányítatlan gráf G **összefüggő**, ha G -ben minden csomópontból minden más csomóponthoz létezik egy út.
- Egy irányított gráf G **erősen összefüggő**, ha G -ben minden csomópontból minden más csomóponthoz létezik egy irányított út.



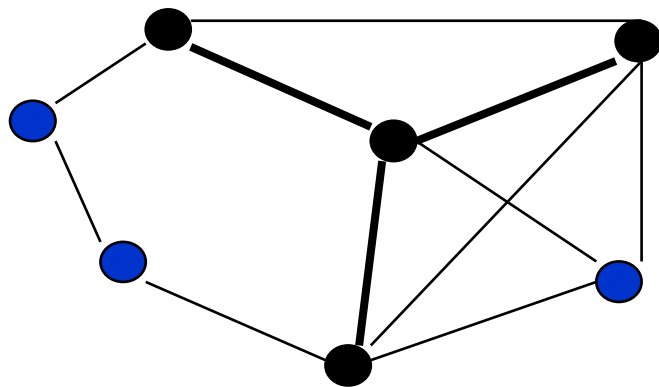
összefüggő



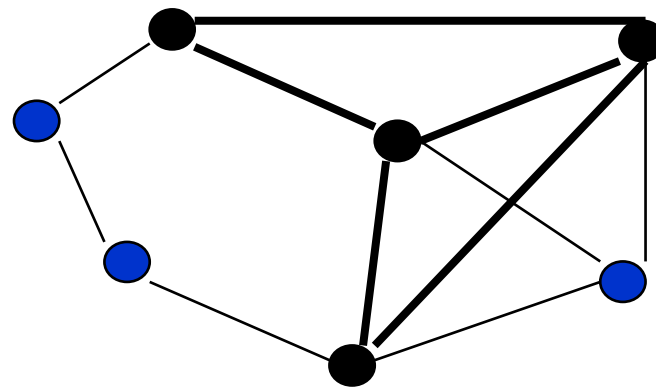
nem erősen összefüggő

Teljes gráf, részgráf, indukált részgráf

- Egy gráf **teljes**, ha a gráf minden lehetséges élet valóban tartalmaz.
- Egy $G'=(V',E')$ gráf **részgráfja** $G=(V,E)$ gráfnak, ha $V' \subseteq V$ és $E' \subseteq E$.
- G' **indukált részgráfja** G -nek, ha E' minden olyan élet tartalmaz E -ből, amely V' két csomópontját köti össze.



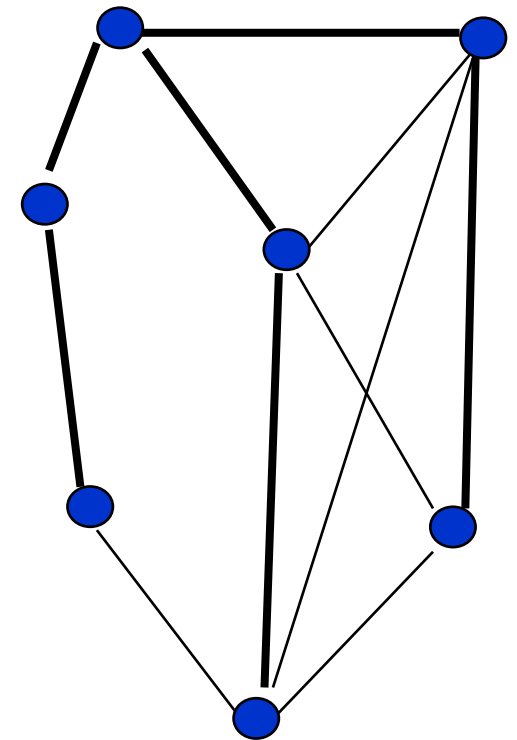
részgráf



indukált részgráf

Fák, Feszítőfák

- Egy irányítatlan gráf egy **fa**, ha összefüggő és nem tartalmaz kört.
- Egy irányítatlan $G=(V,E)$ gráf **feszítőfája** egy olyan fa, melynek csomóponthalmaza V és részgráfja G -nek.
- A csomópontokat, melyek foka 1, **leveleknek** nevezzük.
- Ha egy fa egy csomópontja ki van jelölve mint gyökér, akkor a fát **gyökeres fának** nevezzük.
- Egy gyökeres fában minden v csomópontnak a gyökér kivételével pontosan egy $p(v)$ **szülő** csomópontja van, amely v -hez adjacens és a v -től a gyökérhez vezető egyértelmű úton van. Minden más v -hez adjacens csomópontot v **gyermekének** nevezünk.
- Ha egy irányítatlan gráf nem összefüggő, akkor a maximális összefüggő részgráfjait a gráf **összefüggő komponenseinek** nevezzük.



Egy feszítőfa

Minimális Feszítőfa

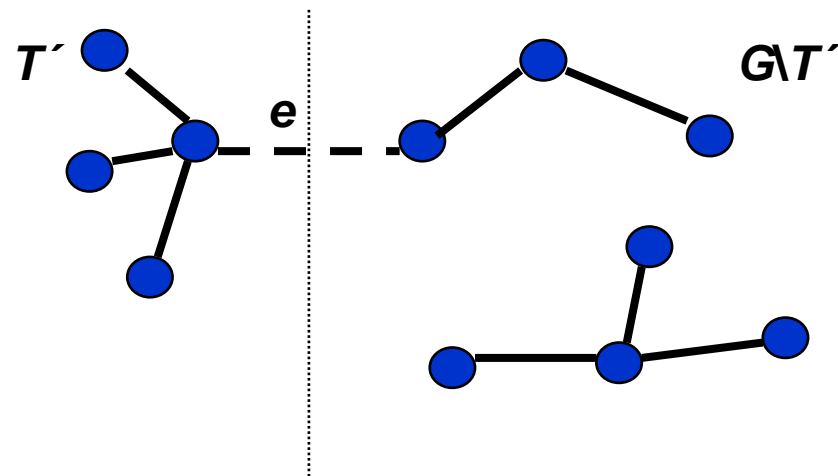
- **Probléma:**

Építsünk fel egy olyan kommunikációs hálózatot, amely az adott kommunikációs csomópontok halmazát összefüggően összeköti. Ehhez a csomópont párok között vezetékelt fektethetünk le, amelyek költsége a két végponttól függ. A cél, a csomópontokat minimális költségű vezetékkel összefüggően összekötni.

- Legyen $G(V, E)$ egy irányítatlan gráf súlyozott élekkel, ahol az élek súlyát egy $c : E \rightarrow \mathbf{R}^+$ függvény adja meg. Egy **minimális feszítőfa (MST)** G -ben egy olyan feszítőfa $T=(V, E')$, amelyre $c(T) := \sum_{e \in E'} c(e)$ minimális.

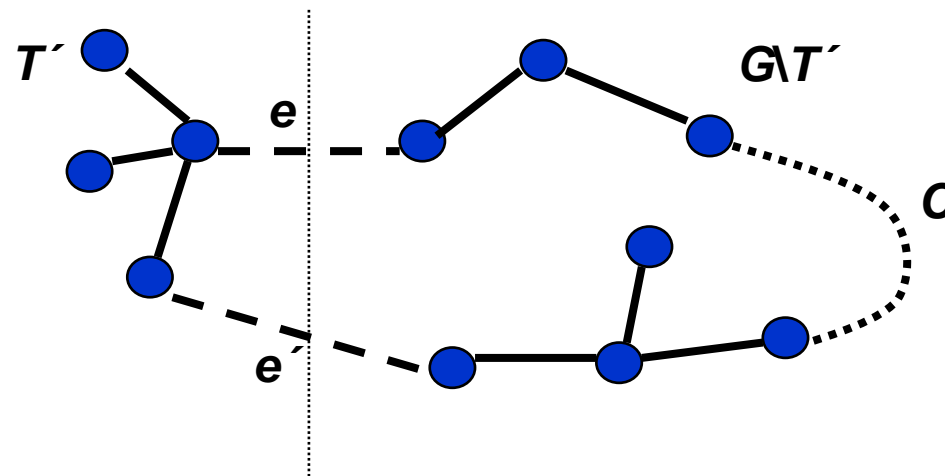
Egy minimális feszítőfa kiszámítása

Tétel 1: Legyen $E' \subset E$ egy olyan élhalmaz, hogy G -nek létezik egy T minimális feszítőfája, amely minden E' -beli élet tartalmaz. Legyen T' egy részfája T -nek, amely E' által adott. Legyen $e \in E$ egy él, melynek súlya minimális azon élek között, melyek egyik végpontja T' -ben, a másik $G \setminus T'$ -ben van. Ekkor létezik egy minimális feszítőfa, ami $E' \cup \{e\}$ minden élet tartalmazza.



Egy minimális feszítőfa kiszámítása

Biz.: Legyen T egy minimális feszítőfája G -nek, ami tartalmazza E' -t.
Ha T az e élet is tartalmazza, kész vagyunk. Tegyük fel, hogy $e \notin T$.
Akkor $T \cup \{e\}$ tartalmaz egy C kört, ami T -beli és $G \setminus T$ -beli csomópontokat is tartalmaz. Így C -nek tartalmaznia kell az e élen kívül legalább még egy e' élnet T és $G \setminus T$ között. Mivel e súlya minimális minden ilyen él között, $c(e) \leq c(e')$. Legyen $T'' = T \cup \{e\} \setminus \{e'\}$. Ekkor T'' egy feszítőfa, melyre $c(T'') \leq c(T)$, és T'' tartalmazza $E' \cup \{e\}$ -t. \square



Egy minimális feszítőfa kiszámítása

Tétel 1 implikálja, hogy egy minimális feszítőfát ki tudunk számítani úgy, hogy egy üres E' élhalmazzal indulunk és minden lépésben hozzádunk E' -hez egy olyan e élet, amely az összes él közül, ami az aktuális részfákból kivezet, minimális súlyú.

Kruskal algoritmus

Input: Egy összefüggő irányítatlan $G=(V,E)$ gráf $c : E \rightarrow \mathbf{R}^+$ élsúlyokkal

Output: G egy minimális feszítőfája $T=(V,E')$

$E' := \emptyset;$

for all $e=\{u,v\} \in E$ súly szerint növekvő sorrendben **do**

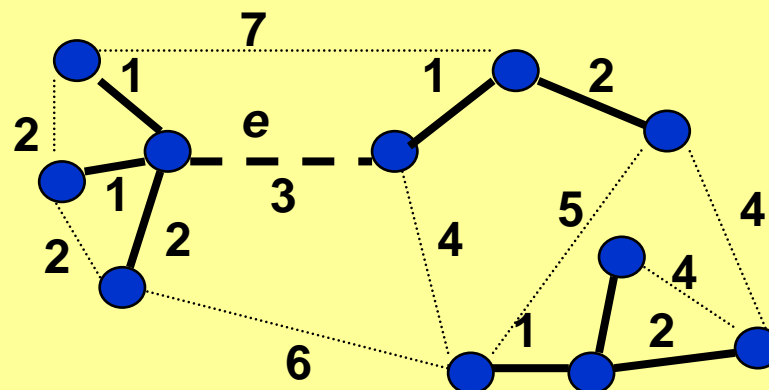
if u és v különböző összefüggő komponensében van $G'=(V,E')$ -nek **then**

$E' := E' \cup \{e\};$

fi;

od;

return $T=(V,E');$



Kruskal algoritmusának helyessége

Indukció:

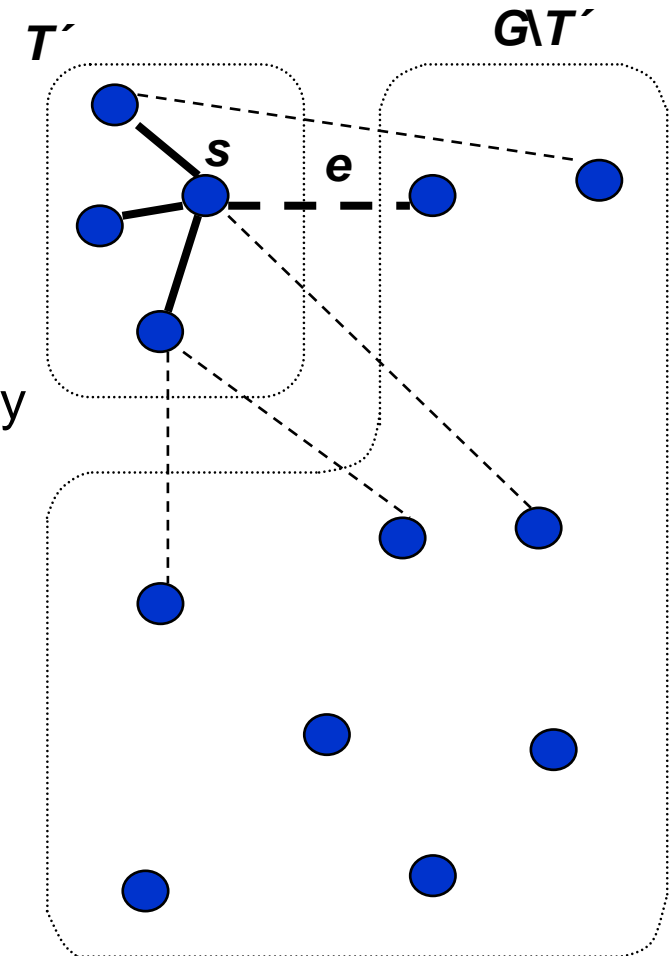
- $E' = \emptyset$: az üres élhalmazt minden minimális feszítőfa tartalmazza.
- Legyen E' az aktuális élhalmaz. Indukciós feltétel: E' -t tartalmazza egy minimális feszítőfa T . Legyen $e = \{u, v\}$ az az él, ami éppen feldolgozásra kerül. Ha u és v csomópont $T \cap E'$ ugyanazon T részfájában van, akkor $T \cup \{e\}$ tartalmaz egy C kört, úgy hogy e egy maximális súlyú él C -ben, mivel az élek súly szerint növekvő sorrendben kerülnek feldolgozásra. Következésképpen e nem kerül bele E' -be.
- Ha u és v csomópont $T \cap E'$ különböző összefüggő komponensében van, akkor $\{u, v\}$ egy minimális súlyú él, ami az u -t tartalmazó részfából kivezet, mivel az élek súly szerint növekvő sorrendben kerülnek feldolgozásra. Ekkor Tétel 1 szerint létezik egy minimális feszítőfa, ami az $E' \cup \{e\}$ élhalmazt tartalmazza. Így e belekerül E' -be.

Kruskal algoritmusának elemzése

- Az élek sorbarendezése: $O(m \log m) = O(m \log n)$ idő, $O(m)$ tár
- Tesztelni, hogy az éppen feldolgozásra kerülő él végpontjai ugyanabban az összefüggő komponensben vannak-e, és ha nem, akkor a két összefüggő komponens egyesítése: amortizáltan $O(\alpha(m,n))$ idő.
 - Union-Find adatstruktúra segítségével (R. Tarjan, 1979)
 - $\alpha(m,n)$ az Ackermann függvény egy funkcionális inverze:
 $\alpha(m,n) = \min\{ z : A(z, 4^{\lceil m/n \rceil}) > \log n \}$, ahol
 $A(i,0)=0, A(i,1)=2$ ha $i \geq 0$,
 $A(0,j)=2j$ ha $j \geq 0$,
 $A(i,j)= A(i-1, A(i,j-1))$ ha $i \geq 1, j \geq 2$.
- Összesen: $O(m \log n)$ idő, $O(n+m)$ tár

Prim algoritmus

- Kiválasztunk egy tetszőleges s csomópontot, mint kezdőpont.
- Mindig azt a T' részfat tekintjük, amely s -t tartalmazza.
- Az összes T' -ből kivezető él közül kiválasztunk egy minimális súlyút és hozzáadjuk E' -hez.
 - Ezáltal T' minden lépésben egy éllel és egy csomóponttal lesz nagyobb.
 - Minden időpontban csak egy T' részfa van, ami több mint egy csomópontot tartalmaz.
- Tétel 1 feltételei teljesülnek, így $n-1$ lépés után T' egy minimális feszítőfává nő.



Prim algoritmus

Hogyan lehet minden lépésben a hozzáadandó e élet hatékonyan meghatározni?

- e súlya minimális kell hogy legyen mindazon élek között, amelyek egy T -beli csomópontot egy $G \setminus T$ -beli csomóponttal kötnek össze.
- Ehhez egy u.n. Priority-Queue adatstruktúrát használunk.
Egy Priority-Queue Q a következő operációkat bocsátja rendelkezésre:
 - $Q.Insert(e,p)$: e elem hozzáadása p prioritással Q -hoz,
 - $Q.DecreasePriority(e,p)$: csökkenti a prioritását a Q -ban már tartalmazott e elemnek p -re,,
 - $Q.DeleteMin()$: visszaadja Q legalacsonyabb prioritású elemét és törli azt Q -ból.
- Alternatívák Q implementálására:
 - **Bináris kupac (binary heap)**: mindhárom operáció $O(\log n)$ idő alatt.
 - **Fibonacci Heap**: $Insert$ és $DecreasePriority$ $O(1)$ időben és $DeleteMin$ $O(\log n)$ időben amortizáltan.

Prim algoritmus

Invariánsok:

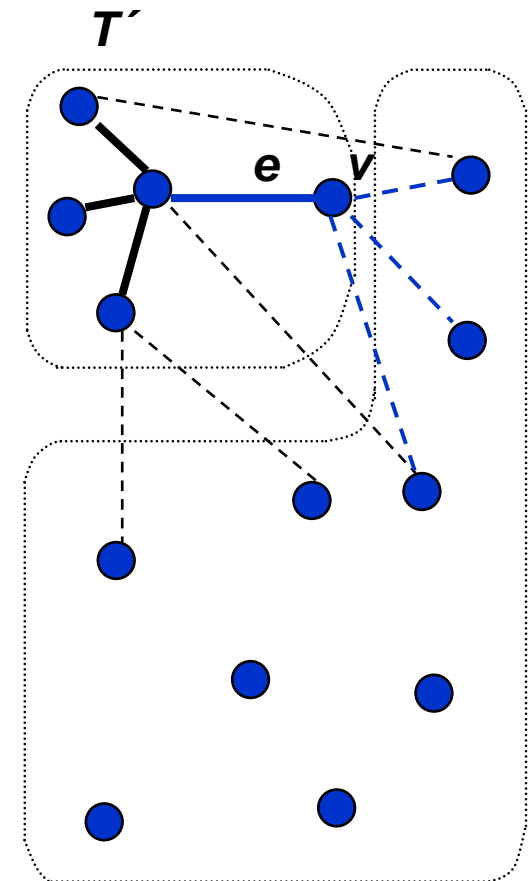
- Q tartalmazza az összes $v \in G \setminus T$ csomópontot, ami T -ből egy éllel elérhető. Legyen e_v egy minimális súlyú él azon élek közül, melyek egy T -beli csomópontot kötnek össze v -vel. Akkor v prioritása Q -ban: $c(e_v)$.
- Minden v csomóponthoz, ami Q -ban van, nyilvántartjuk egy $from[v]$ változóban a minimális súlyú e_v -t élet T und v között.

Inicializálás:

- A startcsomópont s minden v szomszédját beletesszük Q -ba $c(\{s, v\})$ prioritással és
- $e=\{s, v\}$ élet tároljuk $from[v]$ -ban.

Prim algoritmus

- A következő él, amit E' -hez hozzáadunk, a $Q.DeletMin()$ operációval határozhatjuk meg: ezzel megkapjuk azt a $v \in G \setminus T'$ csomópontot, amely T' -ből egy minimális súlyú éllel érhető el.
- Ezután az $e_v = from[v]$ él hozzáadjuk E' -hez.
- Ezután a következő adatokat kell aktualizálni, hogy az invariánsok érvényesek maradjanak (v mostmár T' -ben van):
 - v azon w szomszédjait, amik még nincsenek benne Q -ban, hozzá kell adni Q -hoz $c(\{v, w\})$ prioritással és $from[w]$ -ben tárolni kell $\{v, w\}$ -t.
 - v azon w szomszédjainál, amik már Q -ban vannak, és nagyobb a prioritásuk, mint $c(\{v, w\})$, csökkenteni kell a prioritást $c(\{v, w\})$ -re és $from[w]$ -ben tárolni kell $\{v, w\}$ -t.



Prim algoritmusa

Input: egy összefüggő irányítatlan
 $G=(V,E)$ gráf $c : E \rightarrow \mathbf{R}^+$ élsúlyokkal

Output: G egy minimális feszítőfája
 $T=(V,E')$

```
1.   $s :=$  tetszőleges startcsomópont  
     $V$ -ből;  
2.   $ready[s] := true$ ;  
3.   $ready[v] := false, \forall v \in V \setminus \{s\}$ ;  
4.  priority_queue  $Q$ ;  
5.  for all  $e=\{s,v\} \in E$  do  
6.     $from[v] := e$ ;  
7.     $Q.Insert(v,c(e))$ ;  
8.  od;  
9.   $E' := \emptyset$ ;  
10. while  $|E'| < |V| - 1$  do  
11.    $v := Q.DeleteMin()$ ;  
12.    $E' := E' \cup \{from[v]\}$ ;  
13.    $ready[v] := true$ ;  
14.   for all  $e=\{v,w\}$  do  
15.     if  $w \in Q$  and  $c(e) < c(from[v])$  then  
16.        $from[v] := e$ ;  
17.        $Q.DecreasePriority(w,c(e))$ ;  
18.     else if  $w \notin Q$  and not  $ready[w]$  then  
19.        $from[w] := e$ ;  
20.        $Q.Insert(w,c(e))$ ;  
21.     fi;  
22.   od;  
23. od;  
24. return  $T=(V,E')$ ;
```

Prim algoritmusa

Futási idő (Fibonacci Heap-pel):

- # $Q.Insert()$: n (csomópontonként 1) - $O(n)$ idő
- # $Q.DeleteMin()$: n (csomópontonként 1) - $O(n \log n)$ idő
- # $Q.DecreasePriority()$: $\leq m$ (élenként ≤ 1) - $O(m)$ idő
- # A teszt a 15. és a 18. sorban: m (élenként 1) - $O(m)$ idő
- Inicializálás: $O(n)$ idő
- Összesen: $O(n \log n + m)$ idő

Társzükséglet: $O(n+m)$

Megjegyzések

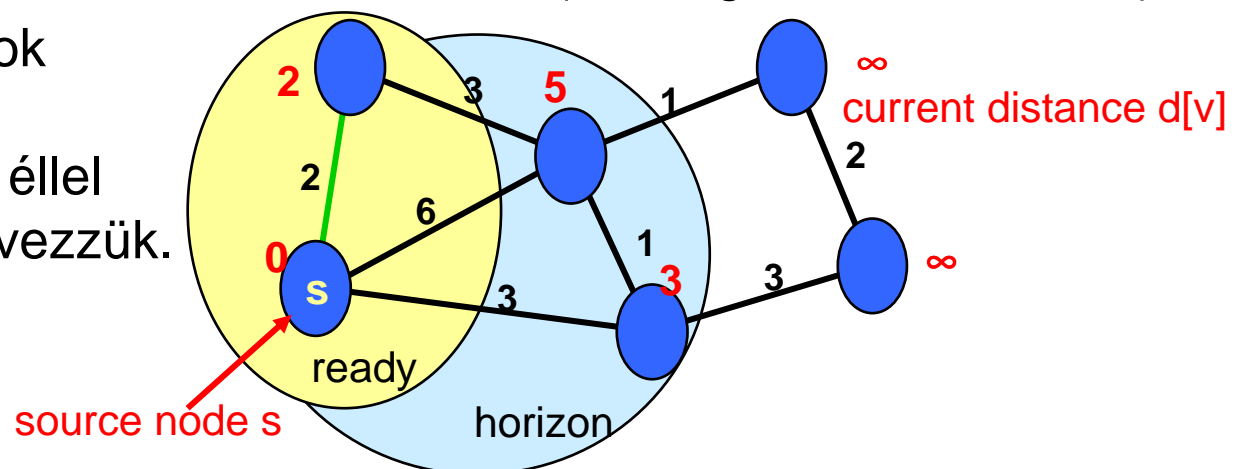
- Az aszimptótikusan leggyorsabb algoritmus a minimális feszítőfa kiszámítására $O(n\alpha(m,n) + m)$ időt és $O(n+m)$ tárat igényel [B. Chazelle 1998].
- A minimális feszítőfa kiszámításának időigényére ismert alsó korlát $\Omega(n+m)$.
- Nyitott Probléma: Meg lehet-e javítani az alsó vagy a felső korlátot?

Legrövidebb utak fája – single source shortest paths

- Adott:
 - Egy irányított gráf $G = (V, E)$, $c : E \rightarrow \mathbf{R}_{\geq 0}$ nem negatív élsúlyokkal
 - Kezdő csomópont $s \in V$
- Legyen
 - **P útvonal súlya $c(P) := \sum_{e \in P} c(e)$** az élek súlyainak összege P-ben
 - **u és v távolsága** G-ben, $u, v \in V$, egy legrövidebb út súlya G-ben u és v között : **$d(u, v) := \min\{ c(P) : P \text{ egy út } u\text{-tól } v\text{-hez } G\text{-ben}\}$.**
- Keressük:
 - egy legrövidebb utat s kezdő csomóponttól minden más $v \in V \setminus \{s\}$ csomóponthoz G-ben
 - Feltesszük, hogy minden $v \in V \setminus \{s\}$ elérhető s-ből. Nem elérhető csomóponthoz nem létezik legrövidebb út sem
- Megoldás:
 - Egy fa, melynek gyökere s és minden $v \in V \setminus \{s\}$ csomóponthoz tartalmaz egy legrövidebb utat s-től v-hez G-ben

Dijkstra algoritmus

- **Ötlet:** A legrövidebb utakat hosszuk szerint növekvő sorrendben számítjuk ki.
- Minden $v \in V$ csomóponthoz kiszámítjuk a következő értékeket:
 - **$d[v]$:** egy legrövidebb út hossza s -től v -hez,
 - **$pred[v]$:** a v -t megelőző csomópont egy legrövidebb úton s -től v -hez.
- Az algoritmus végrehajtása után az élhalmaz $\{ (pred[v],v) : v \in V \setminus \{s\} \}$ megadja egy legrövidebb utak fáját s gyökérrel G -ben.
- Egy v csomópontot „kész“-nek jelölünk: **$ready[v] = true$** , ha már meghatároztunk egy legrövidebb utat s -től v -hez (röv. legrövidebb s - v utat).
- A „nem kész“ csomópontok halmazát, amelyeket egy „kész“ csomópontból egy éllel elérünk, **horizont**-nak nevezzük.



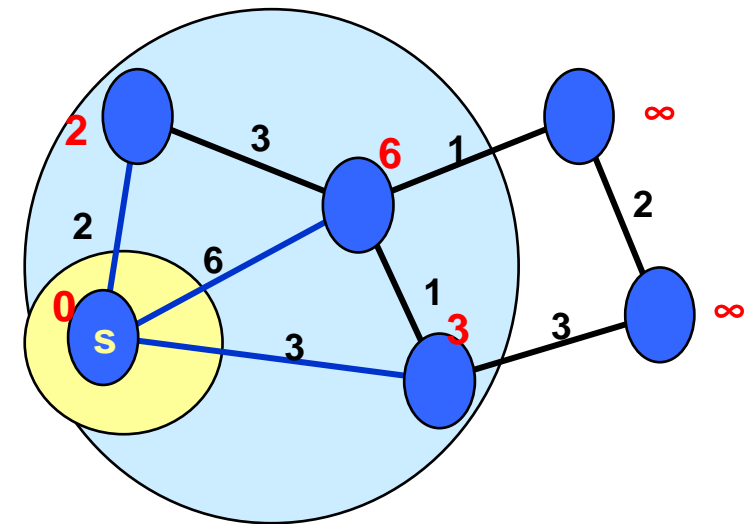
Dijkstra algoritmus

● Invariánsok:

- Minden horizont belüli csomópontot egy **Q priority-queue**-ban tárolunk, úgy hogy minden $v \in Q$ csomópontokra a következő érvényes:
 - $d[v]$ egy legrövidebb s - v út hossza mindazon utak között, melyek v -n kívül csak „kész” csomópontokat tartalmaznak,
 - $pred[v]$ a v -t megelőző csomópont egy ilyen úton,
 - v prioritása Q -ban $d[v]$

● Inicializálás

- $d[s] := 0$, $ready[s] := true$,
- s minden v szomszédjára:
 - $d[v] := c(s, v)$, $pred[v] := s$, $ready[v] := false$,
 - $Q.Insert(v, d[v])$.
- Minden $v \in V \setminus \{s\}$ csomópontokra:
 - $d[v] := \infty$, $ready[v] := false$.



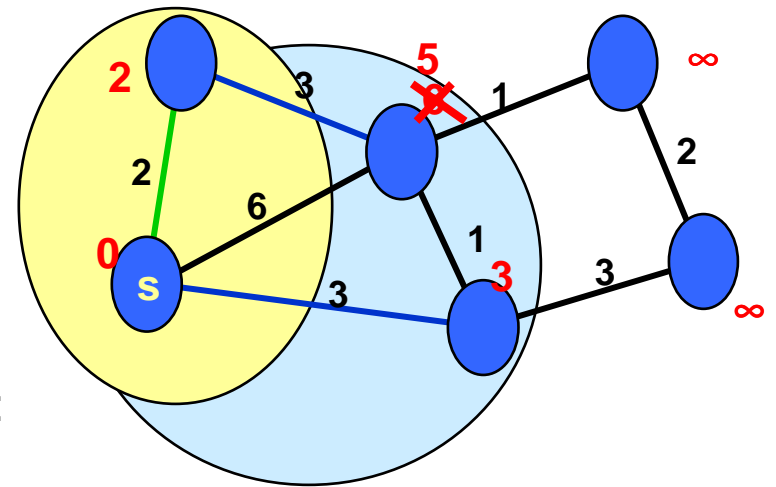
Dijkstra algoritmus

- Az invariánsok megőrzése egy iteráció után
 - Minden lépésben egy új csomópont lesz „kész”, egy csomópont v minimális prioritással.
 - $d[v]$ már tartalmazza a helyes értéket.

Mivel v minimális prioritású csomópont, minden olyan s - v út súlya, amely „nem kész” csomópontot is tartalmaz, legalább olyan nagy, mint annak az útnak a hossza, amit már megtaláltunk a csak „kész” csomópontokat tartalmazó utak között.
 - Legyen $Adj[v] := \{ u : (v,u) \in E \}$, $v \in V$, a v -hez adjacens csomópontok halmaza
 - minden $u \in Adj[v]$, ha $u \in Q$, meg kell vizsgálni, hogy s -től u -hoz direkt v -ből egy rövidebb út vezet-e, mint azok az utak, amik csak v -től különböző „kész” csomópontot tartalmaznak.

Ha igen, akkor aktualizáljuk

 - $pred[u] := v$ és $d[u] := d[v] + c(v,u)$,
 - csökkentsük u prioritását Q -ban.
 - minden $u \in Adj[v]$, ha $u \notin Q$ és u „nem kész”:
 - $pred[u] := v$, $d[u] := d[v] + c(v,u)$,
 - u -t be kell szűrni Q -ba $d[u]$ prioritással.



Dijkstra algoritmus

Dijkstra(G,s,c)

Output: egy legrövidebb utak fája
 $T=(V,E')$ G-ben s gyökérrel

```
01  $E' := \emptyset;$ 
02 ready[s] := true;
03 ready[v] := false;  $\forall v \in V \setminus \{s\};$ 
04 d[s] := 0;
05 d[v] :=  $\infty$ ;  $\forall v \in V \setminus \{s\};$ 

06 priority_queue Q;

07 forall v  $\in$  Adj[s] do
08   pred[v] := s;
09   d[v] := c(s,v);
10   Q.Insert(v,d[v]);
11 od

12 while Q  $\neq \emptyset$  do
13   v := Q.DeleteMin();
14    $E' := E' \cup \{(pred[v],v)\};$ 
15   ready[v] := true;
16   forall u  $\in$  Adj[v] do
17     if u  $\in$  Q and d[v] + c(v,u) < d[u] then
18       pred[u] := v;
19       d[u] := d[v] + c(v,u);
20       Q.DecreasePriority(u,d[u]);
21     else if u  $\notin$  Q and not ready[u] then
22       pred[u] := v;
23       d[u] := d[v] + c(v,u);
24       Q.Insert(u,d[u]);
25     fi
26   od
27 od
```

Dijkstra algoritmus

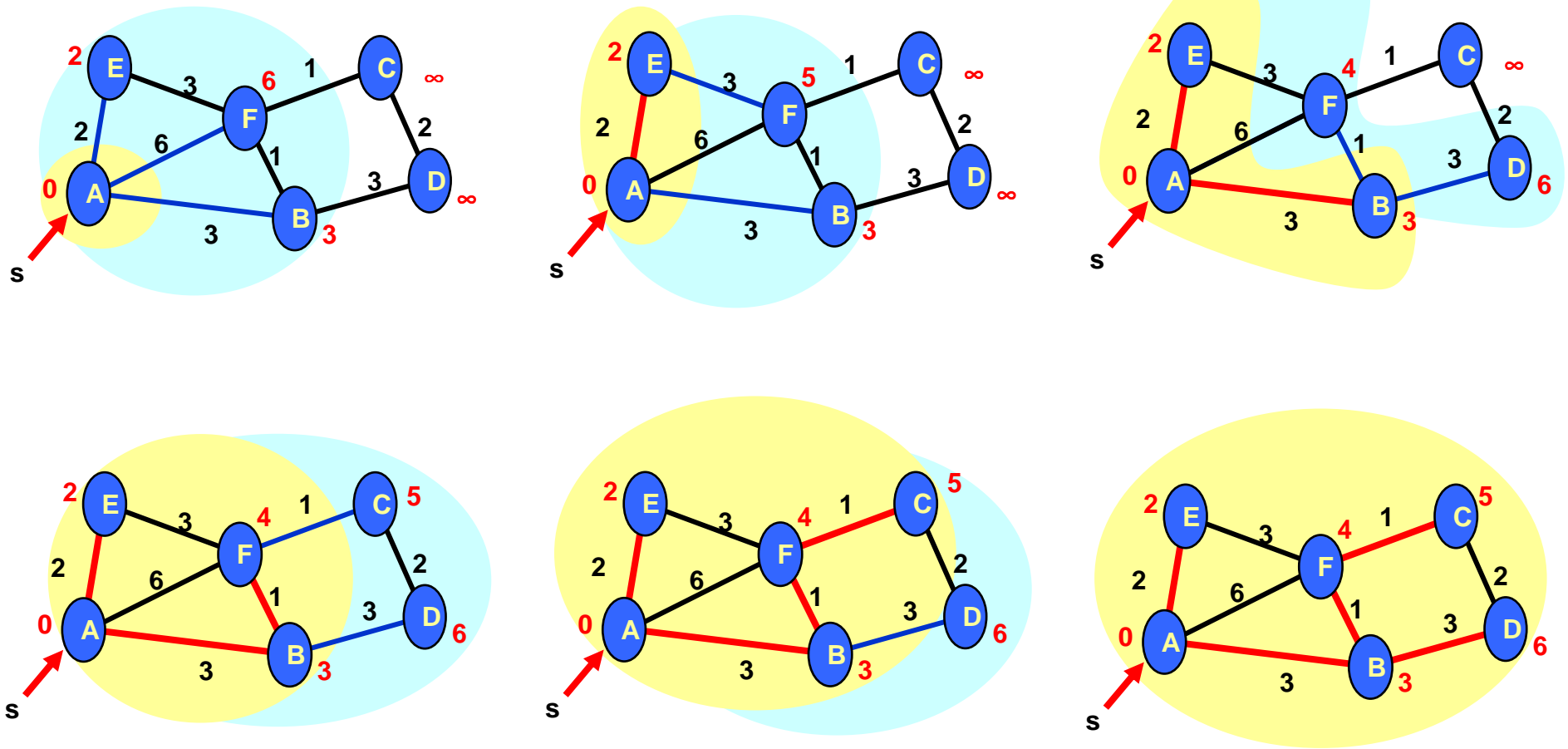
Futási idő (Fibonacci Heap-pel):

- # `Q.Insert()`: n (csomópontonként 1) -- $O(n)$ idő
- # `Q.DeleteMin()`: n (csomópontonként 1) -- $O(n \log n)$ idő
- # `Q.DecreasePriority()`: $\leq m$ (élenként ≤ 1) -- $O(m)$ idő
- # A teszt a 17. és 21. sorban: m (élenként 1) -- $O(m)$ idő
- Inicializálás: $O(n)$ idő
- Összesen: $O(n \log n + m)$ idő

Tárigény: $O(n+m)$

Dijkstra: Példa

szimmetrikusan irányított élek



OSPF (Open Shortest Path First) Routing

- Minden router tárol egy legrövidebb utak fát, a legrövidebb utakat saját magától minden cél-címhez.
- Startup:
 - Amikor egy router-t bekapcsolunk, küld egy „hello“-csomagot minden szomszédjának,
 - Miután a „hello“-csomagokat visszakapja
 - Meghatároz egy routing kapcsolatot mindazon router-ekkel a routing adatbankok szinkronizálásával, amely routerek ezt a szinkronizálást megengedik.
- Update:
 - Minden router rendszeres időközönként küld egy update-üzenetet, amely a saját u.n. „link-state“-jét (a routing-adatbankját minden más routerhez) írja le. Így minden router megkapja a lokális hálózati topológia leírását.
 - Minden router kiszámít egy legrövidebb utak fáját. Ez a fa minden kommunikációs célhoz megadja a következő routert, amin keresztül kell küldeni az üzenetet.