Hálózattervezés Alapjai
ősz 2006

1: Alapok, minimális feszítőfák, legrövidebb utak

Az előadáshoz

- Előadás: Csütörtök 15:05-16:35 óra
- Vizsga írásbeli
- Honlap: http://people.inf.elte.hu/lukovszki/Courses/0607NWT
- 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

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 galária probléma
- Ad hoc hálózatok – topológia kontroll, útvonalválasztás

Aszimptótika

Legyen $f,g : \mathbf{N_0} \rightarrow \mathbf{R}$ két függvény.

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

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:

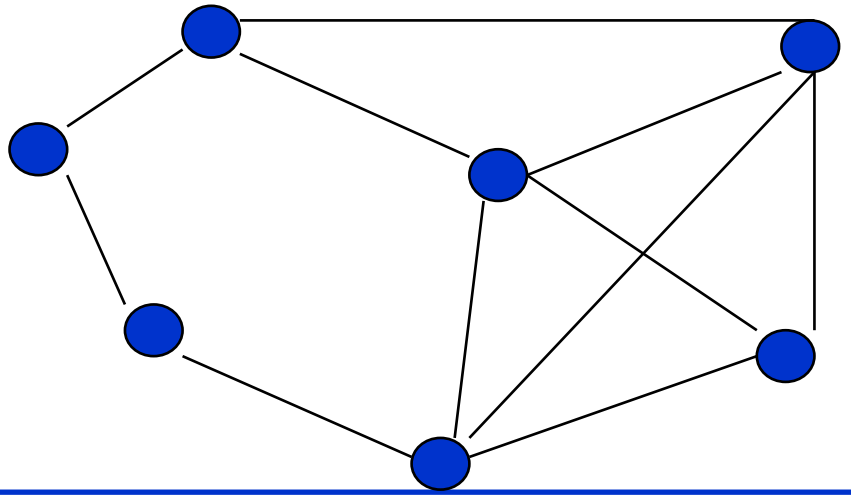
- $T(P) = O(f(n))$, ha $\exists$ *A* algoritmus, hogy *A* megoldja *P*-t és $T_A(n) = O(f(n))$
- $T(P) = \Omega(f(n))$, ha $\forall$ *A* algoritmus ami megoldja *P*-t : $T_A(n) = \Omega(f(n))$
- $T(P) = \Theta(f(n))$, ha $T(P) = O(f(n))$ és $T(P) = \Omega(f(n))$
- $T(P) = o(f(n))$, ha $\exists$ *A* algoritmus,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

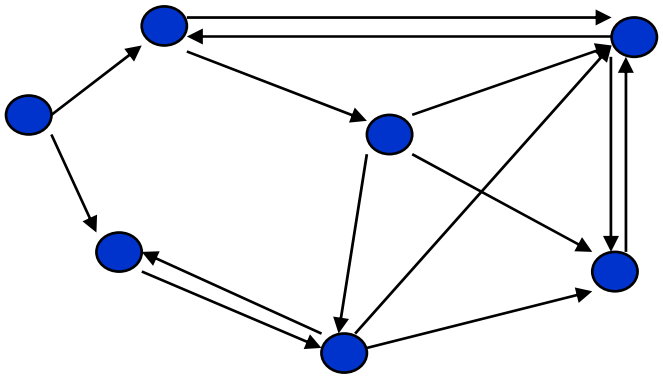
Irányítatlan gráfok

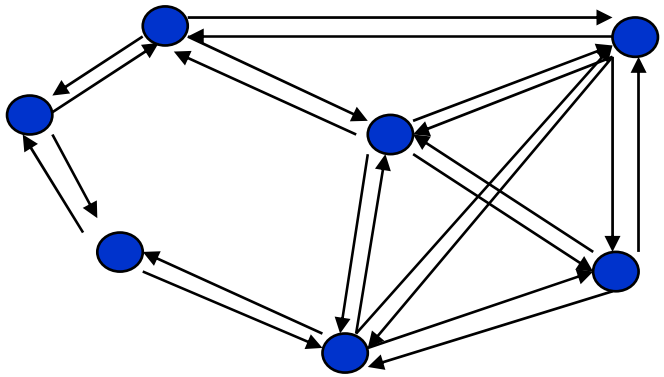
- Az élekhez nics irány hozzárendelve
- Az éleket a csomópontok kételemű részhalmazai reprezentálják: $e=\{u,v\}$
- Azt mondjuk, hogy az él $e=\{u,v\}$ a u és v csomóponthoz incidens.
- Ha a csomópont u és v egy éllel össze van kötve, akkor azt mondjuk, hogy u és v adjacens.



Irányított gráfok (Digráfok)

- Minden élnek van egy kezdöpontja 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:
  $(u,v) \in E \Leftrightarrow (v,u) \in E$.



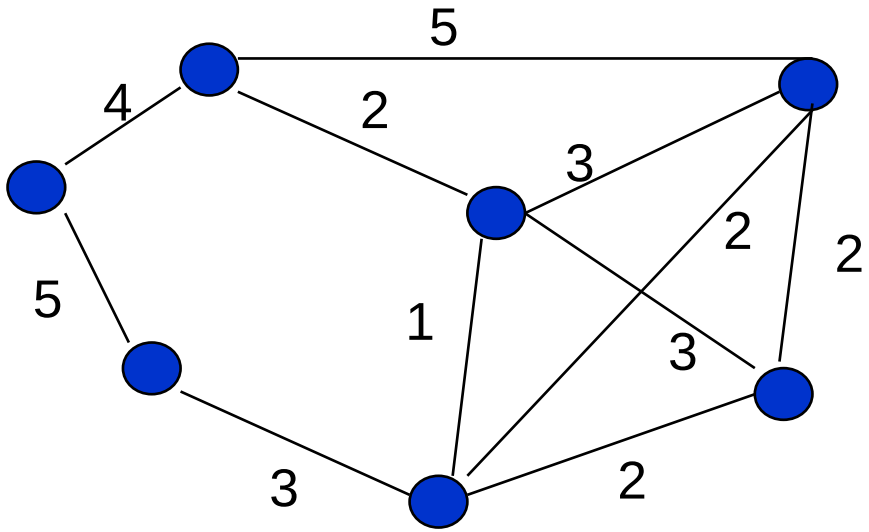


Gráfok reprezentációja

- A továbbiakben 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 es a kimenő élekről)
  - Tárigény: $O(n+m)$
- Alternativ: Adjazecia mátrix
  - nxn 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 *cap* : $E \rightarrow R^+$ függvény által, ami a link sávszélességét adja meg.

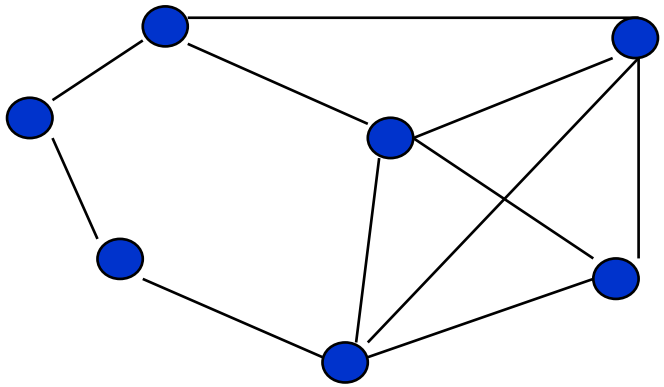


Út, kör

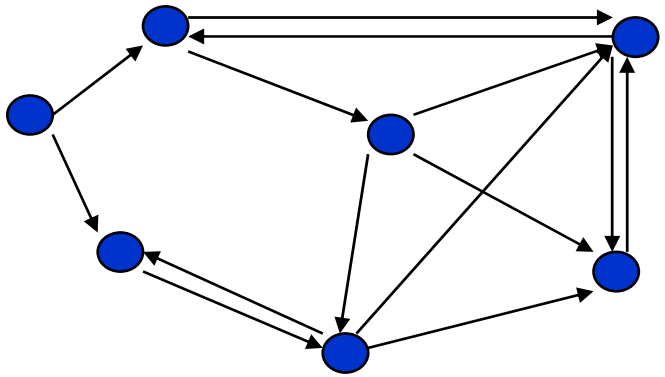
- Egy út u-tól v-hez egy $G=(V,E)$ gráfban ($u,v \in V$) egymást követő élek egy olyan sorozata $(x_0,x_1),(x_1,x_2),\ldots,(x_{k-1},x_k)$, ahol $u = x_0$, $v = x_k$.
- Egy egyszerű út u-tól v-hez egy olyan út u-tól v-hez, ahol minden csomópont csak egyszer fordul elő.
- Egy P út hossza egy súlyozatlan 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 $(x_0,x_1),(x_1,x_2),\ldots,(x_{k-1},x_k)$, 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.



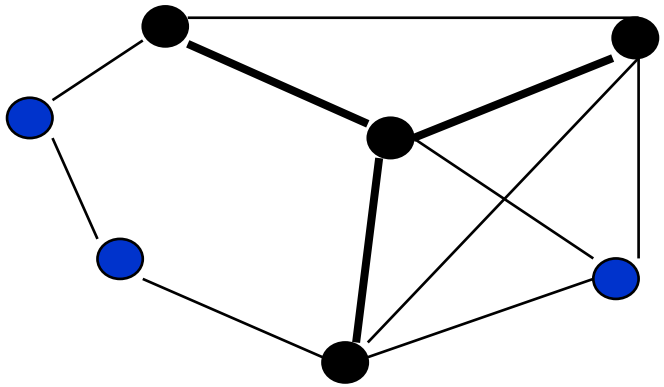
összefüggő



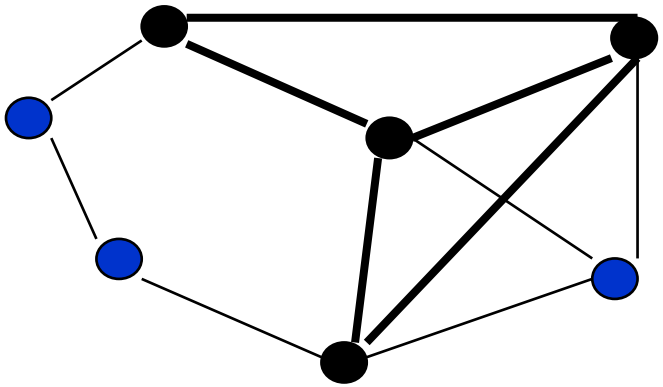
nem erősen összefüggő

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.



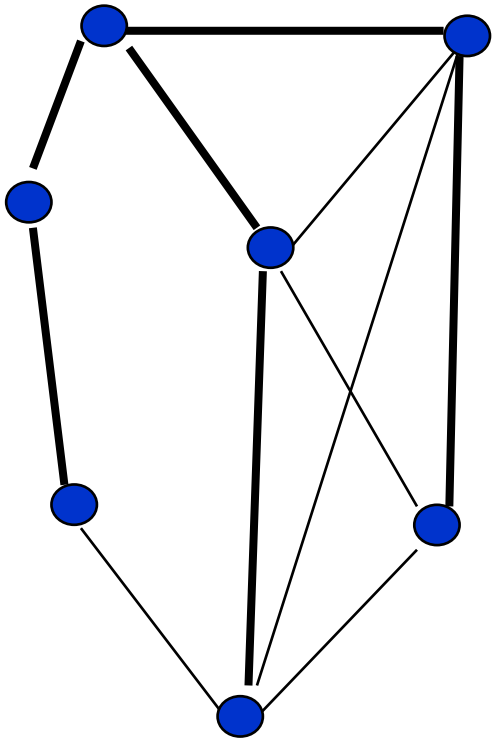
részgráf



indukált részgráf

Fák, Feszítőfák

- Egy irányítatlan gráf egy fa, ha összefügő és nem tartalmaz kört.
- Egy iányítatlan $G=(V,E)$ 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

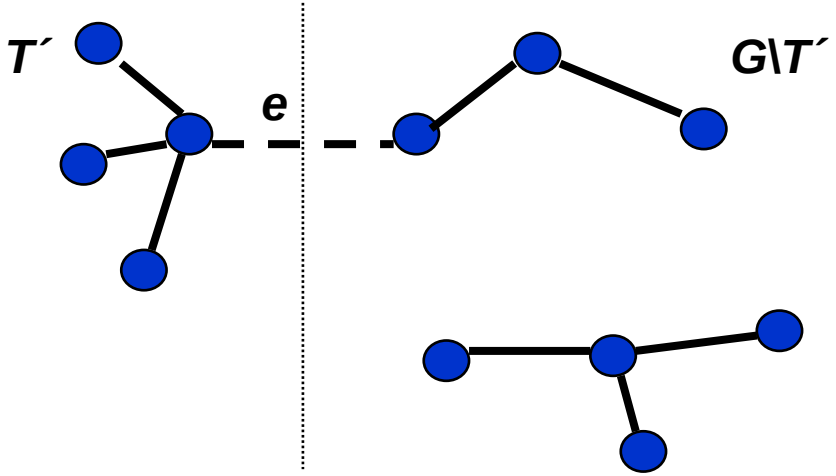
Minimális Feszítőfa

- Probléma: Építsünk fel egy olyan kommunikációs hálózatot, amely az adott kommunikációs csomópontok halmazát összefüggően összeköti. Ehhez a csomópont párok között vezetéket fektethetünk le, amely 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.

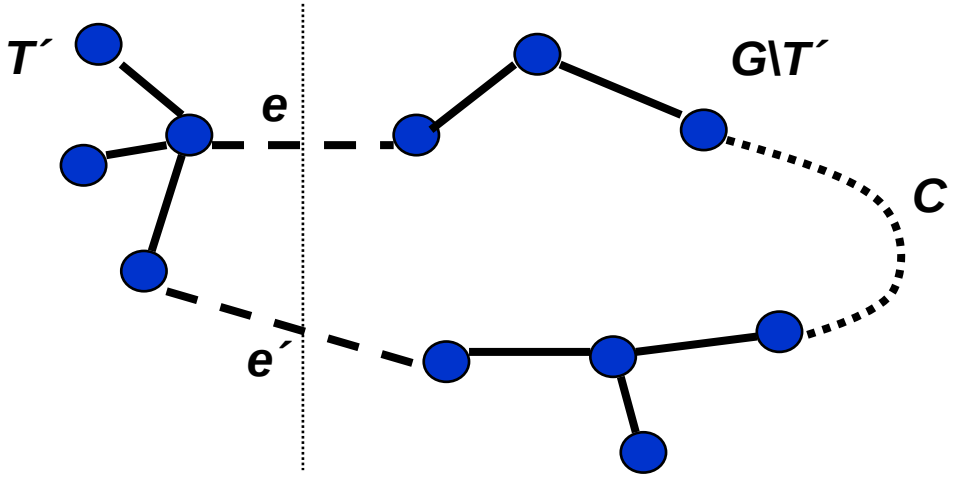
Egy minimális feszítőfa kiszámítása

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 $G \backslash T'$-ben van. Ekkor létezik egy minimális feszítőfa, ami $E' \cup \{e\}$ minden élét 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 mégegy 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. □



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 algorithmusa

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 szetint növekvő sorrendben do

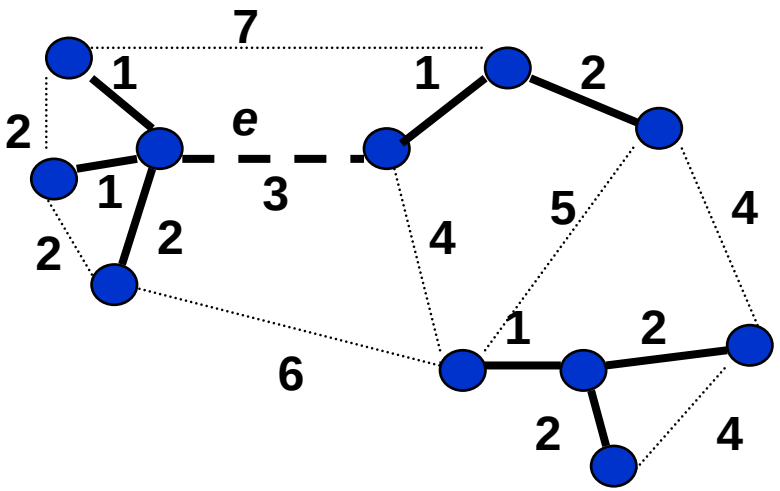
    if u és v különböző összefüggő koponensében van $G'=(V,E')$-nek then

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

    fi;

od;

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



Kruskal algoritmusának helyessége

Indukció:

- $E´=Ø$: 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 analízise

- 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 $i \geq 0$,

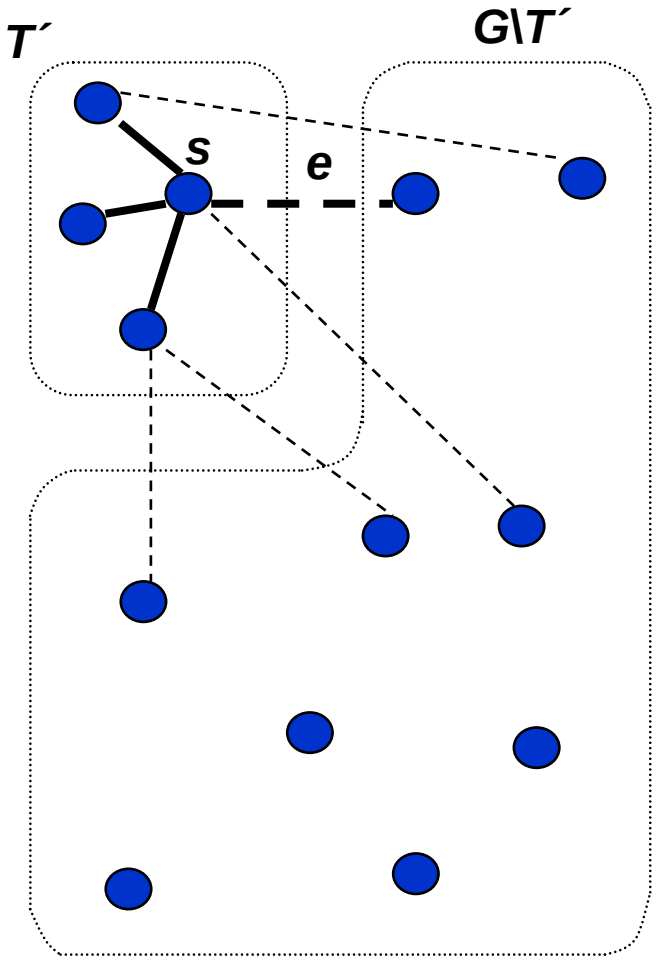
    $A(0,j)=2j$ ha $j \geq 0$,

    $A(i,j)= A(i\text{-}1, A(i,j\text{-}1))$ ha $i \geq 1$, $j \geq 2$.

- Összesen: $O(m \log n)$ idő, $O(n+m)$ tár

Prim algoritmusa

- Kiválasztunk egy tetszőleges *s* csomópontot, mint kezdőpont.
- Mindig azt a *T´* részfát 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 algoritmusa

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\T´*-beli csomóponttal kötnek össze.
- Ehhez egy u.n. Priority-Queue adatstruktúrát használunk. Egy Priority-Queue *Q* a következő operációkat bocsájtja 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 operació *O*(log *n*) idő alatt.
  - Fibonacci Heap: *Insert* és *DecreasePriority* O(1) időben és *DeleteMin* *O*(log *n*) időben amortizáltan.

Prim algoritmusa

Invariánsok:

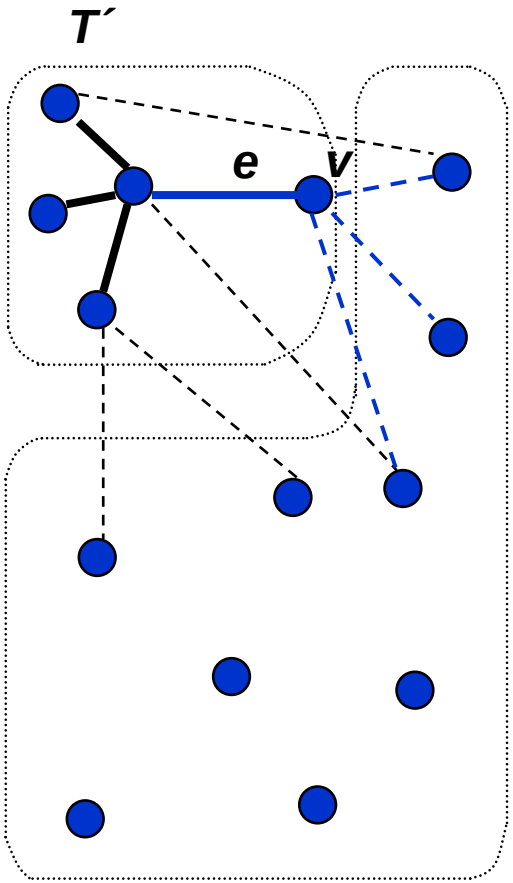
- Q taralmazza az összes $v \in G \setminus T'$ csomópontot, ami T'-ből egy éllel elérhető. Legyen e_v egy minimális súlyú él azon élek közül, melyek egy T'-beli csomópontot kötnek össze v-vel. Akkor v prioritása Q-ban: $c(e_v)$.
- Minden v csomóponthoz, ami Q-ban van, nyilvántartjuk egy $from[v]$ változóban 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 algoritmusa

- A következő élet, amit E'-hez hozzáadunk, a *Q.DeletMin*() operációval határozhatjuk meg:
  ezzel megkapjuk azt a $v \in G \backslash T'$ csomópontot, amely T'-ből egy minimális súlyú éllel érhető el.
- Ezután az e_v=*from*[*v*] élet hozzáadjuk E'-hez.
- Ezután a következő adtokat kell aktualizálni, hogy az invariánsok érvényesek maradjanak (*v* mostmát 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

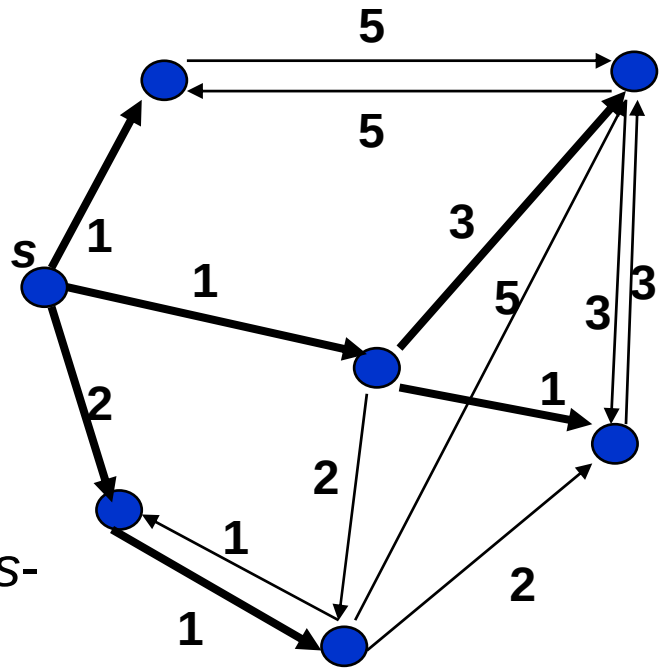
- Az aszimptótikusan leggyorsabb algoritmus a minimális feszítőfa kiszámítására $O(n\alpha(m,n) + m)$ időt és $O(n+m)$ tárat igényel [B. Chazelle 1998].
- A minimális feszítőfa kiszámításának időigényére ismert alsó korlát $\Omega(n+m)$.
- Nyitott Probléma: Meg lehet-e javítani az alsó vagy a felső korlátot?

Legrövidebb utak fája

Probléma: Legyen $G=(V,E)$ egy irányított gráf $c : E \rightarrow \mathbf{R}^+$ élsúlyokkal. Legyen $s \in V$ egy startcsomópont.
Számítsuk ki a legrövidebb utat s-től V minden más csomópontjához.

- Ez az Open Shortest Path First (OSPF) routingprotokoll alapja kommunikációs hálózatokban.

- Feltesszük, hogy minden csomópont elérhető s-ből egy úton. (nem elérhető csomópontokhoz nem létezhet legrövidebb út sem.)



Legrövidebb utak fája

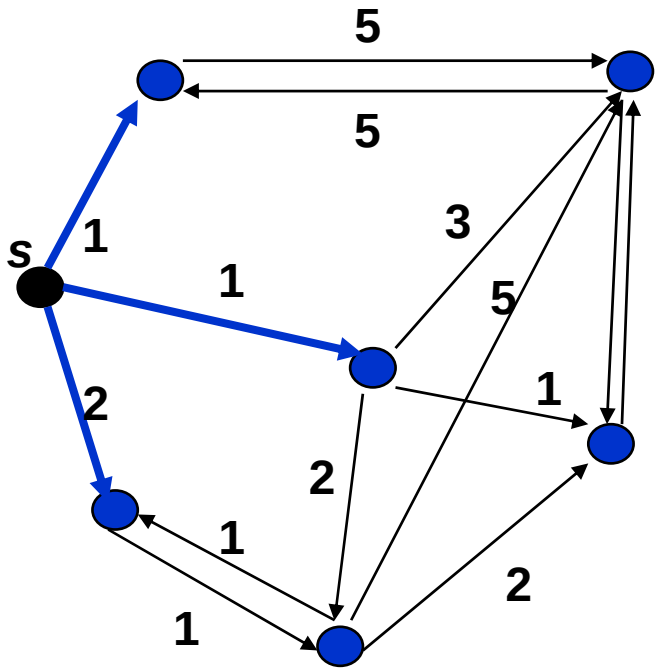
- A távolság *s*-től egy *w* csomóponthoz egy legrövidebb út hossza (az élsúlyok összege az úton) *s*-től *w*-hez.
- Minden $v \in V \setminus \{s\}$ csomóponthoz jelölje *from*[*v*] a *v*-t megelőző csomópontot egy legrövidebb úton *s*-től *v*-hez.
- Az élhalmaz $\{ (from[v],v) : v \in V \setminus \{s\} \}$ megadja *G*-nek egy legrövidebb utak fáját, azaz egy fát *s* gyökérrel, amely legrövidebb utakat tartalmaz *s*-től minden más csomóponthoz.

Dijkstra algoritmusa

- Ötlet: A legrövidebb utakat hosszuk szerint növekvő sorrendben számítjuk ki.
- Minden csomóponthoz kiszámítjuk a következő értékeket:
  - *dist*[*v*]: egy legrövidebb út hossza *s*-től *v*-hez,
  - *from*[*v*]: a *v*-t megelőző csomópont egy legrövidebb úton *s*-től *v*-hez.
- Egy *v* csomópontot „kész“-nek jelölünk: *ready*[*v*] = *true*,
  ha már meghatároztunk egy legrövidebb utat *s*-től *v*-hez.
- Invariánsok:
  - Minden még „nem kész“ csomópontot, amit egy „kész“ csomópontból egy éllel elérünk, egy *Q* priority-queue-ban tárolunk, úgy hogy minden $v \in Q$ csomópontra a következő érvényes:
    - *dist*[*v*] egy legrövidebb út hossza *s*-től *v*-hez mindazon utak között, melyek *v*-n kívül csak „kész“ csomópontokat tartalmaznak,
    - *from*[*v*] a *v*-t megelőző csomópont egy ilyen úton,
    - *v* prioritása *Q*-ban *dist*[*v*].

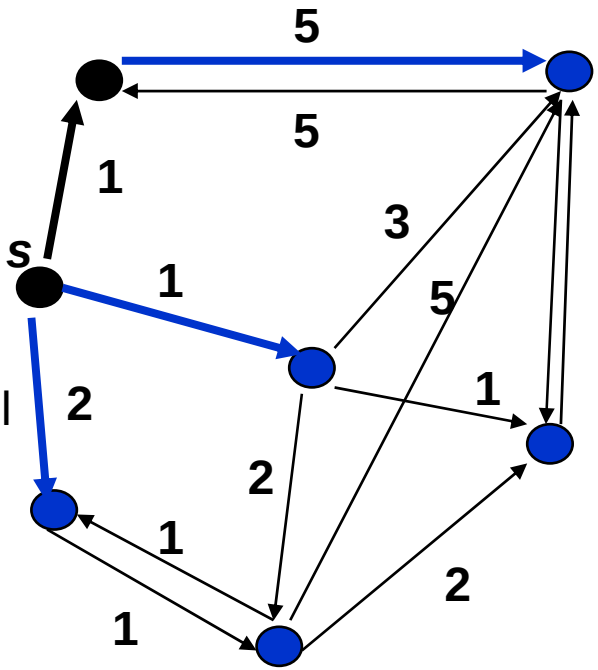
Dijkstra algoritmusa

- Inicializálás
  - *dist*[*s*]:=0, *ready*[*s*]:=true,
  - *s* minden *v* szomszédjára:
    - *dist*[*v*]:=*c*((*s*,*v*)), *from*[*v*]:=*s*, *ready*[*v*]:=*false*,
    - *Q.Insert*(*v*,*dist*[*v*]).
  - Minden más *v* csomópontra:
    - *dist*[*v*]:=∞, *ready*[*v*]:=*false*.



Dijkstra algoritmusa

- Az invariánsok megőrzése egy iteráció után
  - Minden lépésben egy új csomópont lesz „kész“, egy csomópont minimális prioritással.
  - *dist*[*v*] már tartalmazza a helyes értéket.
    (Mivel *v* minimális prioritású csomópont *Q*-ban, minden olyan út hossza *s*-től *v*-hez, 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.)
  - *v* minden *w* szomszédjánál, ami már *Q*-ban van, meg kell vizsgálni, hogy *s*-től *w*-hez direkt *v*-ből egy rövidebb út vezet-e, mint azok az utak, amik csak *v*-től különböző „kész” csomópontot tartalmaznak. Ha igen, akkor *dist*[*w*]-t és *from*[*w*]-t ennek megfelelően aktualizálni kell.
  - *v* minden *w* szomszédjánál, ami még nincs benne *Q*-ban:
    - *from*[*w*] := *v*, *dist*[*w*] := *dist*[*v*]+*c*((*v*,*w*))
    - *w*-t be kell szúrni *Q*-ba *dist*[*w*] prioritással.



Dijkstra algoritmusa

Input: Egy irányított $G=(V,E)$ gráf
$c : E \rightarrow \mathbf{R}^+$ élsúlyokkak és
s startcsomóponttal
Output: Legrövidebb utak s-től minden más csomóponthoz (amik *dist* és *from* által adottak)

```
1.  ready[s]:=true;
2.  ready[v]:=false   ∀ v ∈ V \ {s};
3.  dist[s]:=0;
4.  dist[v]:=∞   ∀ v ∈ V \ {s};
5.  priority_queue Q;
6.  for all e={s,v} ∈ E do
7.      from[v]:=s;
8.      dist[v]:=c(e);
9.      Q.Insert(v,c(e));
10.  od;
11. while Q ≠Ø do
12.     v := Q.DeleteMin();
13.     ready[v]:=true;
14.     E´:= E´U{(from[v],v)};
15.     ready[v]:=true;
16.     for all e={v,w} do
17.         if w∈Q and dist[v]+c(e)<dist[w] then
18.             from[w]:=v;
19.             dist[w]:=dist[v]+c(e);
20.             Q.DecreasePriority(w,dist[w]);
21.         else if w∉Q and not ready[w] then
22.             from[w]:=v;
23.             dist[w]:=dist[v]+c(e);
24.             Q.Insert(w,dist[w]);
25.         fi;
26.     od;
27.  od;
```

Dijkstra 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 17. és 21. sorban: m (élenként 1) - $O(m)$ idő
- Inicializálás: $O(n)$ idő
- Összesen: $O(n \log n + m)$ idő

Tárigény: $O(n+m)$

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 szikronizálásával, 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.