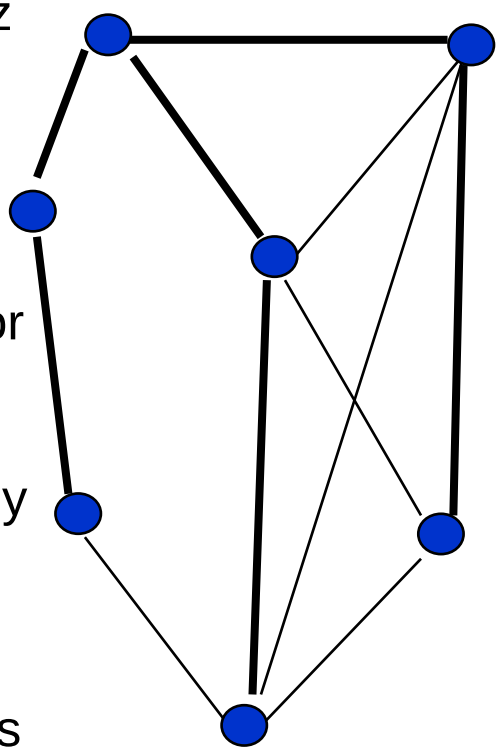
Hálózatok II
2007

2: Minimális feszítőfák, legrövidebb utak

Fák, Feszítőfák

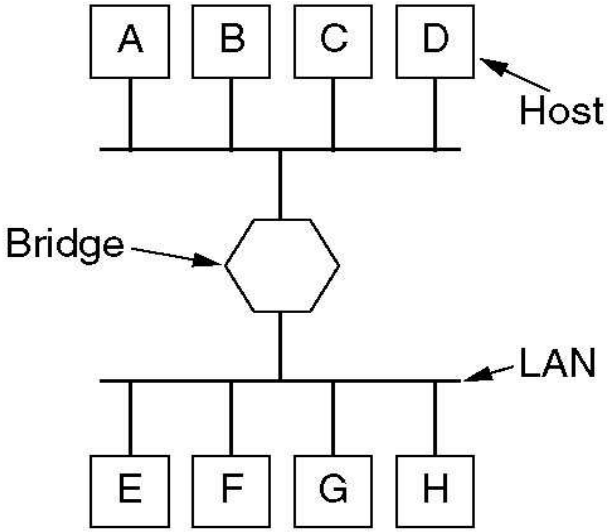
- Egy irányítatlan gráf egy fa, ha összefügő és nem tartalmaz kört.
- Egy irányítatlan $G=(V,E)$ gráf feszítőfája egy olyan fa, melynek csomóponthalmaza V és részgráfja G-nek.
- A csomópontokat, melyek foka 1, leveleknek nevezzük.
- Ha egy fa egy csomópontja ki van jelölve mint gyökér, akkor a fát gyökeres 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

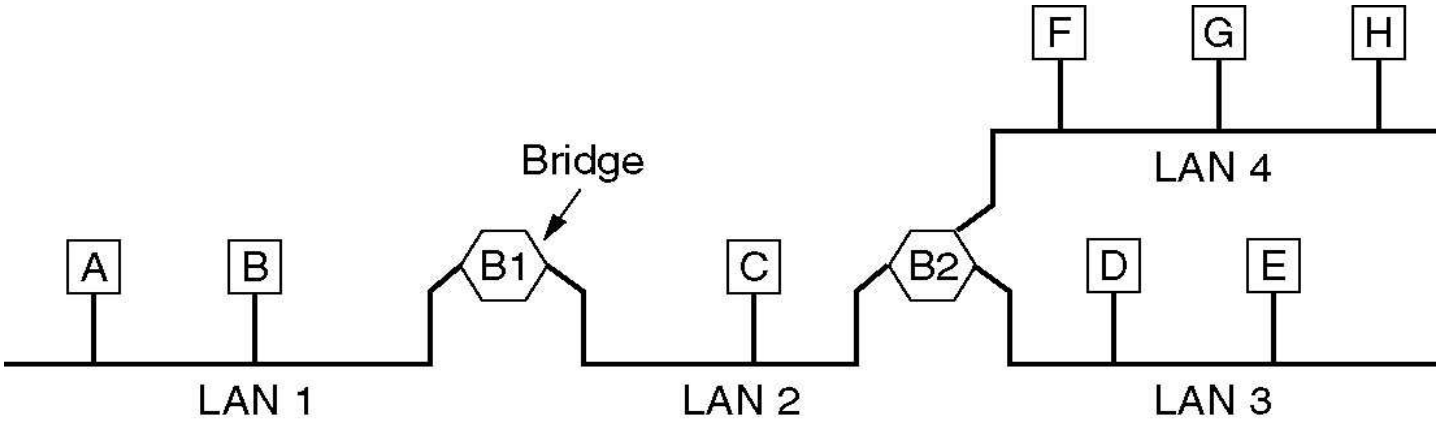
Példa: Feszítőfák és Bridge-ek

- Két lokális hálózatot kapcsol össze
  - Adatkapcsolati réteg komponense
  - Megvizsgálja az érkező csomagokat
  - A másik oldalra továbbítja, ha a cél-cím a másik hálózatban ismert, vagy ha azt még egyik oldal se látta
  - Egy táblázatot tart nyilván:
    - Megfigyeli, hogy honnan jön egy csomag, a küldőt azon a kábelen lehet elérni
    - Backward learning



Backward learning a bridge-ekben

- Példa: A küld csomagot E-nek
  - Tegyük fel, B1 és B2 tudja, hogy hol van E
  - B2 azt fogja látni, hogy A csomagja LAN2-ből jön
  - Mivel B2 nem tud LAN1-ről, B2 azt feltételezi, hogy A LAN2-ben van
  - Ami jó! B1 továbbítani fog minden A-nak küldött csomagot, amely LAN2-re érkezik, LAN1-nek



Backward learning a bridge-ekben – bootstrapping

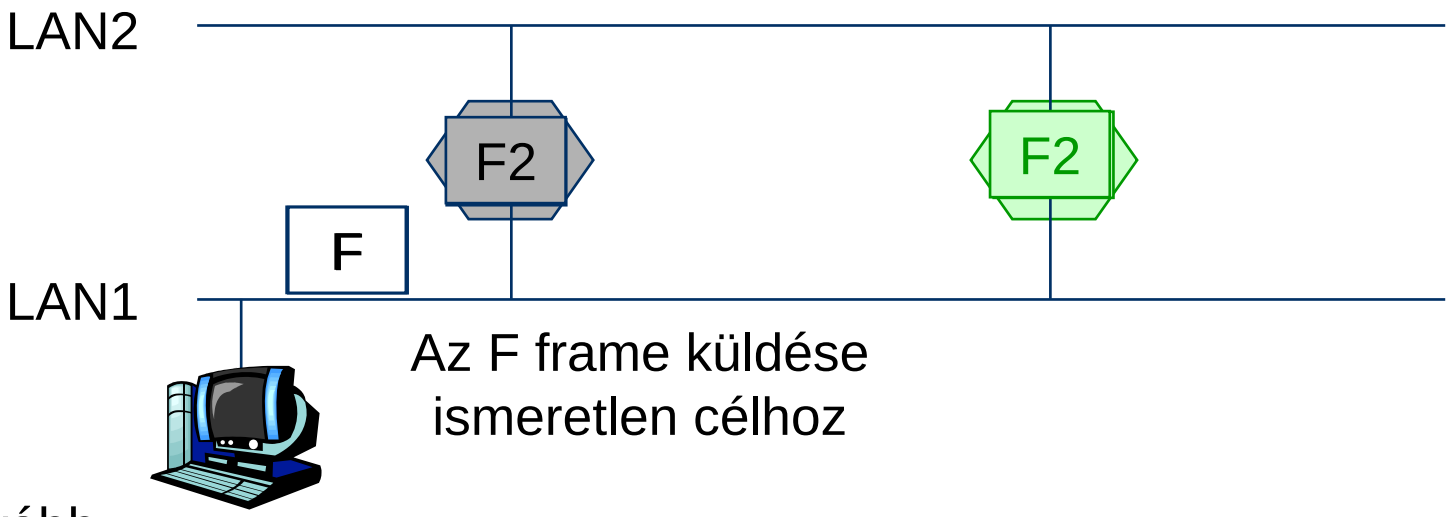
- Az előző példában: honnan tudja B2 kezdetben, hogy hol van E?

- Válasz: NEM tudja
  - Opció 1: kézi konfiguráció – nem éppen szép megoldás!
  - Opció 2: nem számít – egyszerűen továbbítja az ismeretlen című csomagot mindenfele
    - Azon hálózat kivételével, ahonnan érkezett

- Az algoritmus:
  - elárasztás (flood) ha a cím ismeretlen;
  - dobja el ha tudja, hogy nem szükséges;
  - továbbítsa specifikusan, ha a cél címe ismert

Elárasztás bridge által – problémák

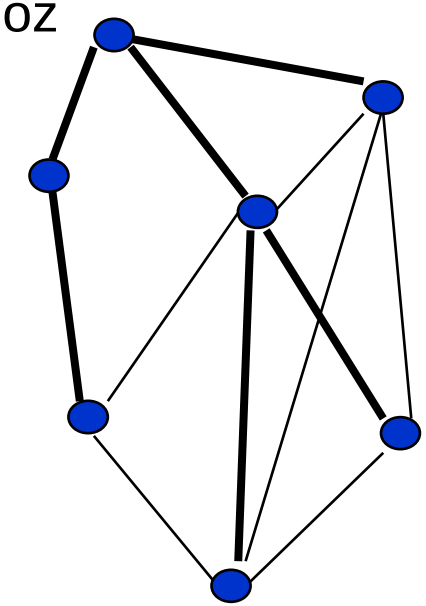
- “backward learning by flooding” egyszerű, de problémás
- Tekintsük a topológia példát:
  - Egy második bridge is összeköti a két LAN-t a nagyobb megbízhatóság miatt



- és így tovább …
- Hogy kerüljünk el ilyen ciklusokat?

Megoldás: Feszítőfák

- A frame-ek ciklusai csak akkor jöhetnek létre, ha a gráf, amit a bridge-ek definiálnak kört tartalmaz
  - Tekintsük a bridge-eket és a LAN-okat csomópontoknak
  - Egy bridge-csomópont és egy LAN-csomópont pontosan akkor van összekötve egy éllel, ha a bridge kapcsolódik a LAN-hoz
  - Redundáns élek köröket formálnak ebben a gráfban
- Ötlet: alakítsuk át a gráfot köröktől mentessé
- Legegyszerűbb megoldás: Számítsunk ki egy feszítőfát ebben a LAN-bridge gráfban
  - Egyszerű, ön-konfiguráló, nem kell kézi beavatkozás
  - De nem optimális: az installált bridge-ek kapacitását nem biztos hogy kihasználja
  - IEEE 802.1D: Spanning Tree Protocol (STP), IEEE 802.1w: Rapid Spanning Tree Protocol (RSTP)



Egy feszítőfa

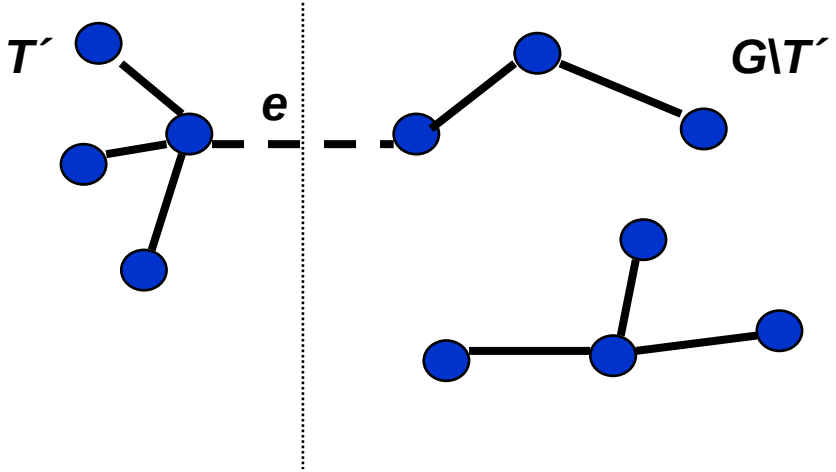
Minimális Feszítőfa

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

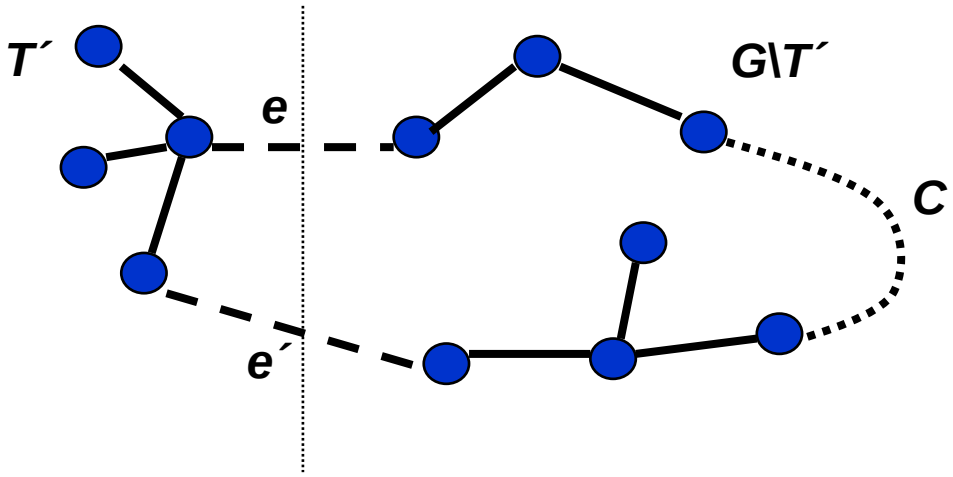
Egy minimális feszítőfa kiszámítása

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 $G \backslash T'$-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. □

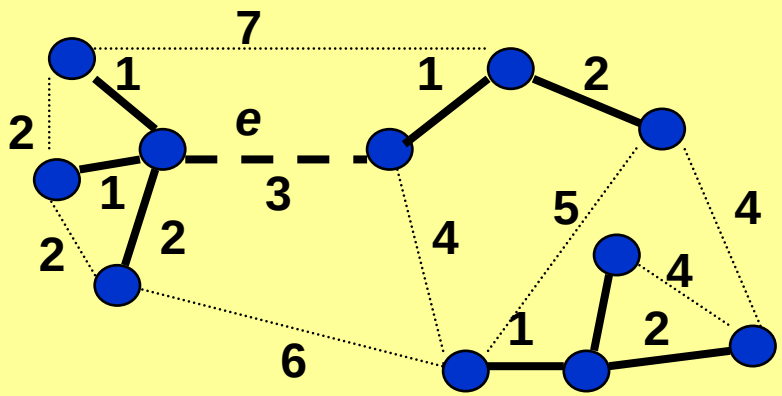


Egy minimális feszítőfa kiszámítása

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 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 $i \geq 0$,

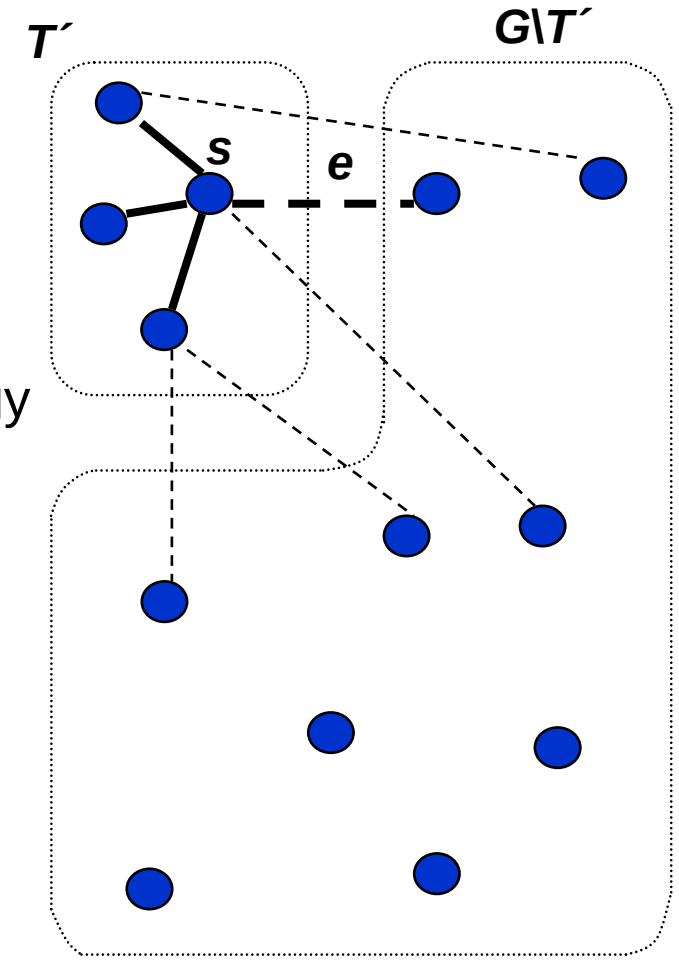
    $A(0,j)=2j$ ha $j \geq 0$,

    $A(i,j)= A(i\text{-}1, A(i,j\text{-}1))$ 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:

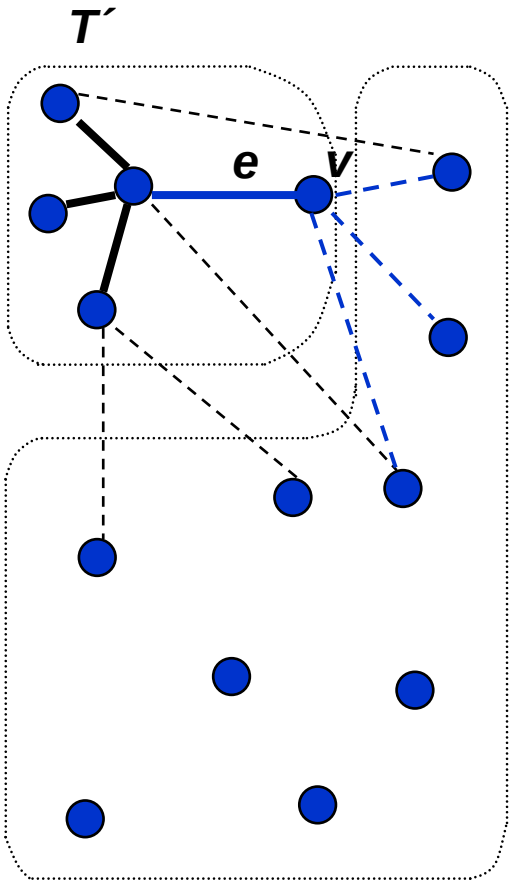
- Q taralmazza az összes $v \in G \setminus T'$ csomópontot, ami T'-ből egy éllel elérhető. Legyen e_v egy minimális súlyú él azon élek közül, melyek egy T'-beli csomópontot kötnek össze v-vel. Akkor v prioritása Q-ban: $c(e_v)$.
- Minden v csomóponthoz, ami Q-ban van, nyilvántartjuk egy *from*[v] változóban 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* 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,  ∀ v ∈ V \ {s};
4.  priority_queue Q;
5.  for all e={s,v} ∈ E do
6.      from[v] := e;
7.      Q.Insert(v,c(e));
8.  od;
9.  E´:=Ø;
10. while |E´| < |V| - 1 do
11.     v := Q.DeleteMin();
12.     E´:= E´ U {from[v]};
13.     ready[v]:=true;
14.     for all e={v,w} do
15.         if w∈ Q and c(e)<c(from[v]) then
16.             from[v]:=e;
17.             Q.DecreasePriority(w,c(e));
18.         else if w∉ 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

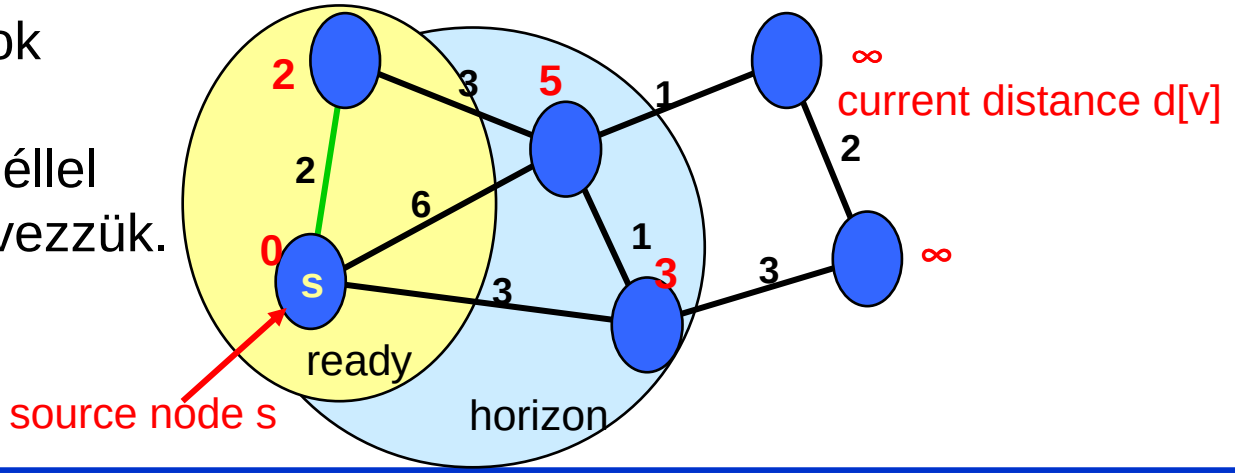
- Az aszimptótikusan leggyorsabb algoritmus a minimális feszítőfa kiszámítására $O(n\alpha(m,n) + m)$ időt és $O(n+m)$ tárat igényel [B. Chazelle 1998].
- A minimális feszítőfa kiszámításának időigényére ismert alsó korlát $\Omega(n+m)$.
- Nyitott Probléma: Meg lehet-e javítani az alsó vagy a felső korlátot?

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 egy út u-tól v-hez G-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 létezhet 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

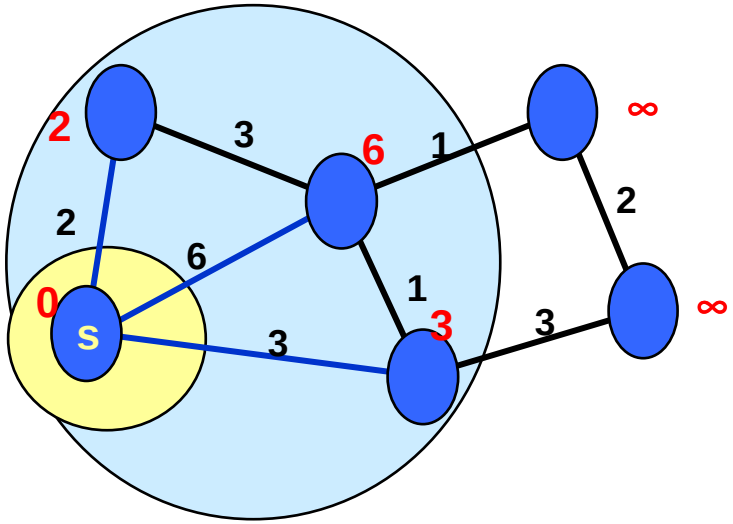
Dijkstra algoritmusa

- **Ötlet:** A legrövidebb utakat hosszuk szerint növekvő sorrendben számítjuk ki.
- Minden v ∈ 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∈V \ {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 algoritmusa

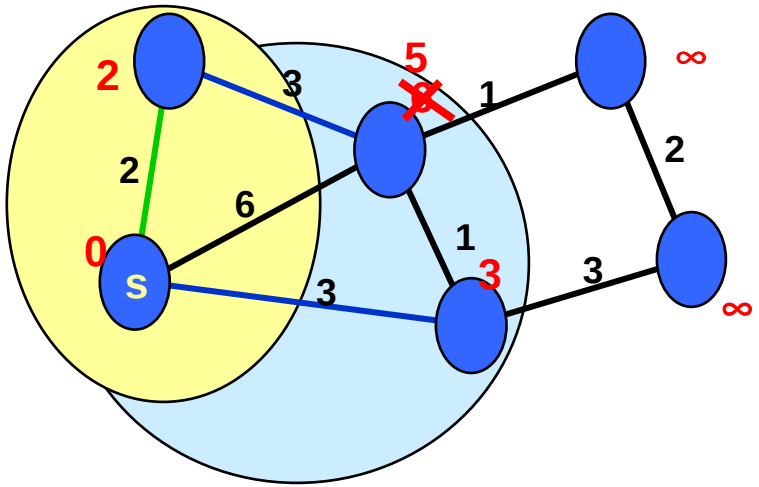
- **Invariánsok:**
  - Minden horizont beli csomópontot egy **Q priority-queue**-ban tárolunk, úgy hogy minden v $\in$ Q csomópontra a következő érvényes:
    - d[v] egy legrövidebb s-v út hossza mindazon utak között, melyek v-n kívül csak „kész“ csomópontokat tartalmaznak,
    - pred[v] a v-t megelőző 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 \ {s} csomópontra:
    - d[v]:=∞, ready[v]:=false.



Dijkstra algoritmusa

- 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) ∈ E }, v∈V, a v-hez adjacens csomópontok halmaza
  - minden u ∈ Adj[v], ha u ∈ 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 ∈ Adj[v], ha u ∉ Q és u „nem kész”:
    - pred[u] := v, d[u] := d[v] + c(v,u),
    - u-t be kell szúrni Q-ba d[u] prioritással.



Dijkstra algoritmusa

```
Dijkstra(G,s,c)
      Output: egy legrövidebb utak fája
      T=(V,E´) G-ben s gyökérrel
01    E´ := Ø;
02    ready[s] := true;
03    ready[v] := false;    ∀ v ∈ V \ {s};
04    d[s] := 0;
05    d[v] := ∞;      ∀ v ∈ V \ {s};

06    priority_queue Q;

07    forall v ∈ Adj[s] do
08       pred[v] := s;
09       d[v] := c(s,v);
10       Q.Insert(v,d[v]);
11    od
12    while Q ≠ Ø  do
13       v := Q.DeleteMin();
14       E´:= E´ U {(pred[v],v)};
15       ready[v] := true;
16       forall u ∈ Adj[v] do
17          if u ∈ 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 ∉ 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 algoritmusa

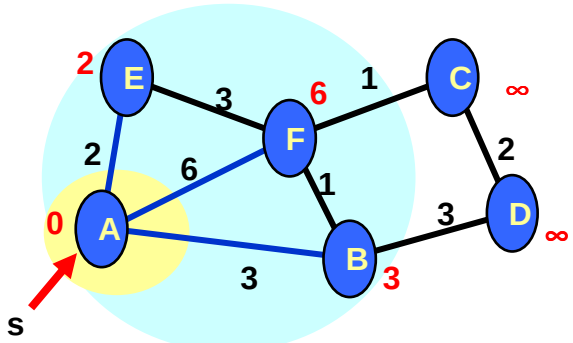
Futási idő (Fibonacci Heap-pel):

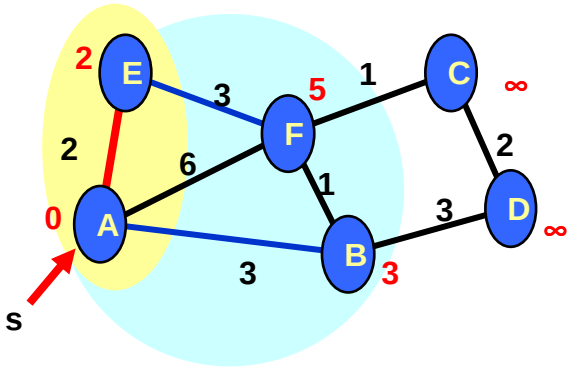
- # Q.Insert(): n (csomópontonként 1) -- O(n) idő
- # Q.DeleteMin(): n (csomópontonként 1) -- O(n log n) idő
- # Q.DecreasePriority(): ≤m (élenként ≤1) -- O(m) idő
- # A teszt a 17. és 21. sorban: m (élenként 1) -- O(m) idő
- Inicializálás: O(n) idő
- Összesen: O(n log n + m) idő

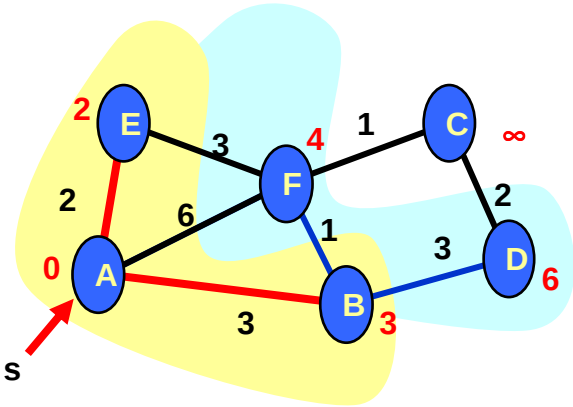
Tárigény: O(n+m)

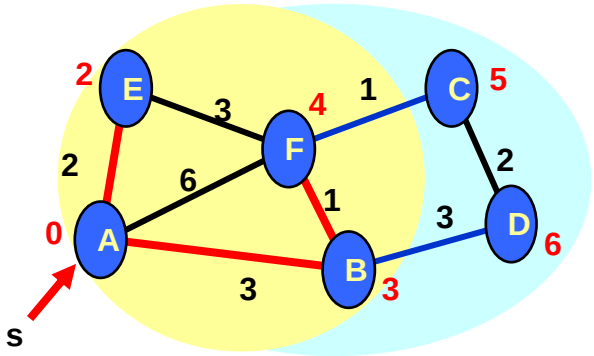
Dijkstra: Példa

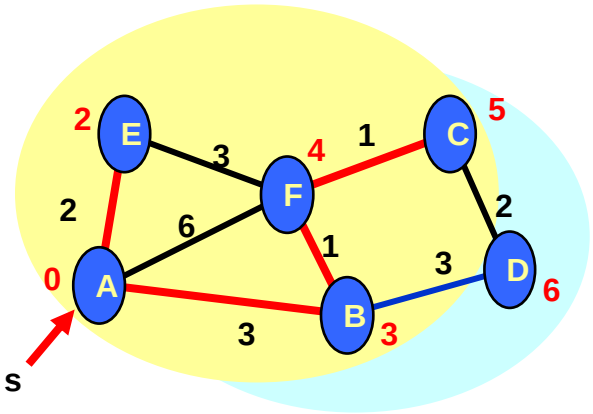
szimmetrikusan irányított élek

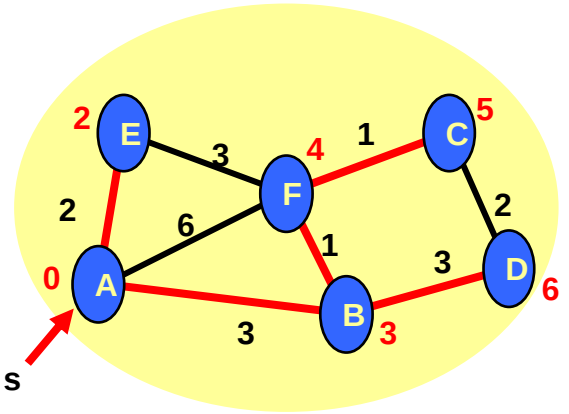












OSPF (Open Shortest Path First) Routing

- 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 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.