

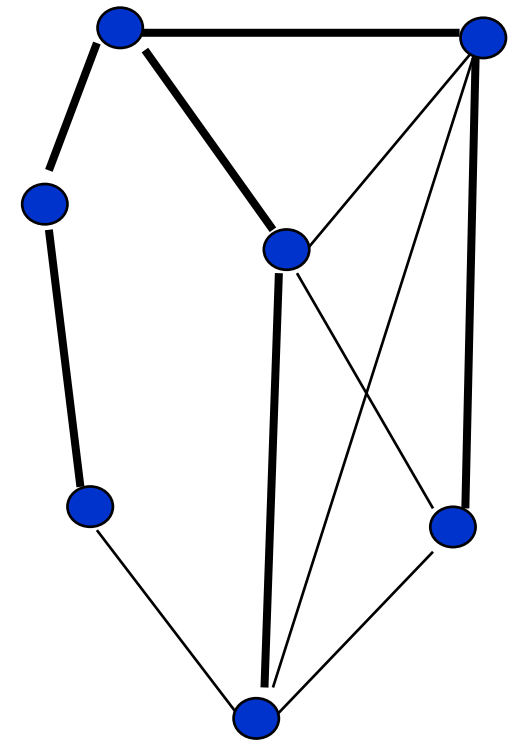
# Hálózatok II

## ősz 2006

### 2: Minimális feszítőfák, legrövidebb utak

## Fák, Feszítőfák

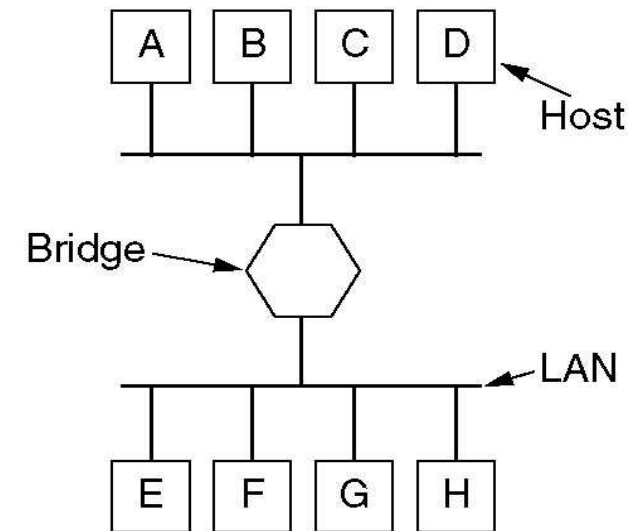
- Egy irányítatlan gráf egy **fa**, ha összefüggő és nem tartalmaz kört.
- Egy irányítatlan  $G=(V,E)$  gráf **feszítőfája** egy olyan fa, melynek csomóponthalmaza  $V$  és részgráfja  $G$ -nek.
- A csomópontokat, melyek foka 1, **leveleknek** nevezzük.
- Ha egy fa egy csomópontja ki van jelölve mint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

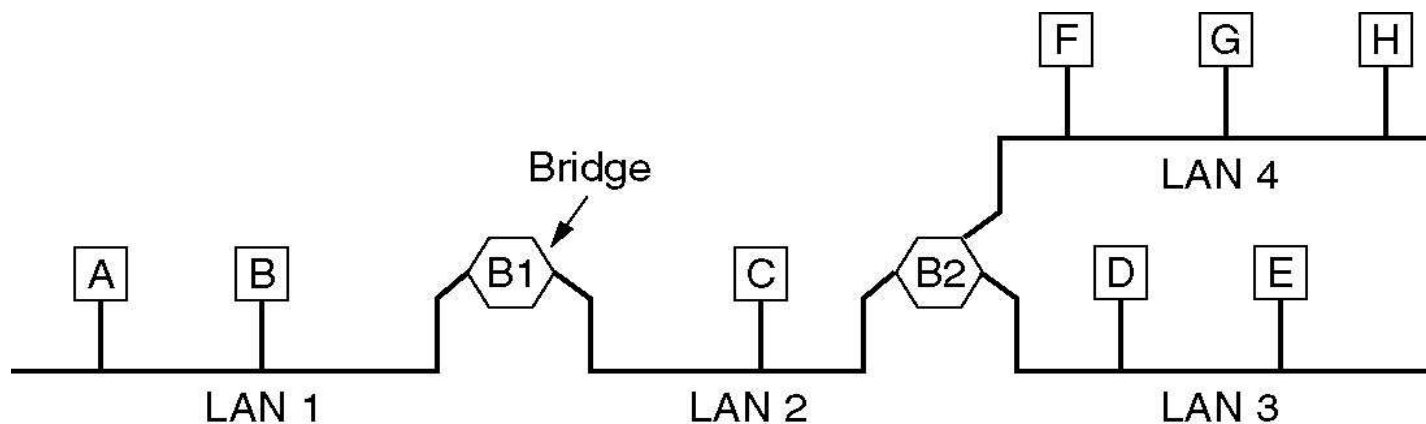
## Példa: Feszítőfák és Bridge-ek

- 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

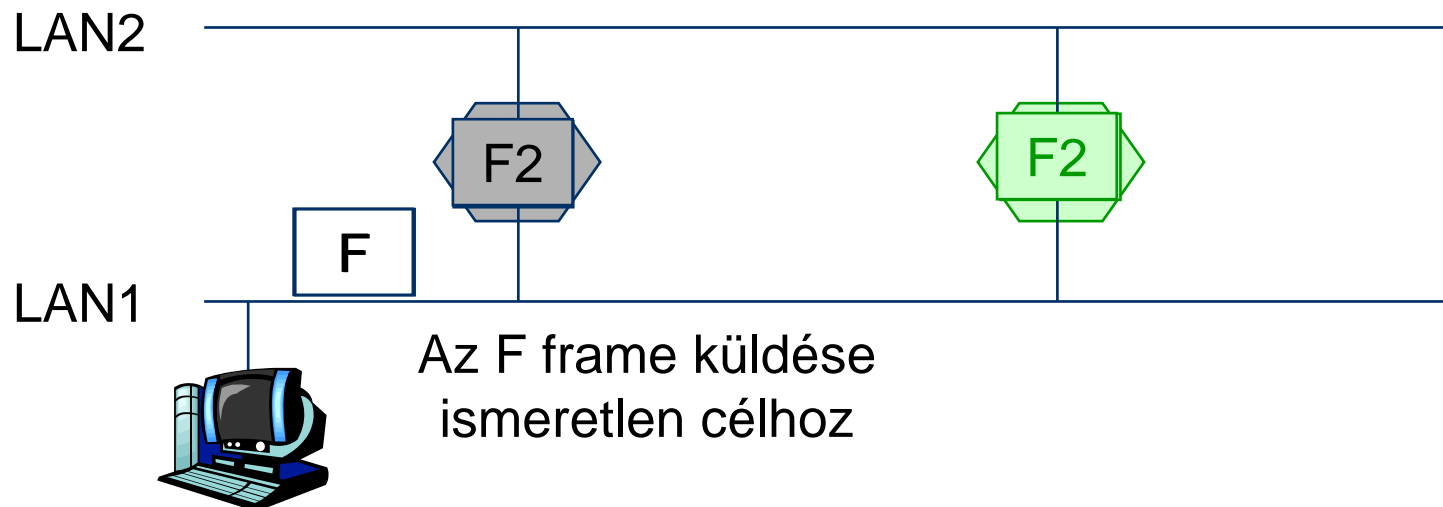


## Backward learning a bridge-ekben – bootstrapping

- 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

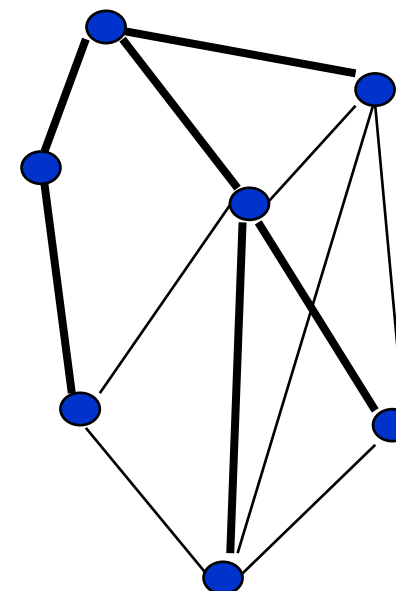
- “backward learning by flooding” egyszerű, de problémás
- Tekintsük a topológia példát:
  - Egy második bridge is összeköti a két LAN-t a nagyobb megbízhatóság miatt



- és így tovább ...
- Hogy kerülünk el ilyen ciklusokat?

## Megoldás: Feszítőfák

- A csomagok ciklusai csak akkor jöhetnek létre, ha a gráf, amit a bridge-ek definiálnak kört tartalmaz
  - Tekintsük a bridge-eket éleknek, a LAN-okat csomópontoknak
  - Redundáns bridge-ek 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 802.1w



Egy feszítőfa

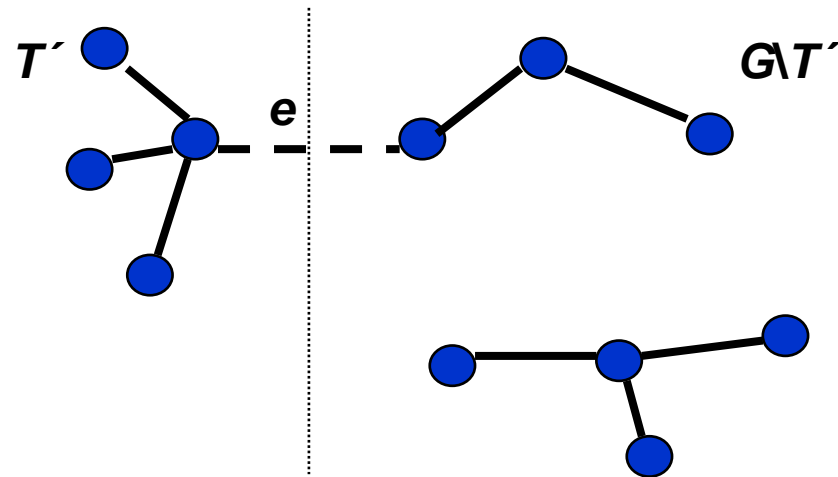
## Minimális Feszítőfa

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



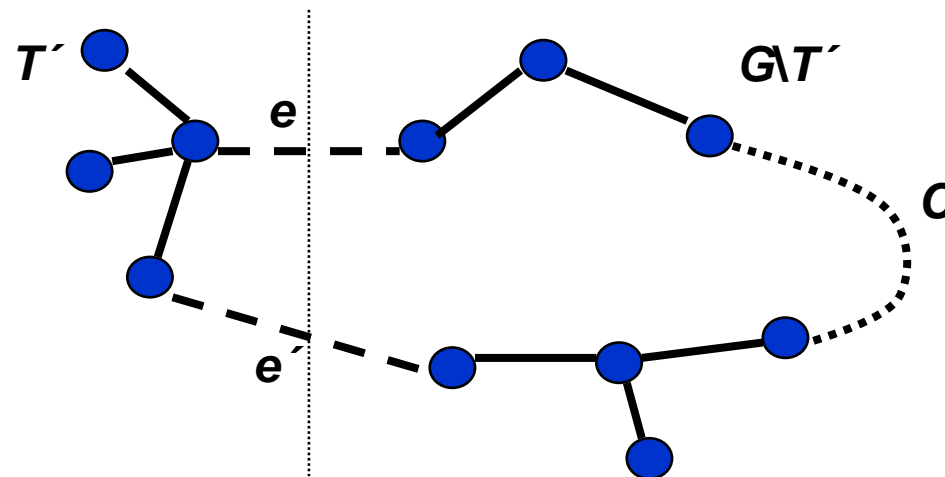
## Egy minimális feszítőfa kiszámítása

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  $G \setminus T'$ -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 hozzáadunk  $E'$ -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 szerint növekvő sorrendben do

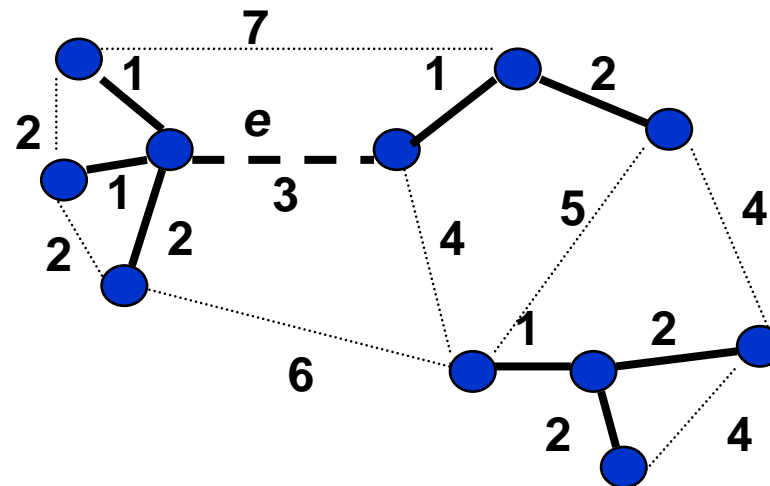
if  $u$  és  $v$  különböző összefüggő komponensében van  $G'=(V,E')$ -nek then

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

fi;

od;

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



## Kruskal algoritmusának helyessége

Indukció:

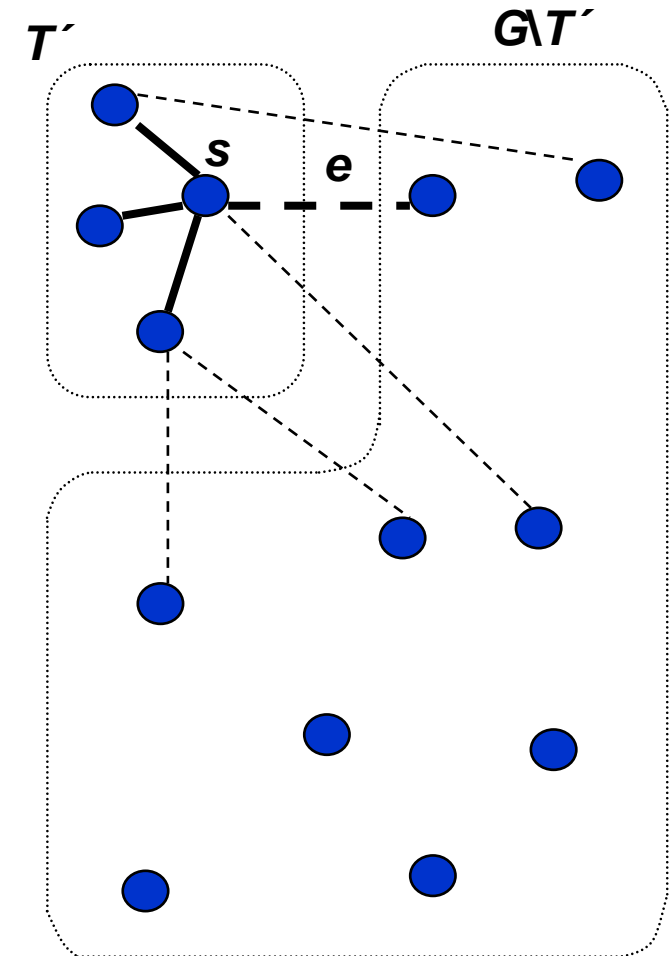
- $E' = \emptyset$  : az üres élhalmazt minden minimális feszítőfa tartalmazza.
- Legyen  $E'$  az aktuális élhalmaz. Indukciós feltétel:  $E'$ -t tartalmazza egy minimális feszítőfa  $T$ . Legyen  $e = \{u, v\}$  az az él, ami éppen feldolgozásra kerül. Ha  $u$  és  $v$  csomópont  $T \cap E'$  ugyanazon  $T$  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 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  $j \geq 0$ ,  
 $A(i,j)= A(i-1, A(i,j-1))$  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 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  $G \setminus T$ -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 operáció  $O(\log n)$  idő alatt.
  - **Fibonacci Heap**:  $Insert$  és  $DecreasePriority$   $O(1)$  időben és  $DeleteMin$   $O(\log n)$  időben amortizáltan.



## Prim algoritmus

### Invariánsok:

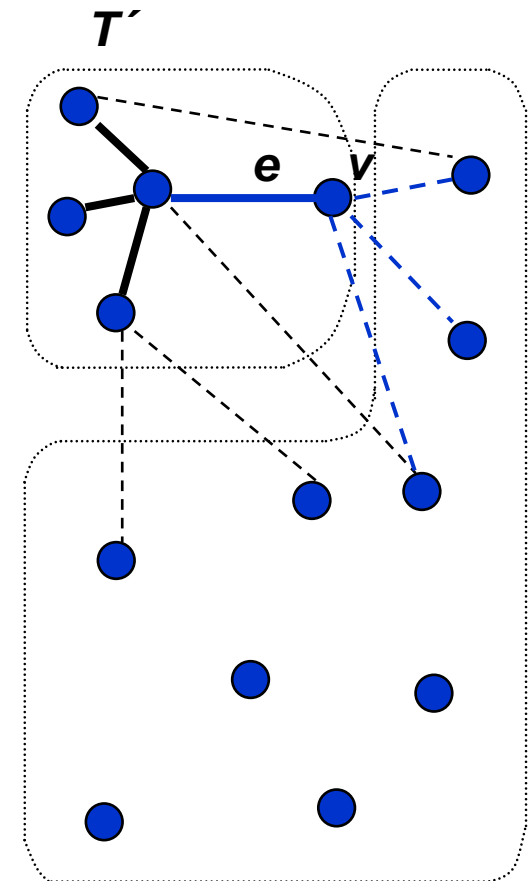
- $Q$  tartalmazza az összes  $v \in G \setminus T$  csomópontot, ami  $T$ -ből egy éllel elérhető. Legyen  $e_v$  egy minimális súlyú él azon élek közül, melyek egy  $T$ -beli csomópontot kötnek össze  $v$ -vel. Akkor  $v$  prioritása  $Q$ -ban:  $c(e_v)$ .
- Minden  $v$  csomópontoz, ami  $Q$ -ban 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 élet, amit  $E'$ -hez hozzáadunk, a  $Q.DeleteMin()$  operációval határozhatjuk meg: ezzel megkapjuk azt a  $v \in G \setminus T'$  csomópontot, amely  $T'$ -ből egy minimális súlyú éllel érhető el.
- Ezután az  $e_v = from[v]$  élet hozzáadjuk  $E'$ -hez.
- Ezután a következő adatokat 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 algoritmus

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 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  $\leq 1$ ) -  $O(m)$  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

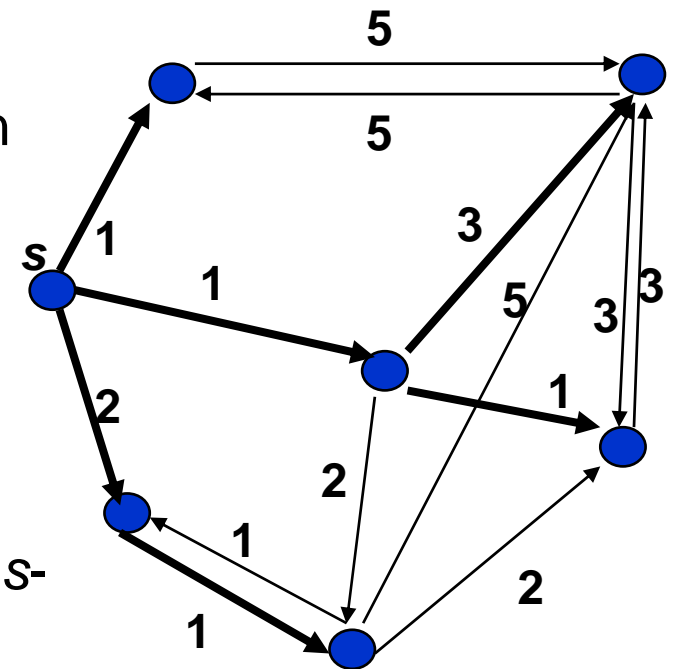
- Az aszimptótikusan leggyorsabb algoritmus a minimális feszítőfa kiszámítására  $O(n\alpha(m,n) + m)$  időt és  $O(n+m)$  tárat igényel [B. Chazelle 1998].
- A minimális feszítőfa kiszámításának időigényére ismert alsó korlát  $\Omega(n+m)$ .
- Nyitott Probléma: Meg lehet-e javítani az alsó vagy a felső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

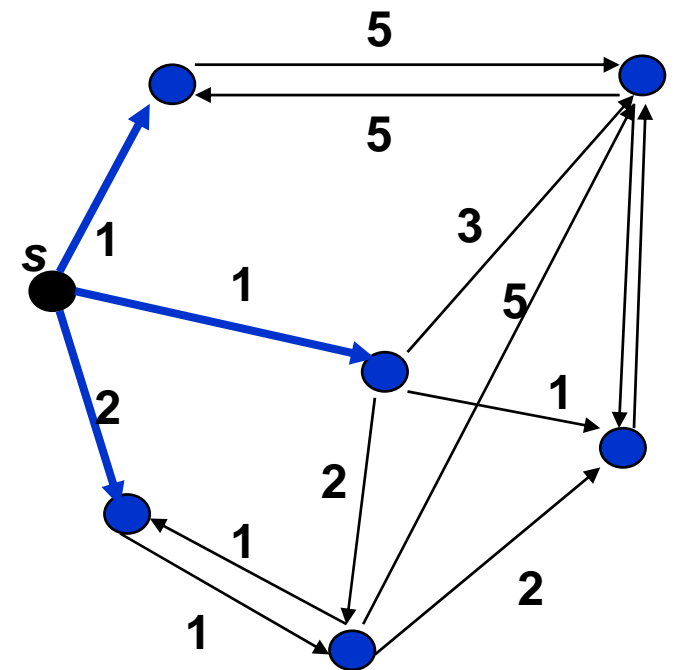
- Ötlet: A legrövidebb utakat hosszuk szerint növekvő sorrendben számítjuk ki.
- Minden csomóponthoz kiszámítjuk a következő értékeket:
  - $dist[v]$ : egy legrövidebb út hossza  $s$ -től  $v$ -hez,
  - $from[v]$ : a  $v$ -t megelőző csomópont egy legrövidebb úton  $s$ -től  $v$ -hez.
- Egy  $v$  csomópontot „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 algoritmus

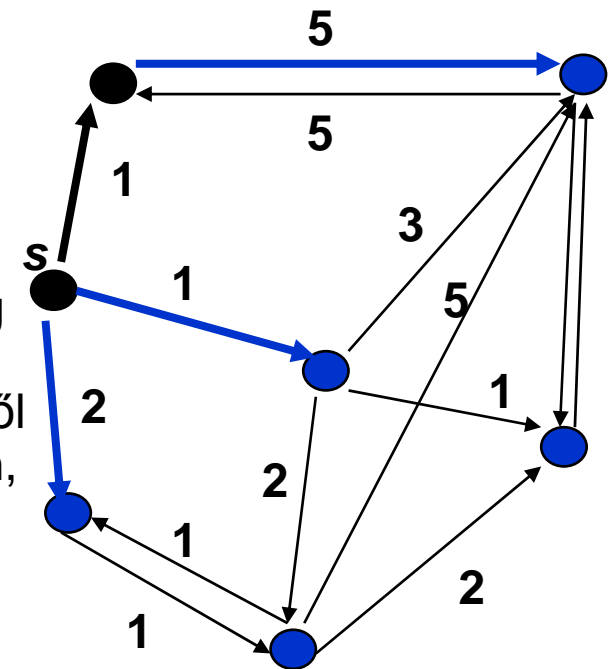
- Inicializálás

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



# Dijkstra algoritmus

- 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 algoritmus

Input: Egy irányított  $G=(V,E)$  gráf  
 $c : E \rightarrow \mathbf{R}^+$  élsúlyokk 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 ≠ ∅ do
12.     v := Q.DeleteMin();
13.     ready[v]:=true;
14.     E' := E' ∪ {(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 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  $\leq 1$ ) -  $O(m)$  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 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.