

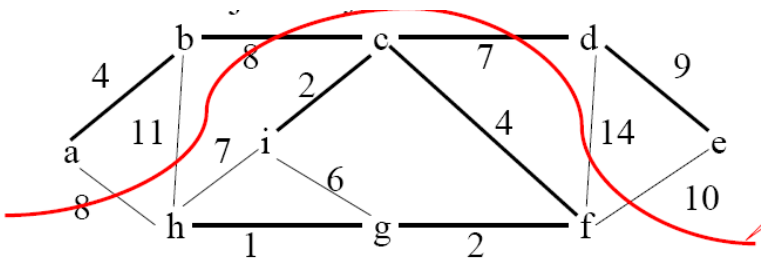
Optimális feszítőfák. Prim és Kruskal algoritmus

Gráfalgoritmusok – Optimális feszítőfa

- Súlyozott gráfok: az élekhez hozzárendelt $w:E \rightarrow R$ függvény.
- Def: egy G irányítatlan gráf feszítő fája: egy olyan fa, amely az eredeti csomóponthalmazából, és az élek egy részhalmazából áll.

A G gráf egy feszítő fája tartalmazza G összes csúcsát, valamint az élek egy olyan részhalmazát, amelyre teljesül az, hogy a keletkező gráf összefüggő, és nem tartalmaz kört.

- Minimális: az élek minimális (súlyú) részhalmazából áll
- Mind a mélységi, mind a széltében keresés létrehoz egy feszítő fát (feszítő erdőt).
- Prim és Kruskal algoritmus (mohó algoritmusok)
- Példa: a fa teljes súlya 37.



Minimális feszítőfa növelése

- Alapötlet: valamelyik csúcpontból kiindulva élenként növelve. Nyilvántartjuk a minimális feszítőfa eddigi felderített részét.
- Mohó stratégia: lépésenként egy „optimálisnak látszó” élt választunk ki.
- Azt az élt, amelyet a felderített fához hozzáveszünk, biztonságos élnak nevezzük, ha a fa még mindig optimális marad.
- Def: egy gráf egy vágata a csúcpontjainak két halmazba osztása. Léteznek a vágatot keresztező és kikerülő élek.
- Def: Egy vágatban könnyű élnak nevezzük a legkevesebb súlyú(ak)at.

Általános Minimális Feszítőfa Növelő algoritmus pszeudokódja

MinFFa

FFa = { }

while FFa nem feszítőfa

 keresünk egy biztonságos (u, v) élt

 FFa-ra nézve

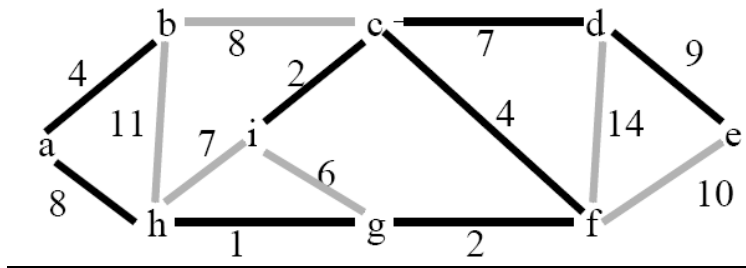
 FFa = FFa + { (u, v) }

Return FFa

- Tétel: Ha $G=(V,E)$ a $w:E \rightarrow R$ függvénnyel súlyozott gráf, és A olyan részhalmaza E -nek, ami egy minimális feszítőfa része is. Ha (u,v) egy könnyű él a FFa-t kikerülő vágatban, akkor az biztonságos is FFa-ra nézve
- Vagyis: a mohó algoritmus egyben globálisan optimális is

Kruskal algoritmusa

- Egy erdőt fogunk addig élekkel bővíteni, amíg fát nem kapunk.
- Alapból a gráf minden egyes pontja külön fát alkot, tehát nincs él az erdőben.
- A főciklus minden egyes lépésében kiválasztjuk a legkisebb súlyú olyan élet, amely két különböző fát köt össze.
- Ehhez könnyedén találunk olyan vágást, amely kielégíti a tétel feltételeit, tehát az biztonságos él lesz. Ezért ezzel az éllel bővítjük az erdőt.
- Ezt addig ismételjük, amíg kész nem lesz a fa.



Kruskal algoritmusa

MinFFaKruskal

```
Ffa={}  
for minden u∈V-re  
do HalmaztKészít(u)  
Élek rendezése súly szerint  
növekvő sorrendben  
for minden (u,v)∈E élre növekvő sorrendben  
do if  
HalmaztKeres(u) <> HalmaztKeres(v)  
then Ffa=Ffa+{(u,v)}  
Egyesít(u,v)  
return Ffa
```

Futási idő: elsősorban a diszjunkt halmaz megvalósítástól függ. Megfelelő megvalósítás esetén $O(|E| \cdot \log(|E|))$

Prim algoritmusa

Ebben az esetben egy fát bővítünk addig, amíg nem kapjuk meg az eredeti gráf egy feszítő fáját.

Egy tetszőleges pontból indulunk ki; kezdetben csak ebből az egy pontból fog állni a fa.

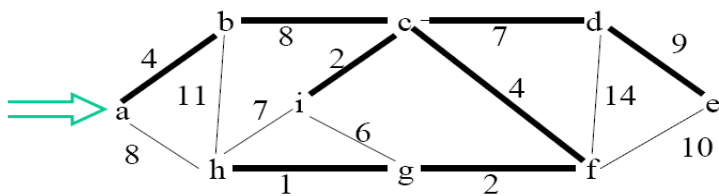
A vágást a fában levő pontok határozzák meg: azok kerülnek az egyik osztályba, a többi pont meg a másikba.

E két osztály közötti egy könnyű élet választunk.

- A bejáratlan csomópontokat prioritásos sorban tároljuk, amely a kulcs tulajdonság szerint van sorbaállítva (az odavezető él súlya) Amíg a prioritási sor nem üres, addig minden egyes lépésben kivesszük a minimális értékű pontot, és azon szomszédait fogjuk megvizsgálni, amely benne van még Q-ban, azaz a vágás másik oldalára esik.

- Ha egy csúcsot már elértünk, és az új irányból könnyebb élt találunk, akkor lecseréljük a kulcs értékét és a Szülő értéket

- Az eredmény a Szülő vektorban tárolt fa lesz



Hatékonysága: az elsőbbségi sor hatékonyságán múlik

- az első ciklus V -szer fut le
- a második ciklus szintén V -szer fut le, a KiveszMin művelet időszükséglete $O(\log V)$
- a for ciklus össz időszükséglete $O(E)$
- a kulcsot csökkent művelet időszükséglete $O(\log V)$
- össz. While*KiveszMin+For*KulcsotCsökkent, azaz $O(V \log V + E \log V) = O((V+E) \log V)$

Fibonacci-kupacok esetén javítható $O(E + V \log V)$ -re...