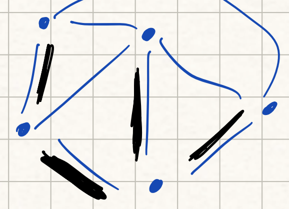
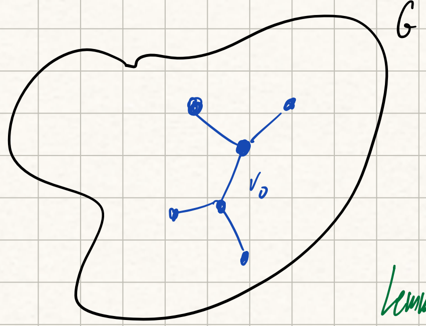
Problém minimální kostry:

- je dán souvislý neorientovaný graf
- jsou dány váhy hran $w: E \to \mathbb{R}$
- chceme najít kostru T s nejmenší vahou → $w(T) := \sum_{e \in T} w(e)$



Jarníkův algoritmus:



Pěstujeme strom T:

1) Vybereme nejlehčí z hran mezi T a $V \setminus T$ a přidáme ji mezi T.

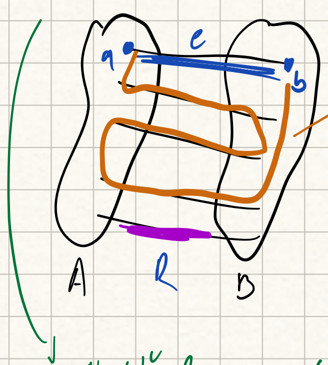
- Tohle je hladový algoritmus
- Není efektivní

→ po max. n krocích

Lemma: Jarníkův alg. se zastaví, T je na konci kostra.

Vždy přidáváme list, takže vytváříme strom / kostru. Pokud by na konci nebyly všechny hrany součástí, byl graf nesouvislý.

Lemma: Řezové lemma:



→ elementární řez

→ cesta z a do b.

Df: Množina $R \subseteq E$ je elementární řez $\equiv$ $\exists A \subseteq V$, $B = V \setminus A$, $A, B \neq \emptyset$ t.ž. $R = E(A,B)$

→ $\{\{a,b\} \in E \mid a \in A \ \& \ b \in B\}$

Nechť G je graf s unikátními hranami, R je elementární řez v G, e je nejlehčí hrana v R a T je nějaká minimální kostra. Pak $e \in T$.

Dk: Nechť T je kostra a $e \notin T$.

Jelikož T je kostra, existuje cesta mezi a,b. Jelikož $e \notin T$, cesta nejde přes e.

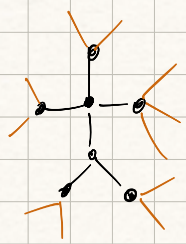
- $R = E(A,B)$, $e = \{a,b\}$: $a \in A$, $b \in B$, $\exists C$ cesta v T mezi a,b.

$\exists$ hrana f, kdy $f \in C \cap R$. $T \setminus f$ má dvě komponenty (jakmile z kostry odebereme hranu) a,b.

$\check{T} \equiv T - f + e$ je také kostra. $w(\check{T}) = w(T) - w(f) + w(e)$ → $-w(f) + w(e) < 0$

my víme, že e je nejlehčí, tedy že $w(f)$ je těžší. Tím pádem jsme vyměnili těžší za lehčí.

Jarníkův algoritmus najde minimální kostru:



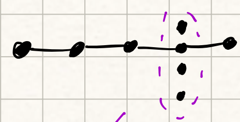
→ hrany mezi T a zbytkem grafu tvoří elem. řez.

- J.a. vybral nejlehčí hranu tohoto řezu.

Nalezená kostra ≤ každá min. kostra => Všechny minimální kostry jsou si rovny => ∃! minimální kostra.

Minimální kostra je jednoznačně určena pořadím podle vah

Co když váhy nebudou unikátní?



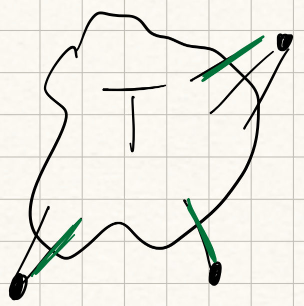
↳ lze rozhodnout a „doporovnat" stejné váhy → lin. uspořádání

Časová složitost J.a.

- $O(N \times M)$

#kroků — čas na jeden krok

Verze podle Dijkstry:



→ každý vrchol si pamatuje nejlehčí hranu, která jde do T

↳ pro v sousedy T:

$h(v) := \min\{w(e) \mid e \in V\}$

hrana vede do vrcholu

1 krok:

Najdu v s min. $h(v)$

Přidám $h(v)$ do T

Přepočítám $h(v)$

přepočítávám jen sousedy přidaného vrcholu v.

- stavy vrcholů:
  - zavřený := je součástí T
  - otevřený := je soused T
  - neviděný := všechny ostatní

tzn. že jde o obdobu relaxačního algoritmu

Implementace:	pole:	halda:
Insert:	1	log n
Extract min:	n	log n
Decrease:	1	log n
Celý alg:	$O(n^2+m)$	$O((n+m)\log n)$

vyhledat min

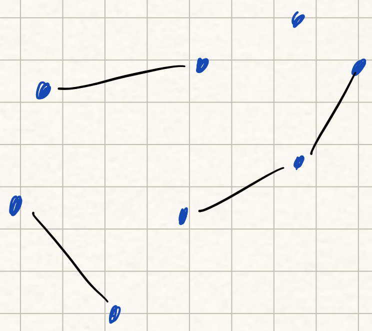
Algoritmus Jarník2

Vstup: Souvislý graf s váhovou funkcí w

1. Pro všechny vrcholy v:
2. $stav(v) \leftarrow mimo$
3. $h(v) \leftarrow +\infty$
4. $p(v) \leftarrow nedefinováno$ ◁ *druhý konec nejlehčí hrany*
5. $v_0 \leftarrow$ libovolný vrchol grafu
6. $T \leftarrow$ strom obsahující vrchol v_0 a žádné hrany
7. $stav(v_0) \leftarrow soused$
8. $h(v_0) \leftarrow 0$
9. Dokud existují nějaké sousední vrcholy:
10.   Označme u sousední vrchol s nejmenším $h(u)$.
11. $stav(u) \leftarrow uvnitř$
12.   Přidáme do T hranu $\{u, p(u)\}$, pokud je $p(u)$ definováno.
13.   Pro všechny hrany uv:
14.   Je-li $stav(v) \in \{soused, mimo\}$ a $h(v) > w(uv)$:
15. $stav(v) \leftarrow soused$
16. $h(v) \leftarrow w(uv)$
17. $p(v) \leftarrow u$

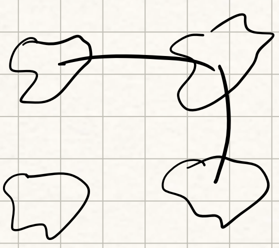
Výstup: Minimální kostra T

Borůvkův algoritmus:



1 Fáze

- Tvoříme postupně malé stromečky, každý si vybere svou minimální hranu a tyto hrany přidáme



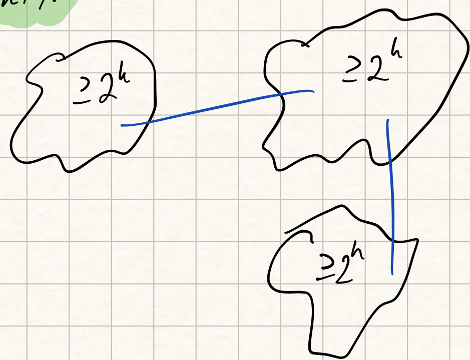
→ Zastavíme jakmile existuje jen jeden stromeček.

Lemma: Počet fází $\leq \log n$

Na konci k-té fázi mají všechny stromečky alespoň 2^k vrcholů.

↳ Indukce: $k=0$ … $1=2^0$ ✓

$k \to k+1$:



- Každý stromeček sroste s alespoň jedním dalším stromečkem

$\#\text{vrcholů} \geq 2^k + 2^k = 2^{k+1}$

$\boxtimes$

Složitost:

$O(m \cdot \log n)$

1 fáze — # fází

Lemma: Výstup je min. kostra

Každá přidaná hrana je nejlehčí v elem. řezu mezi stromečkem a zbytkem grafu.

↳ Proto leží v minimální kostře.

- cyklus tu nevznikne!

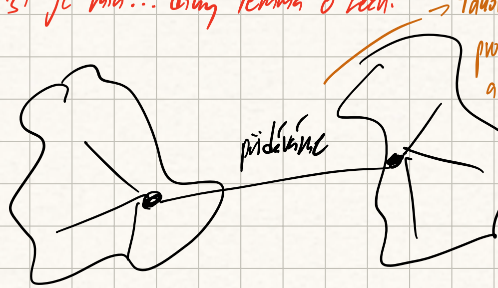
Kruskalův alg.

- setřídíme hrany od nejlehčí k nejtěžší
- postupně přidáváme do podgrafu
  - vznikne cyklus?
    - ano : hranu zahodíme
    - ne : hranu přidám

Lemma: Najde minimální kostru

- kostru najde triviálně. Najde ale minimální?

1) Podgraf je vždy les
2) na konci strom (kostra)
3) je min… díky lemma o řezu. → taková hrana je nejlehčí, protože jdu od nejlehčích a kdybych mezi stromy už měl hranu, tak tou novou bych vytvořil cyklus, takže bych ji zahodil.



Složitost

- musí se testovat acykličnost (což je složité)

$= m \times \mathrm{Find}()$ ⟨ ano: 1 ; ne: $n \times \mathrm{Union}()$

Union-Find (DFU)

- udržujeme komp. souvislosti
- operace Find(u,v) ... jsou u,v v téže komp.?
- operace Union(u,v) ... přidá hranu (u,v)

Implementace:

Blbá:

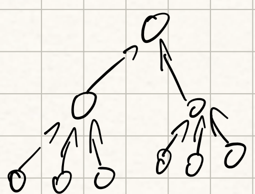
pole: vrchol -> číslo komponenty

Find: O(1)
Union: O(n)

$$\left.\begin{matrix}\text{Find}: O(1)\\ \text{Union}: O(n)\end{matrix}\right\} O(\underbrace{m}_{\text{Find}} + \underbrace{n^2}_{\text{Union (musím vše přepsat)}}) \Rightarrow O(n^2)$$

Lepší:

Komponenty reprezentujeme keříky.

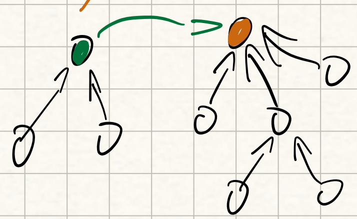


Strom orientovaný ke kořeni: 1 pole
Jeho vrcholy tvoří „keříky"
- vrchol si pamatuje jen svého otce

Find (v,u):
- do u,v do kořene keříků
  kořeny porovnáme
- složitost O(hloubky keříků)

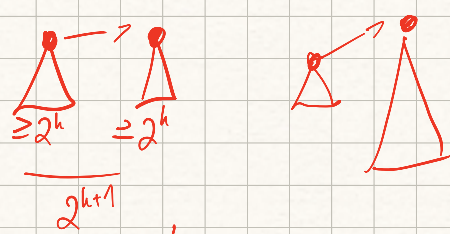
Union (u,v)
- najdeme kořeny u',v'
  - pokud u' = v', hotovo
  - jinak přidáme hranu mezi u',v'
- složitost O(hloubky keříků)

nastavíme otce, tím jsme to spojili



(Vylepšení): Udržuju si v kořenech hloubky keříku, při Union připojím mělčí pod hlubší.

Lemma: Keřík hloubky h má alespoň 2^h vrcholů



- hloubky jsou ≤ log n

Důsl: Union i Find trvají O(log n)

Složitost alg.:

$$O(\underbrace{m\log n}_{\text{sort}} + \underbrace{m\log n}_{\text{findy}} + \underbrace{n\log n}_{\text{uniony}}) = O(m\log n)$$