

# C2142 Návrh algoritmů pro přírodovědce

## 9. Minimální kostry.

Tomáš Raček

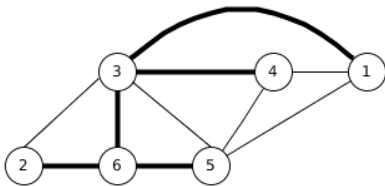
Jaro 2017

# Kostra grafu

---

**Kostra** neorientovaného grafu  $G$  je podgraf, který obsahuje všechny vrcholy  $G$  a je stromem.

**Opakování.** Každá kostra grafu má  $|V|$  vrcholů a  $|V| - 1$  hran.



**Poznámka.** Počet různých koster v úplném grafu je  $n^{n-2}$ .

# Minimální kostra grafu

---

**Minimální kostra** hranově ohodnoceného neorientovaného grafu  $G$  je taková kostra  $G$ , která má součet ohodnocení hran nejnižší.

**Poznámka.** Minimální kostra nemusí být určena jednoznačně. Uvažme např. graf, pro který platí  $\forall \{u, v\} \in E : w_e(u, v) = 1$ .

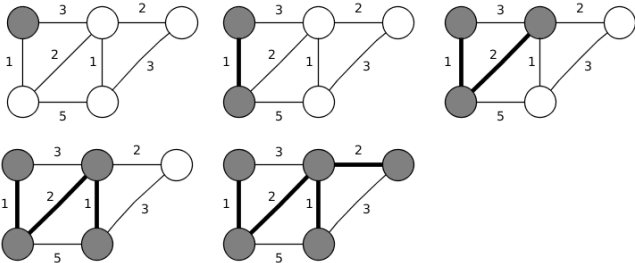
**Zajímavost.** Motivací pro řešení problému minimální kostry byla elektrifikace jižní Moravy (Otakar Borůvka, 1925). Popsáno v článkách:

- Příspěvek k řešení otázky ekonomické stavby elektrovodných sítí
- O jistém problému minimálním

# Primův algoritmus

## Princip

- budování kostry začíná v libovolném vrcholu grafu
- v každém kroce algoritmu je do kostry přidána minimální hrana sousedící s některým již v kostře obsaženým vrcholem tak, aby nebyl utvořen cyklus



# Primův algoritmus – poznámky

---

**Zajímavost.** Tento algoritmus nejprve popsal český matematik Vojtěch Jarník (1930), později (1957, 1959) byl znovuobjeven nezávisle na sobě Robertem Primem a Edsgerem Dijkstrou.

## Implementace

- potřeba udržovat seznam hran, které sousedí s aktuálně zpracovanými vrcholy kostry → rozdělení vrcholů na dvě množiny
- výběr minimální hrany – struktura podobná jako u Dijkstrova algoritmu

# Primův algoritmus – pseudokód

---

```
1: function Prim( $G = (V, E, w_e)$ ,  $s$ ) is
2:    $\forall v \in V : v.key \leftarrow \infty$ 
3:    $s.key \leftarrow 0, s.p \leftarrow NULL$ 
4:    $Q \leftarrow V$ 
5:   while  $|Q| \neq 0$  do
6:      $u \leftarrow t \in Q$  s minimálním  $.key$ 
7:      $Q \leftarrow Q \setminus \{u\}$ 
8:     for all  $\{u, v\} \in E$  do
9:       if  $v \in Q \wedge w_e(u, v) < v.key$  then
10:         $v.key \leftarrow w_e(u, v)$ 
11:         $v.p \leftarrow u$ 
12:      fi
13:    done
14:  done
15: end
```

# Primův algoritmus – volba datové struktury

---

**Pozorování.** Složitost Primova algoritmu je dána volbou datové struktury pro  $Q$ .

## Seznam vrcholů

- odstranění minima –  $O(|V|)$
- snížení klíče –  $O(1)$
- celkem  $O(|V|^2 + |V|) = O(|V|^2)$

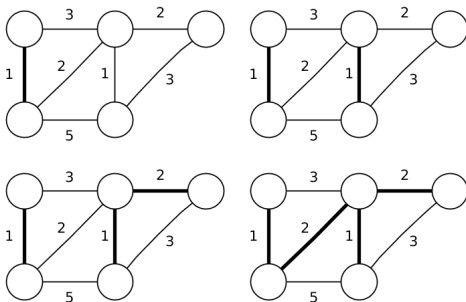
## Binární halda

- odstranění minima –  $O(\log |V|)$
- snížení klíče –  $O(\log |V|)$
- celkem  $O(|V| \log |V| + |E| \log |V|) = O(|E| \log |V|)$

# Kruskalův algoritmus

## Princip

- každý vrchol v grafu představuje jednu komponentu kostry
- v každém kroku jsou dvě komponenty spojeny minimální hranou





# Kruskalův algoritmus – poznámky

---

**Myšlenka.** Seřadím hrany podle jejich ohodnocení. V tomto pořadí je budu uvažovat pro zařazení do kostry.

**Pozorování.** Algoritmus musí udržovat informaci o tom, v jaké komponentě se který vrchol nachází. Nelze totiž vybrat do kostry hranu, která spojuje vrcholy v rámci stejné komponenty.

**Implementace** využívá tři pomocných funkcí:

- **MakeSet( $u$ )** vytvoří jednoprvkovou množinu obsahující vrchol  $u$
- **FindSet( $u$ )** vrátí identifikátor množiny obsahující vrchol  $u$
- **Union( $u, v$ )** sloučí množiny obsahující vrcholy  $u$  a  $v$

# Kruskalův algoritmus – pseudokód

---

```
1: function Kruskal( $G = (V, E, w_e)$ ) is
2:   mst  $\leftarrow \emptyset$ 
3:   for all  $v \in V$  do
4:     MakeSet( $v$ )
5:   done
6:   for all  $\{u, v\} \in E$  od nejmenší podle  $w_e$  do
7:     if FindSet( $u$ )  $\neq$  FindSet( $v$ ) then
8:       mst  $\leftarrow$  mst  $\cup$   $\{\{u, v\}\}$ 
9:       Union( $u, v$ )
10:    fi
11:  done
12:  Vrať mst
13: end
```

# Jednoduchá struktura Union-Find

---

**Myšlenka.** Každá množina bude reprezentována spojovým seznamem.

- **MakeSet( $u$ )** vytvoří jednoprvkový spojový seznam obsahující vrchol  $u$
- **FindSet( $u$ )** vrátí první prvek seznamu obsahující vrchol  $u$
- **Union( $u, v$ )** sloučí dva seznamy obsahující vrcholy  $u$  a  $v$

**Pozorování.** Operace **FindSet** má lineární složitost, ale je v rámci Kruskalova algoritmu volána nejčastěji.

**Vylepšení.** U každého prvku seznamu přidáme navíc ukazatel na hlavu seznamu. **FindSet** bude nyní konstantní, **Union** bude muset přepisovat všechny tyto ukazatele.

# Optimální verze Union-Find

---

**Myšlenka.** Každou množinu reprezentujeme jako strom. Složitosti operací budou závislé na jejich výšce.

- **MakeSet(u)** vytvoří jednovrchový strom s kořenem  $u$
- **FindSet(u)** vrátí kořen stromu obsahující vrchol  $u$
- **Union(u, v)** sloučí dva stromy obsahující vrcholy  $u$  a  $v$

**Vylepšení** snižující složitost operací

- **Union** napojuje vždy strom s nižší výškou pod kořen druhého
- **FindSet** navíc snižuje výšku prohledávaného stromu napojením vrcholů přímo pod kořen

**Složitost Kruskalova algoritmu** při využití této struktury je určena nutností seřazení hran podle jejich ohodnocení, tedy  $O(|E| \log |E|) = O(|E| \log |V|)$ .