Pokročilá algoritmizace

topologické uspořádání, hledání minimální kostry grafu, Union-Find problém

Jiří Vyskočil, Marko Genyg-Berezovskyj

2010

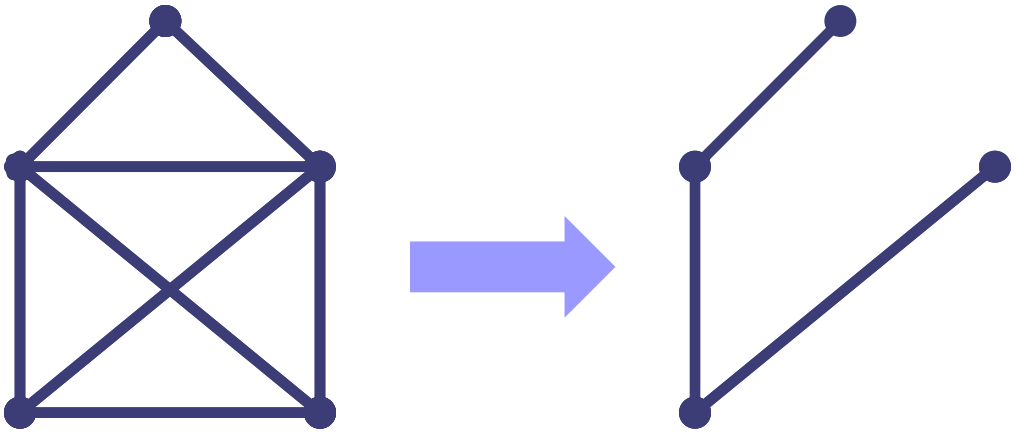
Podgraf

- podgraf
  - Graf H je ***podgraf*** grafu G, jestliže platí následující dvě inkluze:

$$V(H) \subseteq V(G)$$

$$E(H) \subseteq E(G) \cap \binom{V(H)}{2}$$

  - Jinými slovy, podgraf vznikne:
    - vymazáním některých vrcholů původního grafu
    - všech hran do těchto vrcholů zasahujících a případně některých dalších hran.

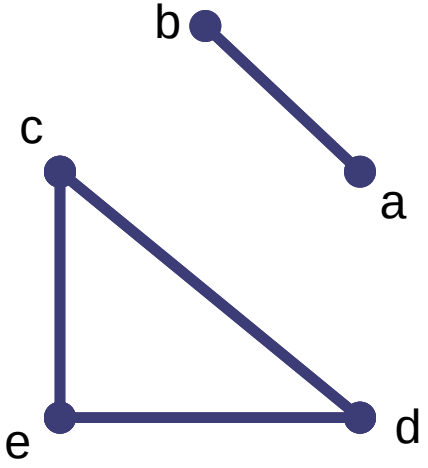


Komponenta souvislosti

- komponenta souvislosti grafu $G=(V,E)$ určená vrcholem v je množina

  $C(v)=\{u \in V \mid \exists \text{ cesta v } G \text{ z } u \text{ do } v\}$.

- jinými slovy: Pokud graf není souvislý, části, ze kterých se skládá a které jsou samy o sobě souvislé, se nazývají *komponenty souvislosti.*



$C(a)=C(b)=\{a,b\}$

$C(c)=C(d)=C(e)=\{c,d,e\}$

DFS pro celý graf rekurzivně

- **vstup:** Graf G.

```
1)  procedure DFS (Graf G) {
2)    for each Vrchol v in V(G) { stav[v] = NENAVŠTÍVENÝ; p[v] = null; }
3)    čas = 0;
4)    for each Vrchol v in V(G)
5)      if (stav[v] == NENAVŠTÍVENÝ) then DFS-Projdi(v);
6)  }

7)  procedure DFS-Projdi(Vrchol u) {
8)    stav[u] = OTEVŘENÝ; d[u] = ++čas;
9)    for each Vrchol v in Sousedí_s u
10)     if (stav[v] == NENAVŠTÍVENÝ) then {p[v] = u; DFS-Projdi(v); }
11)   stav[u] = UZAVŘENÝ; f[u] = ++čas;
12) }
```

- **výstup:** pole p ukazující na předchozí vrchol, pole d s časy otevření vrcholu a pole f s časy uzavření vrcholu.

Topologické uspořádání

- topologické uspořádání vrcholů grafu
  - Mějme graf G, který je DAG. Definujme binární relaci R **topologického uspořádání** nad vrcholy grafu G takovou, že R(*x*,*y*) platí, právě když z *x* vede orientovaná cesta do *y*.
  - Jinými slovy: Očíslujeme-li všechny vrcholy grafu G tak, že pro každé dva vrcholy *x* a *y* platí:

    $x \leq y$, právě když z *x* vede orientovaná cesta do *y*.

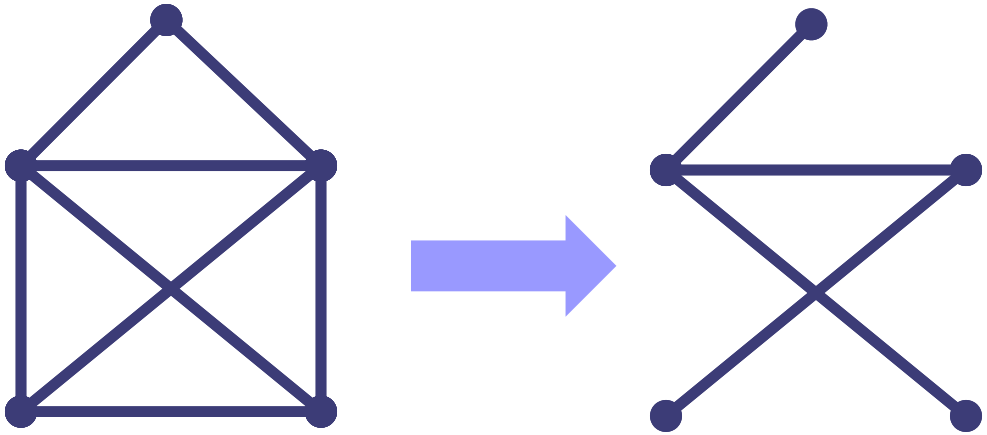
    Potom relace $\leq$ je topologické uspořádání nad grafem G s očíslovanými vrcholy.
- implementace pomocí předchozího DFS algoritmu
  - Očíslování vrcholů polem `f` s relací $\leq$ je topologické uspořádání.

Další využití mírně modifikovaného DFS

- zjišťování acykličnost grafu
- zjišťování souvislosti grafu
- hledání komponent souvislosti grafu
- převod grafu na orientovaný les

Kostra grafu

- kostra grafu
  - Nechť G=(V,E) je graf. ***Kostra grafu G*** je podgraf H grafu G takový, že V(G)=V(H) a H je strom.

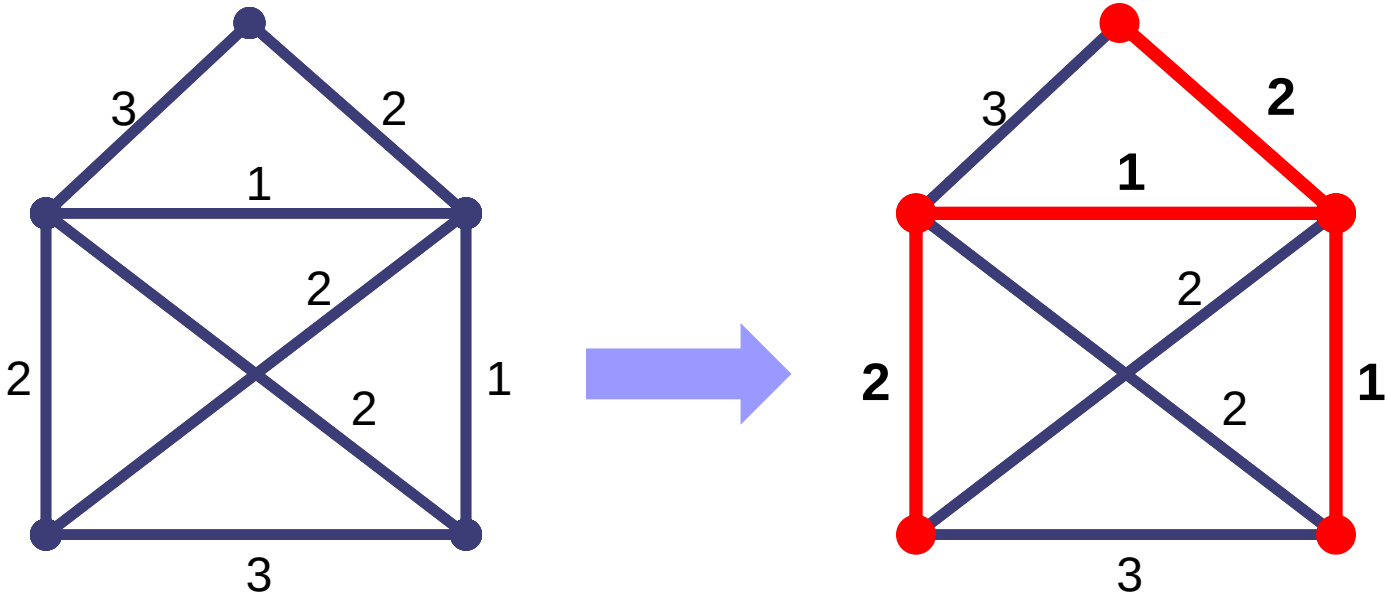


Minimální kostra grafu

- minimální kostra grafu
  - Nechť $G=(V,E)$ je graf a $w: E \to \mathbb{R}$ je jeho váhová funkce.
  - ***Minimální kostra grafu*** G je taková kostra $K=(V,E_K)$ grafu G, že

$$\sum_{e \in E_K} w(e) = w(K)$$

je minimální.



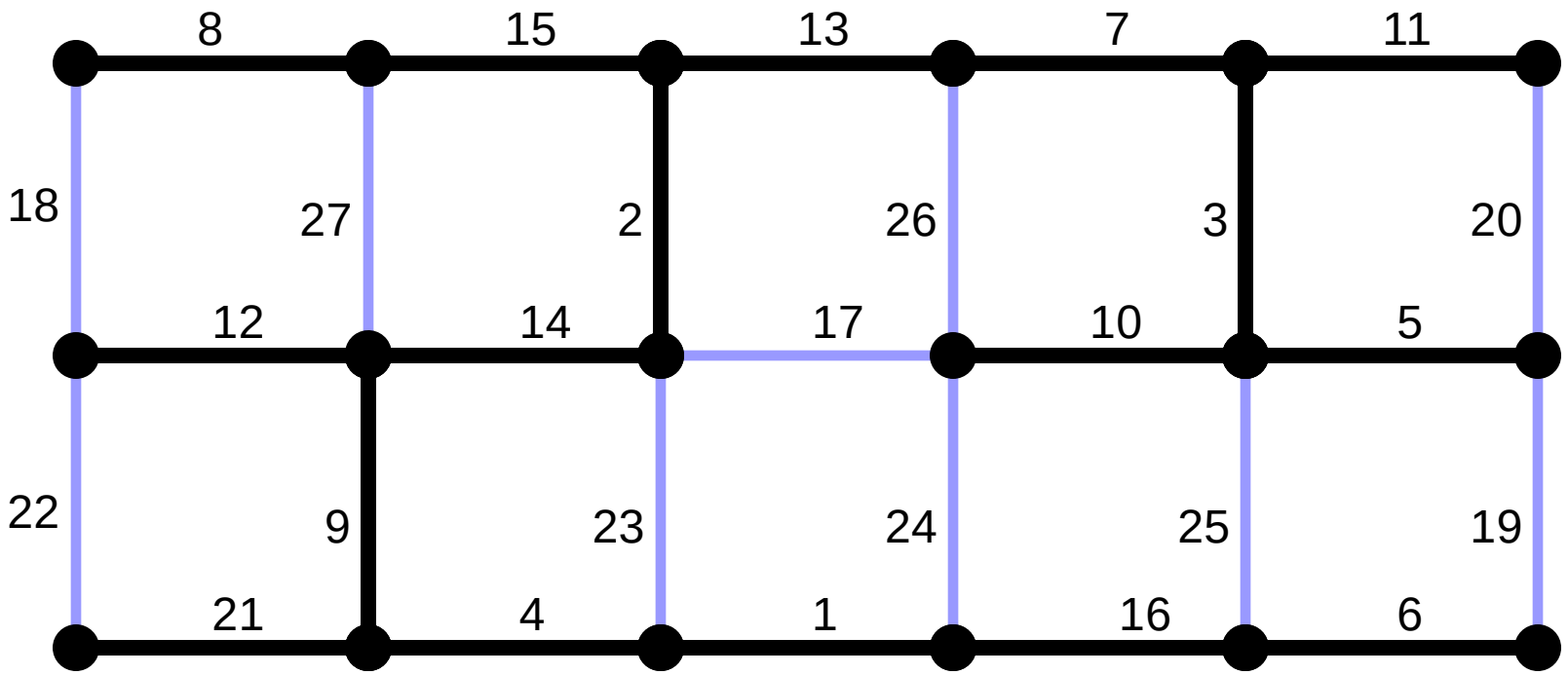
Řez v grafu

- řez
  - Řez v grafu $G = (V,E)$ je množina hran $F \subseteq E$ taková, že $\exists U \subset V: F = \{\{u,v\} \in E \mid u \in U, v \notin U\}$.
- tvrzení: Pokud G je graf, w jeho prosté ohodnocení, F je řez v grafu G a f je nejlehčí hrana v řezu F, pak pro každou minimální kostru K grafu G je $f \in E(K)$.
  - Důkaz sporem: Buď K kostra a $f = \{u,v\} \notin E(K)$. Pak existuje cesta $P \subseteq K$ spojující u a v. Cesta musí řez alespoň jednou překročit. Proto existuje $e \in P \cap F$ a navíc víme, že $w(f) < w(e)$. Uvažme $K' = K - e + f$. Tento graf je rovněž kostra grafu G, protože odebráním hrany e se graf rozpadne na dvě komponenty souvislosti a přidáním hrany f se tyto komponenty opět spojí. Navíc $w(K') = w(K) - w(e) + w(f) < w(K)$.

Jarníkův (Primův) algoritmus

- **vstup:** Graf G s ohodnocením $w : \mathrm{G}(E) \to \mathbb{R}$.
  1) Zvolíme libovolný vrchol $v_0 \in V(\mathrm{G})$.
  2) $\mathrm{K} := (\{v_0\}, \emptyset)$.
  3) **while** $|V(\mathrm{K})| \neq |V(\mathrm{G})|$ {
  4) Vybereme hranu $\{u,v\} \in E(\mathrm{G})$, kde $u \in V(\mathrm{K})$ a $v \notin V(\mathrm{K})$ tak, aby $w(\{u,v\})$ byla minimální.
  5) $\mathrm{K} := \mathrm{K}+$ hrana $\{u,v\}$.
  6) }
- **výstup:** Minimální kostra K.

Jarníkův (Primův) algoritmus



Jarníkův (Primův) algoritmus

- tvrzení: Jarníkův algoritmus se zastaví po max. $|V(G)|$ krocích a vydá minimální kostru grafu G.
  - Při každé iteraci algoritmus přidá jeden vrchol do K, a proto se po maximálně $|V(G)|$ iteracích zastaví.
  - Výsledný graf K je strom, protože se stále přidává list k již existujícímu stromu. Navíc má K $|V(G)|$ vrcholů – tedy je to kostra.
  - Hrany mezi vrcholy stromu K a zbytkem grafu G tvoří řez a algoritmus nejlehčí hranu tohoto řezu přidá do K. Podle předhozího tvrzení tedy všechny hrany K musí být součástí každé minimální kostry a jelikož K je strom, musí být minimální kostrou.

Jarníkův (Primův) algoritmus

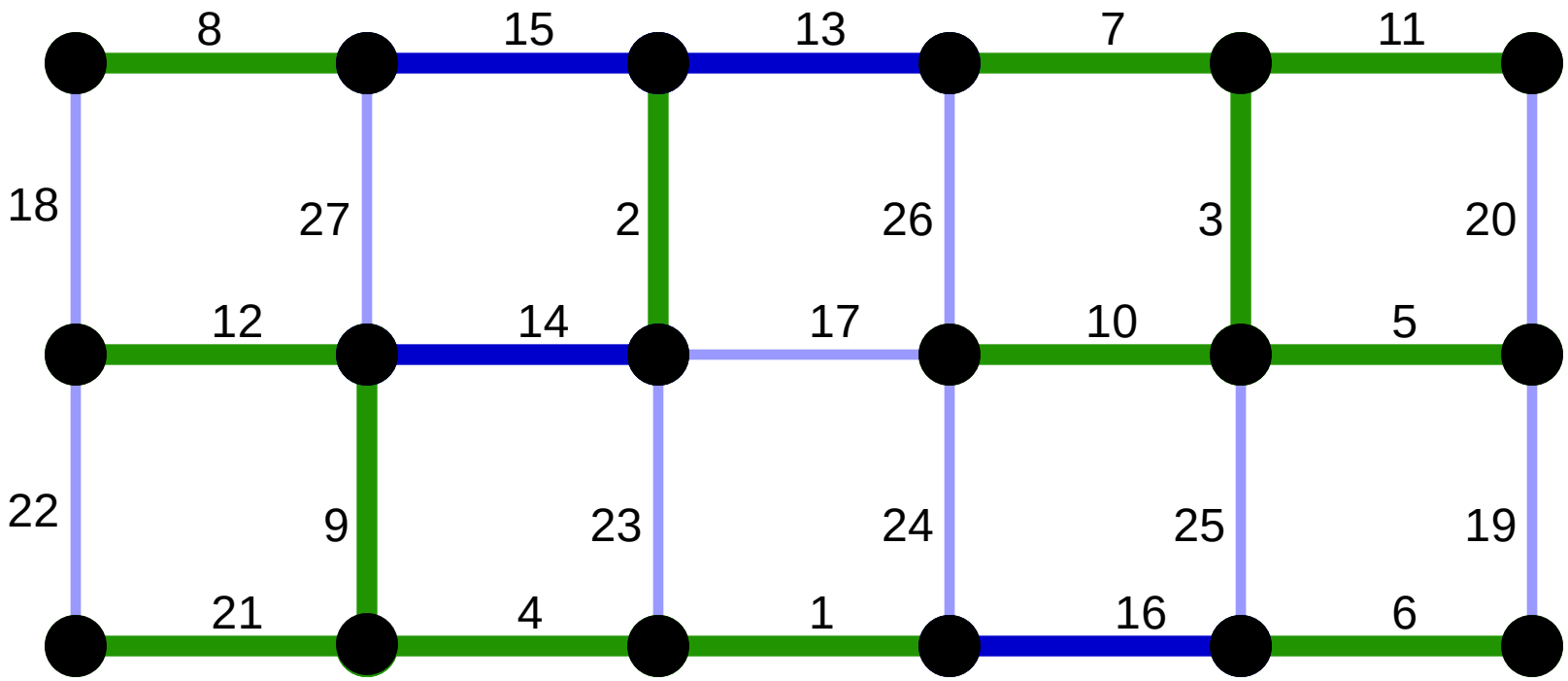
- implementace:
  - „přímočará”
    - Pamatujeme si, které vrcholy a hrany jsou v kostře K a které ne.
    - Časová složitost je $O(n \cdot m)$ kde $n = |V(\mathrm{G})|$ a $m = |E(\mathrm{G})|$.
  - vylepšení
    - Pro $v \notin V(\mathrm{K})$ si pamatujeme $D(v) = \min\{w(\{u,v\}) \mid u \in \mathrm{K}\}$. Při každém průchodu hlavním cyklem pak procházíme všechna $D(v)$ (to vždy trvá $O(n)$) a při přidání vrcholu do K kontrolujeme okolní $D(s)$ pro $\{v,s\} \in E$ a případně je snižujeme (za každou hranu $O(1)$).
    - Časovou složitost tím celkově zlepšíme na $O(n^2 + m) = O(n^2)$.
    - Výslednou časovou složitost lze ještě významně vylepšit použitím vhodného druhu haldy.

Borůvkův algoritmus

- **vstup:** Graf G s ohodnocením $w: G(E) \to \mathbb{R}$, kde všechny váhy jsou různé.
  1) $K := (V(G), \emptyset)$.
  2) **while** K má alespoň dvě komponenty souvislosti **{**
  3) Pro každou komponentu T_i grafu K vybereme ***nejlehčí incidentní hranu***[1] t_i.
  4) Všechny hrany t_i přidáme do K.
  5) **}**
- **výstup:** Minimální kostra K.

[1] ***nejlehčí incidentní hrana*** je hrana, která spojuje komponentu souvislosti T_i s nějakou jinou komponentou souvislosti a váha této hrany je nejmenší.

Borůvkův algoritmus



Borůvkův algoritmus

- tvrzení: Borůvkův algoritmus se zastaví po max. $\lceil \log_2 |V(\mathrm{G})| \rceil$ iteracích a vydá minimální kostru grafu G.
  - Po k iteracích mají všechny komponenty grafu K minimálně 2^k vrcholů.
    - indukcí: Na počátku jsou všechny komponenty jednovrcholové.

      V každé další iteraci se komponenty slučují do větších (každá s alespoň jednou sousední), takže se velikosti komponent minimálně zdvojnásobí.
  - Proto nejpozději po $\lceil \log_2 |V(\mathrm{G})| \rceil$ iteracích už velikost komponenty dosáhne počtu všech vrcholů a algoritmus se zastaví.
  - Hrany mezi každou komponentou souvislosti a zbytkem grafu tvoří řez, takže podle řezového tvrzení všechny hrany přidané do K musí být součástí (jednoznačně určené) minimální kostry. Graf $\mathrm{K} \subseteq \mathrm{G}$ je tedy vždy les (= množina navzájem nepropojených stromů) a až se algoritmus zastaví, bude roven minimální kostře.

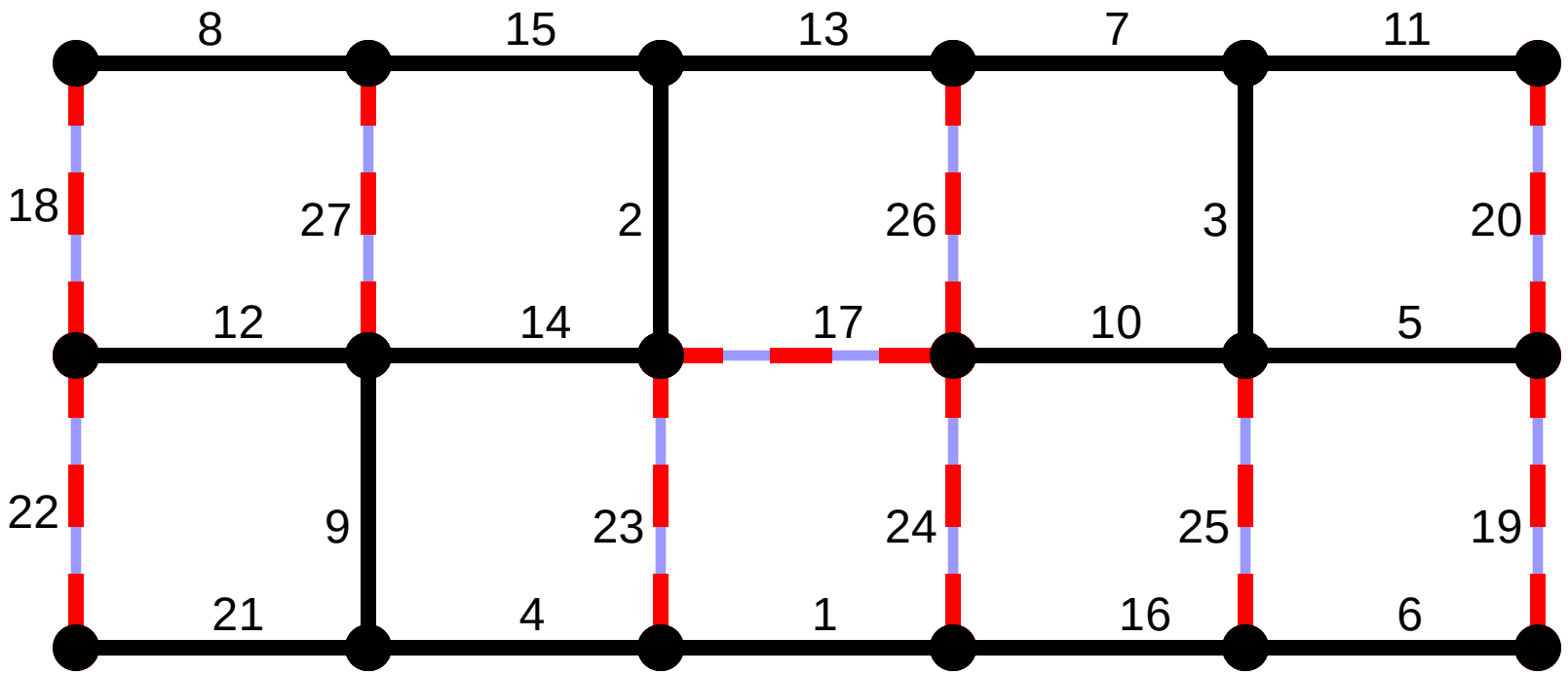
Borůvkův algoritmus

- implementace iterace:
  - Pomocí DFS rozložíme les na komponenty souvislosti. U každého vrcholu si pamatujeme číslo komponenty.
  - Pro každou hranu zjistíme, do které komponenty patří, a pro každou komponentu si uchováme nejlehčí hranu.
  - Takto dokážeme každou iteraci provést v čase $O(|E(G)|)$ a celý algoritmus tedy doběhne v $O(|E(G)| \cdot \log|V(G)|)$.

Kruskalův („hladový“) algoritmus

- **vstup:** Graf G s ohodnocením $w : \mathrm{G}(E) \to \mathbb{R}$.
  1) Setřídíme všechny hrany $e_1, \ldots, e_{m=|E(\mathrm{G})|}$ z $E(\mathrm{G})$ tak, aby $w(e_1) \leq \ldots \leq w(e_m)$.
  2) $\mathrm{K} := (V(\mathrm{G}), \emptyset)$.
  3) **for** $i := 1$ **to** m {
  4) **if** K+hrana $\{u,v\}$ je acyklický graf **then** $\mathrm{K} := \mathrm{K}+\text{hrana } \{u,v\}$.
  5) }
- **výstup:** Minimální kostra K.

Kruskalův („hladový“) algoritmus



Kruskalův („hladový“) algoritmus

- tvrzení: Kruskalův algoritmus se zastaví po $|E(G)|$ iteracích a vydá minimální kostru.
  - Každá iterace algoritmu zpracovává jednu hranu, takže iterací je $|E(G)|$.
  - Indukcí dokážeme, že K je vždy podgrafem minimální kostry: prázdné počáteční K je podgrafem čehokoliv (tedy i minimální kostry), každá hrana, kterou pak přidáme, je minimální v řezu oddělujícím nějakou komponentu K od zbytku grafu (ostatní hrany tohoto řezu ještě nebyly zpracovány, a tudíž jsou těžší). Naopak žádná hrana, kterou jsme se rozhodli do K nepřidat, nemůže být součástí minimální kostry, jelikož s hranami, o kterých již víme, že v minimální kostře leží, tvoří cyklus.

Kruskalův („hladový“) algoritmus

- implementace
  - Setřídění je v čase $O(|E(G)| \cdot \log|E(G)|) = O(|E(G)| \cdot \log|V(G)|)$.
  - Pak potřebujeme udržovat komponenty souvislosti grafu K, abychom uměli rychle určit, jestli právě zpracovávaná hrana vytvoří cyklus.
  - Potřebujeme tedy strukturu pro udržování komponent souvislosti, které se $|E(G)|$-krát zeptáme, zda dva vrcholy leží v téže komponentě (tomu budeme říkat operace **Find**), a právě $(|V(G)| - 1)$-krát spojíme dvě komponenty do jedné (operace **Union**).

Union-Find problém

- Mějme graf $G = (V, E)$.

  Řešíme otázku: „Leží vrcholy u a v ve stejné komponentě souvislosti v grafu G?".

  Problému se také někdy říká dynamické udržování komponent souvislosti a nebo problém udržování ekvivalence.

  V každé komponentě souvislosti vybereme jednoho reprezentanta. Pro jednoduchost budeme reprezentanta komponenty $C(v)$ značit $r(v)$, takže pokud u a v leží ve stejné komponentě, tak $r(u) = r(v)$. Úkol můžeme realizovat pomocí operací:

- $\mathbf{FIND}(v) = r(v)$, operace vrátí reprezentanta komponenty souvislosti $C(v)$.
- $\mathbf{UNION}(u, v)$ provede sjednocení komponent souvislosti $C(u)$ a $C(v)$. To odpovídá přidání hrany $\{u, v\}$ do grafu.

Union-Find problém

- jednoduché řešení:
  - Předpokládejme, že všechny vrcholy jsou očíslované čísly 1 až n. Použijeme pole $R[1..n]$, kde $R[i] = r(i)$, tj. číslo reprezentanta komponenty $C(i)$.
  - Operace $\mathbf{FIND}(v)$ pouze vypíše hodnotu $R[v]$ a tedy bude trvat O(1).
  - K provedení $\mathbf{UNION}(u, v)$ najdeme reprezentanty $r(u) = \mathbf{FIND}(u)$ a $r(v) = \mathbf{FIND}(v)$.
    Pokud jsou různí, tak projdeme celé pole R a každý výskyt $r(u)$ prepíšeme na $r(v)$. To nám zabere čas O(n).

Union-Find problém

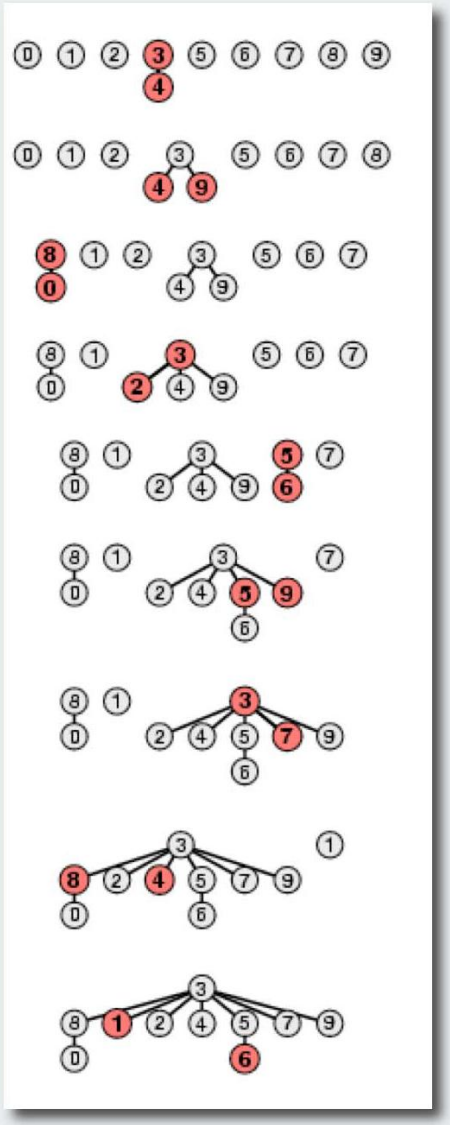
- lepší řešení (pomocí orientovaného stromu):
  - Každou komponentou si uložíme jako strom orientovaný směrem ke kořeni – každý vrchol si pamatuje svého otce, navíc každý kořen si pamatuje velikost komponenty. Kořen každé komponenty bude tedy jejím reprezentantem.
  - Operace $\mathbf{FIND}(v)$ vystoupá z vrcholu v ke kořeni a ten vrátí.
  - K provedení $\mathbf{UNION}(u, v)$ najdeme reprezentanty

    $r(u) = \mathbf{FIND}(u)$ a $r(v) = \mathbf{FIND}(v)$.

    Pokud jsou různí, tak připojíme kořen menší komponenty ke kořeni větší komponenty. V koření nově vzniklé komponenty aktualizujeme její velikost.

Union-Find problém

3-4	0	1	2	3	3	5	6	7	8	9
4-9	0	1	2	3	3	5	6	7	8	3
8-0	8	1	2	3	3	5	6	7	8	3
2-3	8	1	3	3	3	5	6	7	8	3
5-6	8	1	3	3	3	5	5	7	8	3
5-9	8	1	3	3	3	3	5	7	8	3
7-3	8	1	3	3	3	3	5	3	8	3
4-8	8	1	3	3	3	3	5	3	3	3
6-1	8	3	3	3	3	3	5	3	3	3



Union-Find problém

- lepší řešení (pomocí orientovaného stromu):
  - tvrzení: Union-Find strom hloubky h má alespoň 2^h prvků.
  - důkaz indukcí: Pokud **UNION** spojí strom s hloubkou h s jiným stromem s hloubkou menší než h, pak hloubka výsledného stromu zůstává h. Pokud spojuje dva stromy stejné hloubky h, pak má výsledný strom hloubku $h+1$. Z indukčního předpokladu víme, že strom hloubky h má minimálně 2^h vrcholů, a tedy výsledný strom hloubky $h+1$ má alespoň 2^{h+1} vrcholů.
  - důsledek: Složitost operací **UNION** a **FIND** je $O(\log|V|)$.
- nejlepší známé řešení je $O(\alpha|V|)$ pro obě operace, kde funkce α je inverzní Ackermannova funkce.

Kruskalův („hladový“) algoritmus

- složitost Kruskalova algoritmu:
  - Setřídění je v čase $O(|E(G)| \cdot \log|E(G)|) = O(|E(G)| \cdot \log|V(G)|)$.
  - Dále potřebujeme strukturu pro udržování komponent souvislosti, které se $|E(G)|$-krát zeptáme, zda dva vrcholy leží v téže komponentě pomocí operace **Find**, a právě $(|V(G)| - 1)$-krát spojíme dvě komponenty do jedné operací **Union**.
  - Při použití jednoduchého řešení bude celková složitost algoritmu:

    $O(|E(G)| \cdot \log|V(G)| + |E(G)| + |V(G)|^2) = O(|E(G)| \cdot \log|V(G)| + |V(G)|^2)$
  - Při použití lepšího řešení s orientovaným stromem bude celková složitost algoritmu:

    $O(|E(G)| \cdot \log|V(G)| + |E(G)| \cdot \log|V(G)| + |V(G)| \cdot \log|V(G)|) =$
    $O(|E(G)| \cdot \log|V(G)|)$