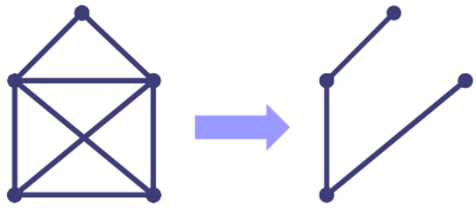
[2] MST problem, Union-Find problem

Základní pojmy, topologické řazení

- **podgraf** (**subgraph**)
  - Graf H je podgrafem grafu G, splňuje-li:

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

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



  - vytvoří se z grafu tedy tak, že:
    - některé vrcholy z G se odstraní
    - odstraní se všechny hrany sousedící s odstraněnými vrcholy (a případně i nějaké další)

- **topologické uspořádání** (**topological ordering**)
  - Nechť je graf G DAG. Definujme pak binární relaci R topologického uspořádání přes vrcholy grafu G tak, že R(x,y) existuje-li orientovaná cesta z *x* do *y* (tzn. *y* je dosažitelný z *x*).
  - Všem vrcholům grafu jsou přiřazena čísla tak, že x <= y platí pro každou dvojici *x* a *y*, existuje-li z *x* do *y* orientovaná cesta.
  - Lze implementovat pomocí následujícího DFS algoritmu, kdy číslování vrcholů pole *f* je topologickým uspořádáním.

- **input:** Graph G.

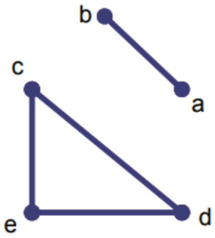
```
procedure DFS (Graph G) {
   for each Vertex v in V(G) {state[v] = UNVISITED; p[v] = null; }
   time = 0;
   for each Vertex v in V(G)
      if (state[ v] == UNVISITED) then DFS-Walk(v);
}

procedure DFS-Walk(Vertex u) {
   state[u] = OPEN; d[u] = ++time;
   for each Vertex v in Neighbors(u)
      if (state[v] == UNVISITED) then {p[v] = u; DFS-Walk(v); }
   state[u] = CLOSED; f[u] = ++time;
}
```

- **output:** array p pointing to predecessor vertex, array d with times of vertex opening and array f with time of vertex closing.

- modifikovaný DFS se dá dále použít pro:
  - testování acyklicity grafu
  - testování souvislosti grafu
  - hledání souvislých komponent grafu
  - transformace grafu na orientovaný les

- **souvislé komponenty**
  - Souvislá komponenta grafu G vzhledem k vrcholu v je množina C(v) = {u ∈ V | existuje cesta v G z u do v}.
  - Čili je-li graf nesouvislý (disconnected), pak ty části, ze kterých se skládá a které jsou samy souvislé, se nazývají souvislými komponentami.

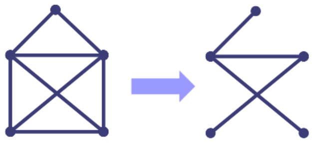


$\mathcal{C}(a)=\mathcal{C}(b)=\{a,b\}$

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

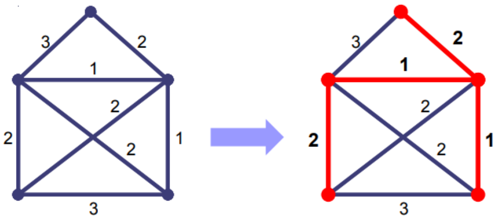
MST (Minimum spanning tree)

- **kostra grafu** (**spanning tree**) = podgraf H grafu G, kde V(G) = V(H) a H je strom



- **minimální kostra** (**minimum spanning tree**)
  - Nechť G = (V,E) je graf a w: E → ℝ je funkce ohodnocení.
  - MST je pak strom K = (V,E_K) grafu G takový, že následující součet je minimální:

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



- **řez v grafu** (**cut of graph**) = podmnožina hran F ⊆ E taková, že:

$$\exists U \subset V: F = \{\{u,v\} \in E \mid u \in U, v \notin U\}$$

- [**Lemma**] Nechť G je graf, w je ohodnocení, F je řez v grafu G a f je nejlehčí hrana řezu F. Pak každá minimální kostra K grafu G obsahuje f ∈ E(K).

Jarník-Primův algoritmus

input: A graph G with a weight function $w: E(G) \to \mathbb{R}$.

Select an arbitrary vertex $v_0 \in V(G)$.

K := $(\{v_0\}, \emptyset)$.

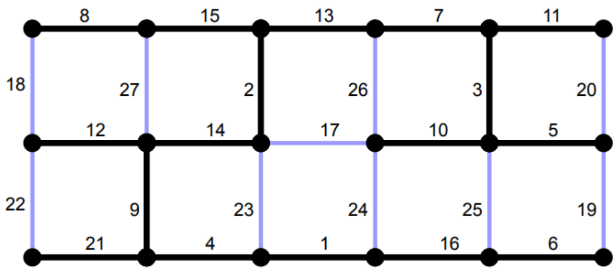
while $|V(K)| \neq |V(G)|$ {

    Select edge $\{u,v\} \in E(G)$,
    where $u \in V(K)$ and $v \notin V(K)$ so that
    $w(\{u,v\})$ is minimum.

    K := K + edge $\{u,v\}$.

}

output: a minimum spanning tree K.



[**Lemma**] Jarníkův algoritmus se zastaví maximálně po |V(G)| krocích a výsledek je MST grafu G.

- V každé iteraci se do K přidá jen jeden prvek, takže cyklus se po |V(G)| iteracích musí zastavit.
- Výsledný graf K je stromem, protože se ke stromu přidá vždy jen list. K má navíc |V(G)| vrcholů, je to tedy kostra grafu.
- Hrany mezi vrcholy K a zbytkem grafu G tvoří řez. Algortimus vždy přidá nejlehčí hranu, všechny hrany z K tedy musí vždy tvořit danou minimální kostru. A jelikož je K strom, musí to být minimální kostra grafu.

- Základní složitost algoritmu je **O(n*m)**, kde n = |V(G)|, m = |E(G)|.
- Lze zrychlit pomocí dodatečných kontrol a užitím vhodného typu haldy až na **O(log(n)*m)**.

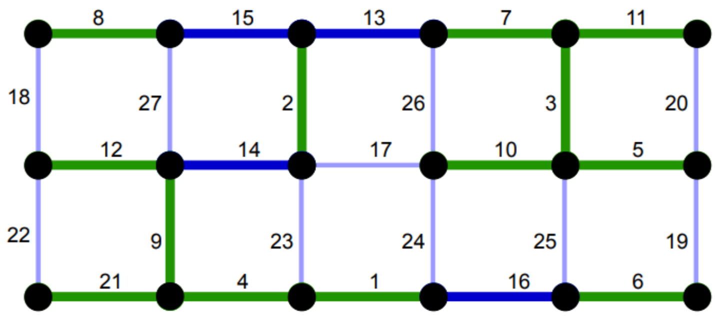
Borůvkův algoritmus

- **input:** A graph G with a weight function $w: E(G) \to \mathbb{R}$, where all weights are **different**.

$K := (V(G), \emptyset)$.
while K has at least two connected components **{**
For all components T_i of graph K the ***light incident edge***[1] t_i is chosen.
All edges t_i are added to K.
}

- **output:** a minimum spanning tree K.

[1] ***A light incident edge*** is an edge connecting a connected component T_i with another connected component while a weight of this edge is the lowest.



[**Theorem**] Borůvka se zastaví maximálně po $\lceil \log_2 |V(G)| \rceil$ iteracích a výsledkem je MST grafu G.

- Po k iteracích mají všechny komponenty grafu K alespoň 2^k vrcholů.
- Po $\lceil \log_2 |V(G)| \rceil$ iteracích musí být tedy velikost každé komponenty alespoň rovna počtu všech vrcholů grafu G, algoritmus se pak zastaví.
- Hrany mezi každou souvislou komponentou a zbytkem grafu tvoří řez. Každé hrany přidané ke K pak musí náležet jedinečné minimální kostře. Graf $K \subseteq G$ je vždy les (tj. množina navzájem nesouvislých stromů) a když algoritmus zastaví, bude se rovnat minimální kostře grafu.

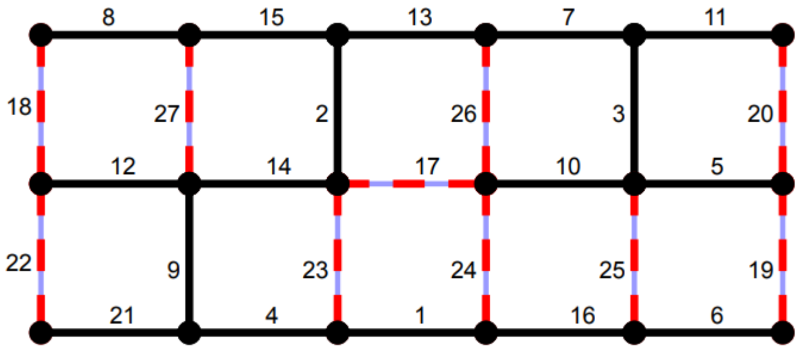
- Celková složitost algoritmu je **O(|E(G)*log₂|V(G)|)**.

Kruskalův (greedy) algoritmus

input: A graph G with a weight function $w: E(G) \to \mathbb{R}$.

1) Sort all edges $e_1, \dots, e_{m=|E(G)|}$ from $E(G)$ so that $w(e_1) \le \dots \le w(e_m)$.
2) K := (V(G), ∅).
3) **for** i := 1 **to** m {
4) **if** K+edge {u,v} is an acyclic graph **then** K := K+edge {u,v}.
5) }

output: a minimum spanning tree K.



[**Theorem**] Kruskal se zastaví po |E(G)| iteracích a vrátí MST.

- Každá iterace zpracuje jednu hranu, počet iterací je tedy |E(G)|.
- Pomocí indukce lze dokázat, že K tvoří vždy podgraf MST a přidání dodatečných hran by tvořilo cykly.

- Složitost řazení je **O(|E(G)|*log|V(G)|)**.
- Celková složitost je pak:
  - **O(|E(G)|*log|V(G)| + |V(G)|^2)** při jednoduché implementaci
  - **O(|E(G)|*log|V(G)|)** při implementaci s orientovaným stromem
    - případně **O(|E(G)|* α|V(G)|)**, viz konec dokumentu

- Cyklus lze zastavit již po přidání |V(G)|-1 hran ke K.
- Je třeba si udržovat přehled o souvislých komponentách grafu K, abychom mohli rychle detekovat případný cyklus u právě zpracovávané hrany.
- Na to potřebujeme nějakou strukturu, které se |E(G)|krát zeptáme, zda-li dva vrcholy náleží ke stejné komponentě (operace **Find**) a |V(G)|-1 krát spojíme dvě komponenty do jedné (operace **Union**).

Union-Find

- Mějme graf G = (V,E) a položme si otázku: Patří vrcholy *u* a *v* do stejné souvislé komponenty grafu G?
- Z každé souvislé komponenty si vybereme jeden uzel – reprezentanta.
  - reprezentant komponenty C(*v*) se značí r(*v*)
  - pokud patří *u* a *v* ke stejné komponentě, pak r(*u*) = r(*v*)
  - lze vyřešit pomocí operací find a union

- **FIND**(v) = r(v) – operace vrátí reprezentanta souvislé komponenty C(v)
- **UNION**(u,v) – spojí souvislé komponenty C(u) a C(v); reflektuje přidání hrany {u,v} do grafu

Jednoduché řešení

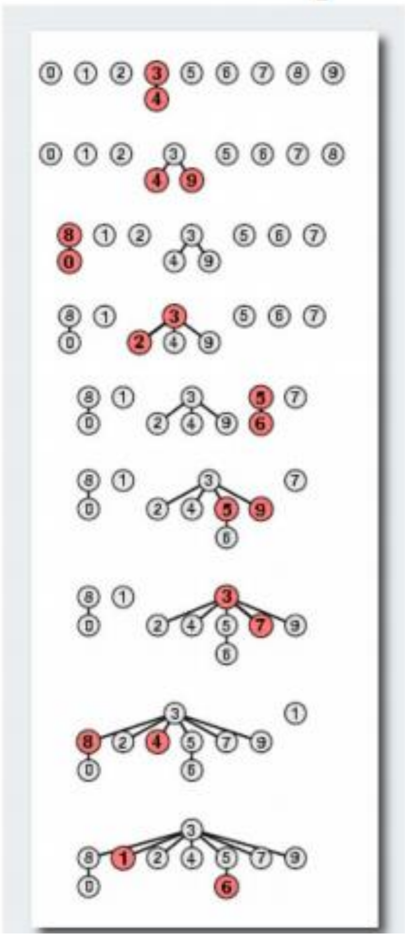
- Předpokládejme, že všem vrcholům jsou přiřazena čísla od 1 do n. Použijeme pole R[1..n], kde R[i] = r(i) je číslo reprezentanta komponenty C(i).
- Operace **FIND**(v) vrátí hodnotu R[v], složitost je tedy O(1).
- U operace **UNION**(u,v):
    - najdeme reprezentanty – r(u) = FIND(u); r(v) = FIND(v)
    - jsou-li různé, pak zpracujeme všechny prvky pole R
    - jakákoliv hodnota r(u) je přepsána na r(v), což zabere O(n)

Vylepšená varianta

- Každá komponenta je uložena jako strom orientovaný ke kořeni.
    - každý uzel má ukazatel na svého předka
    - každý kořen má uloženu velikost komponenty
    - kořen každé komponenty slouží jako její reprezentant

- Operace **FIND**(v) vyšplhá z uzlu v do kořene, který vrátí.
- U operace **UNION**(u,v):
    - najdeme reprezentanty r(u) a r(v).
    - jsou-li různé, kořen menší komponenty sloučíme s kořenem větší komponenty
    - velikost nové komponenty je aktualizována v jejím kořeni

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



- [**Lemma**] Union-Find strom hloubky *h* má nejméně 2^h prvků.
    - lze dokázat indukcí, klíčem je výsledná velikost slučovaných stromů
    - **důsledek**: Časová složitost operací UNION a FIND je **O(log|V|)**.

- Nejlepší známé řešení pro obě operace má složitost **O(α|V|)**, kde α je inverzní Ackermannovou funkcí.

Zdroj: https://cw.fel.cvut.cz/wiki/_media/courses/a4m33pal/2012pal02.pdf

Dobré animace grafových algoritmů lze nalézt např. na Wikipedii.

- https://en.wikipedia.org/wiki/Prim%27s_algorithm
- https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm