Lemma: *Union-Find* strom hloubky h má alespoň 2^h prvků.

Důkaz: Indukcí: Pokud *Union* spojí strom s hloubkou h s jiným 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. ♡

6. Problém minimální kostry

Zadání úlohy: Pro neorientovaný graf G s ohodnocením hran *váhami* $w : E(G) \to \mathbb{R}$, chceme najít kostru T s minimálním ohodnocením $w(T) = \sum_{e \in E(T)} w(e)$.

Navíc předpokládáme: (bez újmy na obecnosti)

- Graf G je souvislý (jinak ho nejprve rozložíme na komponenty).
- $\forall e, f \in E(G) : e \neq f \Rightarrow w(e) \neq w(f)$.

Nyní si ukážeme tři algoritmy pro hledání minimální kostry, konkrétně se jedná o Jarníkův, Borůvkův a Kruskalův algoritmus.

Jarníkův algoritmus

Algoritmus:

1. *Vstup:* Graf G s ohodnocením w.
2. Zvolíme libovolný vrchol $v_0 \in V(G)$.
3. $T \leftarrow (\{v_0\}, \emptyset)$ (zatím jednovrcholový strom)
4. Dokud $|V(T)| \neq n$:
5. Vybereme hranu $uv \in E(G) : u \in V(T), v \notin V(T)$ tak, aby $w(uv)$ byla minimální.
6. $T \leftarrow T + uv$.
7. *Výstup:* Minimální kostra T.

Věta: Jarníkův algoritmus se zastaví po maximálně n iteracích a vydá minimální kostru grafu G.

Důkaz: Při každé iteraci algoritmus přidá jeden vrchol do T, a proto se po maximálně n iteracích zastaví. Vydaný graf je strom, protože se stále přidává list k již existujícímu stromu, a jelikož má n vrcholů, je to kostra. Zbývá nám už jen dokázat, že nalezená kostra je minimální. K tomu pomuže následující lemma:

> **Definice:** *Řez* v grafu $G = (V, E)$ je množina hran $F \subseteq E$ taková, že $\exists U \subset V : F = \{uv \in E : u \in U, v \notin U\}$.
>
> **Lemma (řezové):** 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 T grafu G je $f \in E(T)$.
>
> *Důkaz:* Buď T kostra a $f = uv \notin E(T)$. Pak existuje cesta $P \subseteq T$ spojující u a v. Cesta musí řez alespoň jednou překročit. Proto existuje $e \in P \cap F$ a navíc víme, že $w(e) > w(f)$. Uvažme $T' = T - e + f$. Tento graf je rovněž kostra grafu G, protože odebraním hrany e se graf rozpadne na dvě komponenty a přidáním hrany f se tyto komponenty opět spojí. Navíc $w(T') = w(T) - w(e) + w(f) < w(T)$. ♡

V důkazu korektnosti Jarníkova algoritmu toto lemma využijeme tak, že si všimneme, že hrany mezi vrcholy stromu T a zbytkem grafu tvoří řez a algoritmus nejlehčí hranu tohoto řezu přidá do T. Podle lemmatu tedy všechny hrany T musí být součástí každé minimální kostry a jelikož T je strom, musí být minimální kostrou.

♡

Důsledky: Graf G s prostým ohodnocením má pravě jednu minimální kostru. Minimální kostra je jednoznačně určená lineárním uspořádáním hran.

Implementace:

- Přímočará: pamatujeme si, které vrcholy a hrany jsou v kostře T a které ne. Časová složitost je $\mathcal{O}(nm)$.
- Chytřejší: Pro $v \notin V(T)$ si pamatujeme $D(v) = \min\{w(uv) : u \in T\}$. Při každém průchodu hlavním cyklem pak procházíme všechna $D(v)$ (to vždy trvá $\mathcal{O}(n)$) a při přidání vrcholu do T kontrolujeme okolní $D(w)$ pro $vw \in E$ a případně je snižujeme (za každou hranu $\mathcal{O}(1)$). Časovou složitost tím celkově zlepšíme na $\mathcal{O}(n^2 + m) = \mathcal{O}(n^2)$.
- Také se dá použít halda pro uchovávání hran nebo hodnot $D(v)$.

Borůvkův algoritmus

Algoritmus:

1. *Vstup:* Graf G s ohodnocením w.
2. $F \leftarrow (V(G), \emptyset)$
3. Dokud F má alespoň dvě komponenty:
4. Pro každou komponentu T_i grafu F vybereme nejlehčí incidentní hranu t_i.
5. Všechny hrany t_i přidáme do F.
6. *Výstup:* Minimální kostra F.

Věta: Borůvkův algoritmus se zastaví po $\lceil \log_2 n \rceil$ iteracích a vydá minimální kostru grafu G.

Důkaz: Všimněme si nejprve, že po k iteracích mají všechny komponenty grafu F 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 $\lceil \log_2 n \rceil$ iteracích už velikost komponenty dosáhne počtu všech vrcholů a algoritmus se zastaví.

Hrany mezi každou komponentou a zbytkem grafu tvoří řez, takže podle řezového lemmatu všechny hrany přidané do F musí být součástí (jednoznačně určené) minimální kostry. Graf $F \subseteq G$ je tedy vždy les a až se algoritmus zastaví, bude roven minimální kostře. ♡

Implementace:

- Inicializace přímočará.
- Pomocí DFS rozložíme les na komponenty. Pro každý vrchol 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 $\mathcal{O}(m)$ a celý algoritmus doběhne v $\mathcal{O}(m \log n)$.

Kruskalův neboli hladový algoritmus

Algoritmus:

1. *Vstup:* Graf G s ohodnocením w.
2. Setřídíme všechny hrany z $E(G)$ tak, aby: $w(e_1) < ... < w(e_m)$.
3. $F \leftarrow (V(G), \emptyset)$.
4. Pro $i = 1$ do m:
5. Pokud $F + e_i$ je acyklický, provedeme $F \leftarrow F + e_i$.
6. *Výstup:* Minimální kostra F.

Věta: Kruskalův algoritmus se zastaví po m iteracích a vydá minimální kostru.

Důkaz: Každá iterace algoritmu zpracovává jednu hranu, takže iterací je m. Indukcí dokážeme, že F je vždy podgrafem minimální kostry: prázdné počáteční F je podgrafem čehokoliv, každá hrana, kterou pak přidáme, je minimální v řezu oddělujícím nějakou komponentu F od zbytku grafu (ostatní hrany tohoto řezu ještě nebyly zpracovány, a tudíž jsou těžší). Naopak žádná hrana, kterou jsme se rozhodli do F nepřidat, nemůže být součástí minimální kostry, jelikož s hranami, o kterých již víme, že v minimální kostře leží, tvoří kružnici. ♡

Implementace:

- Setřídění v čase $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$.
- Pak potřebujeme udržovat komponenty souvislosti grafu F, abychom uměli rychle určit, jestli právě zpracovávaná hrana vytvoří kružnici. Potřebujeme tedy strukturu pro udržování komponent souvislosti, které se m-krát zeptáme, zda dva vrcholy leží v téže komponentě (tomu budeme říkat operace *Find*), a $(n-1)$-krát spojíme dvě komponenty do jedné (*Union*).

Kruskalův algoritmus tedy poběží v čase $\mathcal{O}(m \log n + mT_f + nT_u)$, kde T_u je čas na provedení jedné operace *Union* a T_f na operaci *Find*.

Jednoduchá struktura pro komponenty: Budeme si pamatovat v poli čísla komponent, ve kterých leží jednotlivé vrcholy. *Find* zvládneme v čase $\mathcal{O}(1)$, ale *Union* bude stát $\mathcal{O}(n)$. Celý algoritmus pak poběží v čase $\mathcal{O}(m \log n + m + n^2) = \mathcal{O}(m \log n + n^2)$.

Chytřejší struktura: Každou komponentou si uložíme jako strom orientovaný směrem ke kořeni – každý vrchol si pamatuje svého otce, navíc každý kořen si pamatuje velikost komponenty. Operace *Find* vystoupá z obou vrcholů ke kořeni a kořeny porovná. *Union* rovněž najde kořeny a připojí kořen menší komponenty pod kořen té větší. Obojí zvládneme v čase lineárním v hloubce stromu a jak si ukážeme, tato hloubka je vždy nejvýše logaritmická, a proto celý Kruskalův algoritmus poběží v čase $\mathcal{O}(m \log n + m \log n + n \log n) = \mathcal{O}(m \log n)$.[⟨1⟩]

[⟨1⟩] Drobnou úpravou bychom mohli dosáhnout daleko efektivnější struktury, ale tu bychom neupotřebili, jelikož by nás stejně brzdilo třídění, a analýza složitosti by byla ... inu, složitější.