

MINIMÁLNÍ KOSTRA GRAFU

Petr Ryšavý

17. září 2025

Katedra počítačů, FEL, ČVUT

MINIMÁLNÍ KOSTRA GRAFU

Vstup:

- Neorientovaný graf $G = (V, E)$
- každá hrana má cenu

Výstup:

- strom $T \subseteq E$ (též. kostra), který propojuje všechny vrcholy grafu a má nejmenší možnou cenu

Předpoklady:

- (pro pohodlnost) Ceny všech hran jsou různé.
- Graf je souvislý.

PRIMŮV ALGORITMUS

```
function MINIMUM-SPANNING-TREE(graph) returns minimum  
spanning tree of graph  
    VISITED  $\leftarrow$  {RANDOMLY-CHOSEN-VERTEX(graph)}  
    T  $\leftarrow$   $\emptyset$   
    while VISITED  $\neq$  V do  
        (u, v)  $\leftarrow$  MINIMIZING-EDGE(SELECT-EDGES(graph,  
u  $\in$  VISITED & v  $\notin$  VISITED), cost(u, v))  
        T  $\leftarrow$  T  $\cup$  {(u, v)}  
        VISITED  $\leftarrow$  VISITED  $\cup$  {v}  
    end while  
    return T  
end function
```

KOREKTNOST PRIMOVA ALGORITMU

Tvrzení *Pro každý graf Primův algoritmus nalezne nejlevnější kostru grafu.*

Dva kroky:

- Dokážeme, že algoritmus nalezne nějakou kostru. (na přednášce vynecháme)
- Ukážeme, že tato kostra je minimální.

Uvažujme hranu e . Pokud existuje řez (A, B) grafu G takový, že e je nejlevnější hrana grafu G , která kříží řez (A, B) , pak e patří do minimální kostry grafu G , pokud je tato kostra unikátní.

Uvažujme hranu e . Pokud existuje řez (A, B) grafu G takový, že e je nejlevnější hrana grafu G , která kříží řez (A, B) , pak e patří do minimální kostry grafu G , pokud je tato kostra unikátní.

Důkaz

IMPLEMENTACE PRIMOVA ALGORITMU

Jaký je čas běhu, pokud Primova algoritmus naimplementujeme naivně podle pseudokódu?

1. $\Theta(m + n)$
2. $\Theta(m \log n)$
3. $\Theta(n^2)$
4. $\Theta(mn)$

Lepší implementace?

- Stále opakujeme operaci hledání minima.
- Nešlo by tyto opakované výpočty urychlit pomocí lepší organizace dat?

- Datová struktura co provádí operace `insert` a `extract-min` v $\mathcal{O}(\log n)$.
- Perfektně vyvážený binární strom.
- V každém vrcholu je splněná vlastnost haldy: velikost klíče je menší než velikosti klíčů potomků.
- `extract-min`- odebereme vrchol, na jeho místo vložíme poslední uzel a probubláme dolů
- `insert`- vložíme na konec a probubláme nahoru
- Dále máme možnost odebrat z prostředku (probublávání nahoru nebo dolů podle potřeby)

Invariant 1 *V haldě máme vrcholy z množiny $V \setminus VISITED$.*

Invariant 1 *V haldě máme vrcholy z množiny $V \setminus VISITED$.*

Invariant 2 *Pro každý $v \notin VISITED$ platí, že $\text{key}[v]$ je nejmenší cena ze všech hran $(u, v) \in E$ s $u \in VISITED$ (popř. ∞ , neexistuje-li taková hrana)*

Invariant 1 *V haldě máme vrcholy z množiny $V \setminus VISITED$.*

Invariant 2 *Pro každý $v \notin VISITED$ platí, že $\text{key}[v]$ je nejmenší cena ze všech hran $(u, v) \in E$ s $u \in VISITED$ (popř. ∞ , neexistuje-li taková hrana)*

Pokud udržíme tyto invarianty pravdivé, pak `extract-min` vede ke správnému vrcholu w^* , který přidáme do `VISITED` v dalším kroku algoritmu.

Jak udržet Invariant 2 pravdivý?

- Mění se množina hran, která přechází hranici z *VISITED* do $V \setminus VISITED$.
- Přidáním v se mohlo snížit minimální skóre.

Jak udržet Invariant 2 pravdivý?

- Mění se množina hran, která přechází hranici z *VISITED* do $V \setminus VISITED$.
- Přidáním v se mohlo snížit minimální skóre.

Když je v přidáno, provedeme následující kroky:

```
for each  $(v, w)$  in  $E$  do  
    odeber  $w$  z haldy  
    přepočítej  $\text{key}[w] = \min\{\text{key}[w], c_{vw}\}$   
    znovu vlož  $w$  do haldy  
end for
```

- $n - 1$ krát provedeme operaci `extract-min`
- Každá hrana způsobí maximálně jednu operaci `delete` a `insert`.
- Čas běhu je tedy

$$\mathcal{O}((n - 1) \log n + m \log n) = \mathcal{O}((m + n) \log n).$$

- Pro nalezení všech hran, co vedou z vrcholu je třeba $\Theta(n)$ práce.
- Nepotřebujeme haldy, nalezení minima stačí v $\Theta(n)$.
- Místo haldy postačuje pole.
- $n - 1$ operací `extract-min`
- Pro každou $\Theta(n)$ práce, celkem tedy

$$\Theta(n^2).$$

Vzorová implementace Primova algoritmu

- Zkuste nejprve naimplementovat Primův algoritmus sami.
- V např. <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/>.

KRUSKALŮV ALGORITMUS

- Budeme hrany procházet jinak, v pořadí rostoucích cen
- Musíme kontrolovat, zda jsme neuzavřeli cyklus


```
function MINIMUM-SPANNING-TREE(graph) returns minimum  
spanning tree of graph  
     $E' \leftarrow \text{SORTED-EDGES}(\textit{graph}, \text{increasing})$   
     $T \leftarrow \emptyset$   
    for each  $e$  in  $E'$  do  
        if ACYCLIC( $T \cup \{e\}$ ) then  
             $T \leftarrow T \cup \{e\}$   
        end if  
    end for  
    return  $T$   
end function
```

Tvrzení *Pro každý graf Kruskalův algoritmus nalezne nejlevnější kostru grafu.*

Tvrzení *Pro každý graf Kruskalův algoritmus nalezne nejlevnější kostru grafu.*

Důkaz

Jaký je čas běhu, pokud Kruskalův algoritmus naimplementujeme naivně podle pseudokódu?

1. $\Theta(m + n)$
2. $\Theta(m \log n)$
3. $\Theta(n^2)$
4. $\Theta(mn)$

Jaký je čas běhu, pokud Kruskalův algoritmus naimplementujeme naivně podle pseudokódu?

1. $\Theta(m + n)$
2. $\Theta(m \log n)$
3. $\Theta(n^2)$
4. $\Theta(mn)$

Jak zrychlit? Nějakou vhodnou datovou strukturou?

UNION-FIND

- Datová struktura, která rozděljuje objekty do několika skupin.
- `find` najde jméno skupiny pro objekt
- `union` sloučí dvě skupiny do jedné

- Datová struktura, která rozděluje objekty do několika skupin.
- `find` najde jméno skupiny pro objekt
- `union` sloučí dvě skupiny do jedné
- využijeme v Kruskalovi, abychom kontrolovali uzavření cyklu (spojení dvou komponent)

- pro každou skupinu vybereme vůdce
- každý vrchol ukazuje na vůdce své komponenty
- $\mathcal{O}(1)$ find, ale $\mathcal{O}(n)$ union

První zlepšení - měníme vůdce menší komponenty

- Během Kruskala dojde maximálně k $\log_2(n)$ změnám stavu každého prvku
- Čas běhu je $\mathcal{O}(m \log_2 n)$
- Potřebujeme pole navíc

- Stromová struktura
- Nyní $\mathcal{O}(n)$ find

- Rank znamená hloubku stromu
- Pod vyšší strom pověsíme ten méně hluboký (s menším rankem)
- Hloubka je vždy nejvýše logaritmická, tj. máme $\mathcal{O}(\log(n))$ `union` a `find`

- Při operaci `find` přepojujeme všechny odkazy na nový kořen
- Čas běhu (Hopcroft-Ullman, Tarjan) je $\mathcal{O}(m \cdot \alpha^{-1}(n))$ amortizovaně

CO VÍME A NEVÍME

- Známe lineární randomizovaný algoritmus v $\mathcal{O}(m)$ (Karger-Klein-Tarjan, JACM, 1995)
- Nevíme jestli existuje lineární deterministický algoritmus
- Nejlepší alg. co známe je $\mathcal{O}(m \cdot \alpha^{-1}(n))$
- Známe optimální deterministický algoritmus (žádný jiný nemůže být asymptoticky rychlejší), ale neznáme jeho čas běhu (Pettie, Ramachandra, JACM, 2002)
- Neexistuje jednoduchý randomizovaný lineární algoritmus (výše zmíněný je redukce na komplikovaný verifikátor kostry)

- heavily inspired by Tim Roughgarden's online courses,
<http://theory.stanford.edu/~tim/videos.html>
- Robert Sedgewick and Kevin Wayne, Algorithms,
<http://algs4.cs.princeton.edu/home/>, namely
<http://algs4.cs.princeton.edu/44sp/>

DĚKUJI ZA POZORNOST.
ČAS NA OTÁZKY!