C2142 Návrh algoritmů pro přírodovědce

9. Minimální kostry.

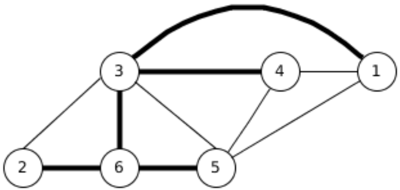
Tomáš Raček

Jaro 2019

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 $\forall \{u, v\} \in E : w_e(u, v) = 1$.

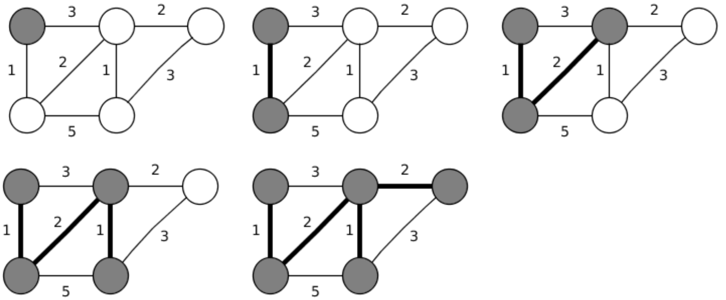
Zajímavost. Motivací pro řešení problému minimální kostry byla elektrifikace jižní Moravy (Otakar Borůvka, 1925). Popsáno v článcích:

- 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 $\rightarrow$ rozdělení vrcholů na dvě množiny
- výběr minimální hrany – struktura podobná jako u Dijkstrova algoritmu

Primův algoritmus – pseudokód

```
function Prim(G = (V, E, w_e), s) is
    ∀v ∈ V : v.key ← ∞
    s.key ← 0, s.p ← NULL
    Q ← V
    while |Q| ≠ 0 do
        u ← t ∈ Q s minimálním .key
        Q ← Q \ {u}
        for all {u, v} ∈ E do
            if v ∈ Q ∧ w_e(u, v) < v.key then
                v.key ← w_e(u, v)
                v.p ← u
            fi
        done
    done
end
```

Primův algoritmus – volba datové struktury

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

Seznam vrcholů

- odstranění minima – $\mathcal{O}(|V|)$
- snížení klíče – $\mathcal{O}(1)$
- celkem $\mathcal{O}(|V|^2 + |E|) = \mathcal{O}(|V|^2)$

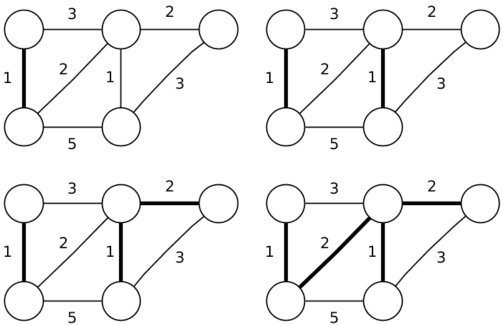
Binární halda

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

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.

Implemetace využívá tří 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

```
function Kruskal(G = (V, E, w_e)) is
    mst ← ∅
    for all v ∈ V do
        MakeSet(v)
    done
    for all {u, v} ∈ E od nejmenší podle w_e do
        if FindSet(u) ≠ FindSet(v) then
            mst ← mst ∪ {{u, v}}
            Union(u, v)
        fi
    done
    Vrať mst
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 jednoprvkový 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 $\mathcal{O}(|E| \log |E|) = \mathcal{O}(|E| \log |V|)$.