MINIMÁLNÍ KOSTRA GRAFU

Petr Ryšavý

18. září 2024

Katedra počítačů, FEL, ČVUT

MINIMÁLNÍ KOSTRA GRAFU

Minimum spanning tree

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.

Příklad

PRIMŮV ALGORITMUS

Pseudokód

function MINIMUM-SPANNING-TREE(*graph*) **returns** minimum spanning tree of *graph*

  VISITED $\leftarrow \{$RANDOMLY-CHOOSEN-VERTEX(*graph*)$\}$

  $T \leftarrow \emptyset$

  while *VISITED* $\neq V$ **do**

    $(u, v) \leftarrow$ MINIMIZING-EDGE(SELECT-EDGES(*graph*, $u \in$ *VISITED* & $v \notin$ *VISITED*), $\mathrm{cost}(u, v)$)

    $T \leftarrow T \cup \{(u, v)\}$

    VISITED $\leftarrow$ *VISITED* $\cup \{v\}$

  end while

  return T

end function

KOREKTNOST PRIMOVA ALGORITMU

Korektnost Primova algoritmu

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

Důkaz

Dva kroky:

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

Cut property

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.

Cut property

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

Příklad

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?

Halda

- 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)

Prim s prioritní frontou.

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

Prim s prioritní frontou.

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

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

Prim s prioritní frontou.

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

Invariant 2 *Pro každý* $v \notin VISITED$ *platí, že* $\mathrm{key}[v]$ *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 $\mathrm{key}[w] = \min\{\mathrm{key}[w], c_{vw}\}$
    znovu vlož w do haldy
end for

Čas běhu (list sousednosti)

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

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

Čas běhu (matice sousednosti)

- Pro nalezení všech hran, co vedou z vrcholu je třeba $\Theta(n)$ práce.
- Nepotřebujeme haldu, 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

Kruskalův algoritmus

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

Příklad

Pseudokód

```
function MINIMUM-SPANNING-TREE(graph) returns minimum
spanning tree of graph
    E' ← SORTED-EDGES(graph, increasing)
    T ← ∅
    for each e in E' do
        if ACYCLIC(T ∪ {e}) then
            T ← T ∪ {e}
        end if
    end for
    return T
end function
```

Korektnost Kruskalova algoritmu

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

Korektnost Kruskalova algoritmu

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

Důkaz

Příklad

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. $\Theta(mn)$

Příklad

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

Union-find

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

Union-find

- 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)

Základní myšlenka

- 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

Druhé (zlepšení) - *lazy union*

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

Třetí zlepšení - *union by rank*

- 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`

State-of-the-art - path compression

- 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

Zajímavosti ze života koster

- 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)

References

- 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!