

DIJKSTRA

Petr Ryšavý

18. září 2019

Katedra počítačů, FEL, ČVUT

DIJKSTRŮV ALGORITMUS

Vstup:

- Orientovaný graf $G = (V, E)$
- každá hrana má **nezápornou** váhu
- počáteční vrchol s

Výstup:

- Pro každý vrchol $v \in V$ spočtěme

$$L(v) = \text{délka nejkratší cesty z } s \text{ do } v.$$

Předpoklady:

- (pro pohodlnost) Vše je dostupné z s .
- Délky hran jsou nezáporné.

Jaké jsou délky nejkratších cest z vrcholu s do vrcholů s, v, w, t v grafu na tabuli?

- 0, 1, 2, 3
- 0, 1, 4, 7
- 0, 1, 4, 6
- 0, 1, 3, 6

Neumíme redukovat problém na BFS?

1. Nepočítá už BFS nejkratší cesty v lineárním čase?

Neumíme redukovat problém na BFS?

1. Nepočítá už BFS nejkratší cesty v lineárním čase?
 - Ano, ale $l_e = 1$ pro všechny hrany $e \in E$

Neumíme redukovat problém na BFS?

1. Nepočítá už BFS nejkratší cesty v lineárním čase?
 - Ano, ale $l_e = 1$ pro všechny hrany $e \in E$
2. Nešlo by nahradit celočíselné váhy cestami jednotkové délky?

Neumíme redukovat problém na BFS?

1. Nepočítá už BFS nejkratší cesty v lineárním čase?
 - Ano, ale $l_e = 1$ pro všechny hrany $e \in E$
2. Nešlo by nahradit celočíselné váhy cestami jednotkové délky?
 - Neškáluje, problém se může značně zvětšit.

Neumíme redukovat problém na BFS?

1. Nepočítá už BFS nejkratší cesty v lineárním čase?
 - Ano, ale $l_e = 1$ pro všechny hrany $e \in E$
2. Nešlo by nahradit celočíselné váhy cestami jednotkové délky?
 - Neškáluje, problém se může značně zvětšit.

Řešením je Dijkstrův algoritmus.

function DIJKSTRA-ALGORITHM(*graph*, *s*) **returns** nejkratší cesty z

s

VISITED \leftarrow {*s*} \triangleright Množina vrcholů *v*, pro které známe $L(v)$

dist[*s*] = 0 \triangleright Spočtená délka cesty

path[*s*] = *emptypath* \triangleright Spočtená cesta

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

 (*v**, *w**) \leftarrow hrana (*v*, *w*) $\in E$ s *v* \in *VISITED*, *w* \notin *VISITED*,

 která minimalizuje

$$dist[v] + l_{(v,w)}$$

VISITED \leftarrow *VISITED* \cup {*w**}

dist[*w**] = *dist*[*v**] + $l_{(v^*,w^*)}$

path[*w**] = *path*[*v**] \cup (*v**, *w**)

end while

end function

KOREKTNOST DIJKSTROVA ALGORITMU

Tvrzení (Dijkstra, 1959) *Pro každý orientovaný graf s nezápornými délkami hran Dijkstrův algoritmus spočte korektně všechny nejkratší cesty z vrcholu s , tj.*

$$\forall v \in V : dist[v] = L(v).$$

Indukcí přes počet iterací.

- Stáhněte si graf, na kterém budeme Dijkstrův algoritmus testovat. Naimplementujte algoritmus podle pseudokódu, který je ve slidech. Snažte se o co nejjednodušší implementaci

IMPLEMENTACE DIJKSTROVA ALGORITMU

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

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

- 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 $key[v]$ je nejmenší Dijkstrovo hladové skóre 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 $key[v]$ je nejmenší Dijkstrovo hladové skóre 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 w se mohlo snížit minimální skóre.

Jak udržet Invariant 2 pravdivý?

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

Když je w přidáno, provedeme následující kroky:

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


- $n - 1$ krát provedeme operaci `extract-min`
- Každá hrana způsobí maximálně jednu dvojici operací `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

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

- Zkuste nejprve naimplementovat Dijkstrův algoritmus sami.
- V Javě např.
`http://keithschwarz.com/interesting/code/?dir=dijkstra`.
- Fibbonacciho halda je pouze pro lepší asymptotickou složitost. Poskytuje stejné operace jako klasická binární halda, pouze je v součtu rychlejší. Implementace je na `http://keithschwarz.com/interesting/code/?dir=fibonacci-heap`. Pokud použijete `PriorityQueue` z knihoven Javy, tak kód bude pomalejší.
- V C++ je implementace např. `http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/`. (Pozor, implementace je tentokrát pro matici sousednosti.)

PŘESTÁVKA

- Začněte nejprve s grafem, který máte na stránkách.
- Nejprve naimplementujte Dijkstraův algoritmus tak, aby používal zabudovanou haldu (**pozor na problém s časovou složitostí**). Poté zkuste použít haldu vlastní.
- Pokud budete chtít něco pokročilejšího, můžete zkusit následující příklady.
- Jednoduchá: UVA 10986 - Sending email
- Středně těžká: UVA 10801 - Lift Hopping
- Těžší: UVA 11635 - Hotel booking

- 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!