DAG A DP V NĚM, BELLMAN-FORD, FLOYD-WARSHALL

Petr Ryšavý

24. září 2018

Katedra počítačů, FEL, ČVUT

TOPOLOGICKÉ OČÍSLOVÁNÍ

Topologické očíslování

Definice (Topologické očíslování) ***Topologické očíslování*** *orientovaného grafu* $G = (V, E)$ *je bijekce* $f : V \mapsto 1, 2, \ldots, k$ *taková, že*

$$(u, v) \in E \quad \Rightarrow \quad f(u) < f(v).$$

Proč nás zajímá topologické očíslování?

- V jakém pořadí je třeba vystudovat předměty, abychom měli splněné prerekvizity.
- V jakém pořadí skládat výrobek.
- Aplikace v mnoha algoritmech.

Existence topologického očíslování

Věta *Graf má topologické očíslování právě tehdy, když neobsahuje orientovaný cyklus.*

Grafy bez orientovaných cyklů se nazývají acyklické, angličtině *directed acyclic* (DAG).

Naivní přístup

- Každý DAG musí obsahovat vrchol, ze kterého nevedou hrany
- Tento vrchol odstraníme (s odpovídajícími hranami) a opakujeme
- Všechny dosažitelné vrcholy musí být v číslování, abychom mohli vrchol odstranit

function $\textsc{Dummy-topsort}$(*graph*) **returns** topological sort f
    $v \leftarrow$ sink vertex
    $f(v) = n$
    $\textsc{Dummy-topsort}(\textit{graph} \setminus \{v\})$
end function

Prohledáváním do hloubky

```
function Topological-Ordering(graph)  returns topological ordering f of the graph
    f ← empty ordering
    currnet-label = Vertices-Count(graph)
    for all node ∈ graph do
        if Unvisited(node) then
            Depth-First-Search(graph, node)
        end if
    end for
end function

function Depth-First-Search(graph, s)
    Mark-Visited(s)
    for all edges (s, v) do
        if Unvisited(v) then
            Depth-First-Search(graph, v)
        end if
    end for
    f(s) = current-label
    current-label = current-label − 1
end function
```

Příklad

Čas běhu a korektnost

- Běží v $\mathcal{O}(m+n)$.

NEJKRATŠÍ CESTY V DAG

Hledání nejkratší cesty v DAG

Nechť $G = (V, E)$ je vážený orientovaný acyklický graf a s je vrchol z V. Nalezněte délky nejkratších cest L z s do každého $v \in V$.

Dynamické programování

- Definujeme jak řešit větší problém pomocí známých řešení podproblémů.
- Podobné rekurzi, ale
  - jdeme odspodu, ne shora a
  - řešení podproblémů máme první.

Základní myšlenka

Známe-li nejkratší cesty do předchůdců vrcholu v, pak stačí vybrat

$$\min_{(u,v)\in E} L(u) + c_{u,v}.$$

Vrcholy projdeme v topologickém pořadí.

Pseudokód

function SSSP-DAG(*graph, s*) **returns** shortest path to each v

$$\mathrm{dist}[v] = \begin{cases} \infty, & v \neq s, \\ 0, & v = s. \end{cases}$$

spočti topologické očíslování grafu

for all $v \in V$ v topologickém pořadí **do**

$$\mathrm{dist}[v] = \min_{(u,v)\in E}\{\mathrm{dist}[u] + c_{u,v}\}$$

end for

end function

Příklad

Jiný přístup

- Hodnoty můžeme „tlačit“ vpřed

function SSSP-DAG(*graph, s*) **returns** shortest path to each v

$$\mathrm{dist}[v] = \begin{cases} \infty, & v \neq s, \\ 0, & v = s. \end{cases}$$

spočti topologické očíslování grafu

for all $(v, w) \in E$ v topologickém pořadí **do**

$$\mathrm{dist}[w] = \min\{\mathrm{dist}[w], \mathrm{dist}[v] + c_{v,w}\}$$

end for

end function

Příklad

- Nejkratší cesta mezi dvěma vrcholy s a t

Analogické algoritmy

- Nejkratší cesta mezi dvěma vrcholy s a t
- Nejdelší cesta v DAG
  - Dva přístupy

Analogické algoritmy

- Nejkratší cesta mezi dvěma vrcholy s a t
- Nejdelší cesta v DAG
  - Dva přístupy
    - $\min$ nahradíme za $\max$
    - ceny vah vynásobíme -1

Analogické algoritmy

- Nejkratší cesta mezi dvěma vrcholy s a t
- Nejdelší cesta v DAG
  - Dva přístupy
    - $\min$ nahradíme za $\max$
    - ceny vah vynásobíme -1
- Počet cest v grafu

V čem je DAG jednodušší

- Čas běhu je $\mathcal{O}(m+n)$ (pro porovnání Dijkstra $\mathcal{O}((m+n)\log n)$
- Umíme řešit i nejdelší cesty (jinak NP-úplný problém)

Vzorová implementace

- Zkuste nejprve naimplementovat hledání nejkratších/nejdelších cest sami.
- https://www.geeksforgeeks.org/find-longest-path-directed-acyclic-graph/
- https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/
- https://cs.stackexchange.com/questions/3078/algorithm-that-finds-the-number-of-simple-paths-from-s-to-t-in-g

Přestávka

ZÁPORNÉ HRANY U NEJKRATŠÍCH CEST

Single-source shortest path

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.

Příklad

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

- $0, 1, 2, 3$
- $0, 1, 4, 7$
- $0, 1, 4, 6$
- $0, 1, 3, 5$

Dijkstrův algoritmus

function DIJKSTRA-ALGORITHM(*graph, s*) **returns** shortest path to each v

$X \leftarrow \{s\}$ ▷ Množina vrcholů, pro které známe $L(v)$

$A[s] = 0$ ▷ Spočtená délka cesty

$B[s] = emptypath$ ▷ Spočtená cesta

while $X \neq V$ **do**

$(v^*, w^*) \leftarrow$ hrana $(v, w) \in E$ s $v \in X$, $w \notin X$, která minimalizuje

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

$X \leftarrow X \cup \{w^*\}$

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

$B[w^*] = B[v^*] \cup (v^*, w^*)$

end while

end function

Příklad

Příklad

Dijkstra na grafu se záporně ohodnocenými hranami.

Zkusme přidat kladnou konstantu ke všem hranám.

Zkusme přidat kladnou konstantu ke všem hranám.

- Problém, že nejkratší cesta nemusí být stále nejkratší.
- Cesty mají rozdílnou délku!

Závěr

- Dijkstra může spočíst špatně nejkratší cestu, pokud graf obsahuje záporně ohodnocené hrany.
- Dijkstrův algoritmus je rychlý, ale ne vždy korektní.
- Dijkstrův algoritmus se špatně distribuuje.
- Řešením je Bellman-Fordův algoritmus

Definice nejkratší cesty v grafu s cykly záporné délky

- Funguje naše dosavadní chápání nejkratší cesty pro graf s cykly záporné délky?

Definice nejkratší cesty v grafu s cykly záporné délky

- Funguje naše dosavadní chápání nejkratší cesty pro graf s cykly záporné délky?
  - Nefunguje. Nejkratší cesta občas neexistuje.

Definice nejkratší cesty v grafu s cykly záporné délky

- Funguje naše dosavadní chápání nejkratší cesty pro graf s cykly záporné délky?
  - Nefunguje. Nejkratší cesta občas neexistuje.
- Zkusme spočíst nejkratší cestu z s do v, která neobsahuje cyklus.

Definice nejkratší cesty v grafu s cykly záporné délky

- Funguje naše dosavadní chápání nejkratší cesty pro graf s cykly záporné délky?
  - Nefunguje. Nejkratší cesta občas neexistuje.
- Zkusme spočíst nejkratší cestu z s do v, která neobsahuje cyklus.
  - NP-těžký problém ...

Definice nejkratší cesty v grafu s cykly záporné délky

- Funguje naše dosavadní chápání nejkratší cesty pro graf s cykly záporné délky?
  - Nefunguje. Nejkratší cesta občas neexistuje.
- Zkusme spočíst nejkratší cestu z s do v, která neobsahuje cyklus.
  - NP-těžký problém ...
- Předpokládejme, že graf neobsahuje cyklus se zápornou délkou.
  - Chceme, aby algoritmus byl schopný takovýto cyklus objevit.

Příklad

Nechť graf G neobsahuje cyklus se zápornou délkou. Pak platí:

1. Pro všechny vrcholy existuje nejkratší cesta s nejvýše $n-1$ hranami.
2. Pro všechny vrcholy existuje nejkratší cesta s nejvýše n hranami.
3. Pro všechny vrcholy existuje nejkratší cesta s nejvýše m hranami.
4. Každá nejkratší cesta může obsahovat libovolný počet hran.

Single-source shortest path, revisited

Vstup:

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

Výstup:

1. Pro každý vrchol $v \in V$ spočtěme

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

nebo

2. Identifikujeme cyklus se zápornou délkou.

BELLMAN-FORDŮV ALGORITMUS

Aplikace DP na SSSP

- Podcesta nejkratší cesty je sama o sobě nejkratší cestou.
- Omezíme problém podle počtu hran v nejkratší cestě.

Aplikace DP na SSSP

- Podcesta nejkratší cesty je sama o sobě nejkratší cestou.
- Omezíme problém podle počtu hran v nejkratší cestě.

Máme tedy jeden podproblém na

1. každý vrchol v
2. počet hran, které připouštíme na nejkratší cestě.

Lemma *Nechť* $G = (V, E)$ *je orientovaný graf s délkami hran* l_e *a počátečním vrcholem* s*. Pro všechny* $v \in V$ *a* $i \in \mathbb{N}$ *označme jako* P *nejkratší cestu z* s *do* v *s nejvýše* i *hranami (cykly jsou povoleny). Pak platí*

1. *pokud má* P *nejvýše* $i - 1$ *hran, pak je i nejkratší cestou z* s *do* v *s nejvýše* $i - 1$ *hranami;*
2. *pokud má* P *právě* i *hran s poslední hranou z* w *do* v*, pak cesta* P' *je nejkratší cestou z* s *do* w *s nejvýše* $i - 1$ *hranami.*

Důkaz

Příklad

Kolik existuje kandidátu pro optimální řešení pro podproblém zahrnující vrchol v a i hran?

1. 2
2. $1 + \mathrm{in} - \mathrm{degree}(v)$
3. $n - 1$
4. n

Bellman-Fordův algoritmus

- Nechť $L_{i,v}$ je délka nejkratší cesty z s do v, která připouští nejvýše i hran.
- Pak $\forall v \in V, i \in \mathbb{N}$ platí

$$L_{i,v} = \min \begin{cases} L_{i-1,v}, \\ \min_{(w,v)\in E}\{L_{i-1,v} + l_{wv}\}. \end{cases}$$

Korektnost vychází z předchozího lemmatu.

Pokud graf G neobsahuje cyklus záporné délky

Pokud graf G neobsahuje cyklus záporné délky, pak

- nejkratší cesty neobsahují cykly,
- mají nejvýše $n - 1$ hran,
- stačí uvažovat pouze i do $n - 1$.

Pseudokód

function BELLMAN-FORD(*graph*, s) **returns** shortest path to each v

$A \leftarrow$ prázdné pole indexované i a v

$A[0,s] = 0$

$\forall v \in V \setminus \{s\} : A[0,v] = \infty$

for $i = 1$ **to** $n-1$ **do**

  for each $v \in V$ **do**

    $A[i,v] = \min\left\{A[i-1,v], \min_{(w,v)\in E}\{A[i-1,w] + l_{wv}\}\right\}$

  end for

end for

end function

Příklad

Příklad

Jaký je čas běhu Bellman-Fordova algoritmu?

1. $\mathcal{O}(n^2)$
2. $\mathcal{O}(mn)$
3. $\mathcal{O}(n^3)$
4. $\mathcal{O}(m^2)$

Předčasné zastavení

- Pokud se nic nezmění pro nějaké $j < n-1$, pak víme, že

$$\forall v \in V : A[j,v] = A[j-1,v]$$

- Nic se nezmění ani pro $j+1$, ani nikdy dále.
- Můžeme zastavit výpočet dříve.

DETEKCE ZÁPORNÝCH CYKLŮ

Co se stane při existenci cyklu záporné délky

Tvrzení G *nemá cyklus záporné délky*

$$\Leftrightarrow$$

pokud přidáme jednu iteraci B-F algoritmu, pak $\forall v \in V : A[n-1, v] = A[n, v]$.

Důkaz

Vzorová implementace Bellman-Fordova algoritmu.

- Zkuste nejprve naimplementovat Bellman-Fordův algoritmus sami.
- V Javě např.
  `http://www.geekviewpoint.com/java/graph/bellman_ford_shortest_path`.
- V C++ je implementace např. `http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/`.

Programovací příklady

- Jednoduchá: UVA 558 - Wormholes
- Středně těžká: UVA 436 - Arbitrage (II) (Řešení je na dalších slidech, ale zkuste na ně přijít sami a nečíst ho.)

Arbitrage detection

- Máme převodní kurzy několika měn.
- Detekujeme cyklus, kde je součin kurzů ≥ 1.

Arbitrage detection

- Máme převodní kurzy několika měn.
- Detekujeme cyklus, kde je součin kurzů ≥ 1.
- Formulujeme jako detekci negativního cyklu po nahrazení délek hran zápornou hodnotou jejich logaritmu.

Přestávka

FLOYD-WARSHALLŮV ALGORITMUS

All-pairs shortest path

Vstup:

- Orientovaný graf $G = (V, E)$
- každá hrana má váhu l_e

Výstup:

1. Pro každou dvojici vrcholů $u, v \in V$ spočtěme délku nejkratší cesty z u do v.

nebo

2. Identifikujeme cyklus se zápornou délkou.

Souvislost s předchozími

APSP se redukuje na n volání SSSP.

Algoritmus	Čas běhu	Přípustná data
n volání Dijkstry	$\mathcal{O}(mn \log n)$	nezáporné délky hran
n volání Bellman-Forda	$\mathcal{O}(mn^2)$	obecný graf
Floyd-Warshall	$\mathcal{O}(n^3)$	obecný graf
Johnson[1]	$\mathcal{O}(mn \log n)$	obecný graf

[1] 1 volání B-F, n volání Dijkstry

Proč další algoritmus pro hledání nejkratších cest?

- Floyd-Warshall funguje i pro záporné délky hran v čase $\mathcal{O}(n^3)$
- Lepší než Bellman-Ford spuštěný n-krát
- V hustých grafech srovnatelný s n běhy Dijkstry
- Dodnes je otevřenou otázkou, zda lze dosáhnout v hustých grafech výrazně lepšího času než $\mathcal{O}(n^3)$

Rekurzivní struktura

- Podobně jako u Bellman-Forda omezíme vrcholy, přes které cesta z s do t může procházet
- Omezíme se na konkrétní výběr vrcholů

Hlavní myšlenka

Lemma *Nechť G neobsahuje cyklus záporné délky a vrcholy $V = \{1, 2, \ldots, n\}$ jsou očíslované a nechť $V^{(k)} = \{1, 2, \ldots, k\}$. Zafixujme počátek $i \in V$ a cíl $j \in V$. Nechť P je nejkratší cesta (bez cyklu) z i do j taková, že všechny vnitřní vrcholy jsou z $V^{(k)}$. Pak platí jedna z následujících možností.*

- *Pokud není k na cestě P, pak P je nejkratší cesta z i do j s vnitřními vrcholy z $V^{(k-1)}$.*
- *Pokud je k vnitřním vrcholem cesty P, pak je zde jednou a P lze rozdělit na dvě podcesty P_1 z i do k a P_2 z k do j. Přitom platí že obě dvě cesty mají vnitřní vrcholy z $V^{(k-1)}$ a jsou nejkratšími cestami bez cyklu mezi odpovídajícími vrcholy.*

Příklad

Nechť A je trojrozměrné pole indexované $A[i, j, k]$ uchovávající délky nejkratších cest z i do j přes vrcholy z $\{1, 2, \ldots, k\}$. Jak bude inicializované $A[i, j, 0]$ pro případy, kdy $i = j$, $(i, j) \in E$ a $(i, j) \notin E$ (v tomto pořadí):

1. $0, 0, \infty$
2. $0, c_{i,j}, c_{i,j}$
3. $0, c_{i,j}, \infty$
4. $\infty, c_{i,j}, \infty$
5. $\infty, c_{i,j}, 0$

Pseudokód

function FLOYD-WARSHALL(*graph*) **returns** minimal path between all pairs of nodes

$A \leftarrow \text{EMPTY-3D-ARRAY}(n, n, n)$

$$A[i, j, 0] = \begin{cases} 0 & \text{if } i = j, \\ c_{ij} & \text{if } (i, j) \in \text{VERTICES}(graph), \\ \infty & \text{otherwise.} \end{cases}$$

for $k = 1$ **to** n **do**

for $i = 1$ **to** n **do**

for $j = 1$ **to** n **do**

$$A[i, j, k] = \min\{A[i, j, k-1], A[i, k, k-1] + A[k, j, k-1]\}$$

end for

end for

end for

return $A[., ., n]$

end function

Detekce cyklu záporné délky

- Co se stane, máme-li v grafu cyklus záporné délky?

Detekce cyklu záporné délky

- Co se stane, máme-li v grafu cyklus záporné délky?
- Alespoň jedno $A[i, i, n]$ bude záporné.

Rekonstrukce cesty

- Stačí si zapamatovat pole $B[i, j]$ udávající nejvyšší vrchol na cestě z i do j
- Víme, že daný vrchol na cestě, je, můžeme ji rozdělit
- při druhém případu, kdy volíme $A[i, j, k] = A[i, k, k-1] + A[k, j, k-1]$ nastavujeme $B[i, j]$ na k

Vzorová implementace Floyd-Warshallova algoritmu.

- Zkuste nejprve naimplementovat Floyd-Warshallův algoritmus sami.
- Implementace např.

  `https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/`.

Programovací příklady

- UVA 423 - MPI Maelstrom
- UVA 186 - Trip Routing

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/`
- Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). Competitive Programming 3. Lulu Independent Publish.

Děkuji za pozornost.

Čas na otázky!