Jarníkův algoritmus

Z Wikipedie, otevřené encyklopedie

Jarníkův algoritmus (v zahraničí známý jako **Primův algoritmus**) je v teorii grafů algoritmus hledající minimální kostru ohodnoceného grafu. Najde tedy takovou podmnožinu hran grafu, která tvoří strom obsahující všechny vrcholy původního grafu a součet ohodnocení hran z této množiny je minimální. Poprvé algoritmus popsal Vojtěch Jarník roku 1930, později byl znovuobjeven roku 1957 Robertem Primem a poté ještě jednou roku 1959 Edsgerem Dijkstrou. V zahraničí se téměř výlučně používá označení *Primův algoritmus*, vzácně pak *Jarníkův algoritmus* nebo *DJP algoritmus*.

Obsah

Popis

Algoritmus začíná s jedním vrcholem a postupně přidává další a tím zvětšuje velikost stromu do té doby, než obsahuje všechny vrcholy.

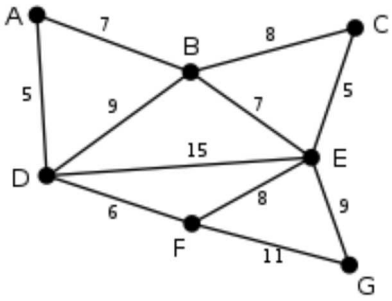
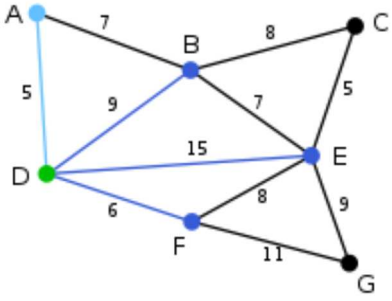
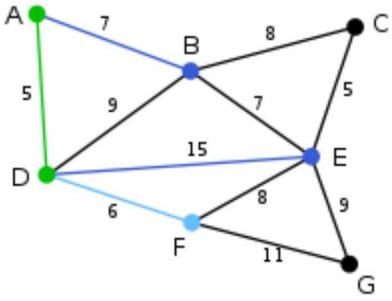
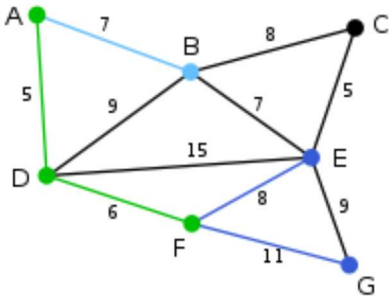
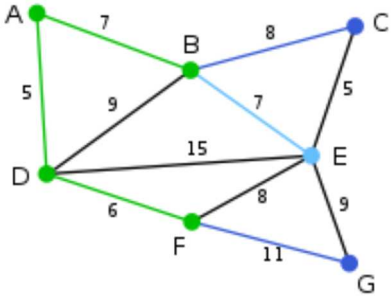
- Vstup: souvislý ohodnocený graf G(V,E)
- Inicializace: V' = {x}, kde x je libovolný vrchol z V, E' = {}
- Opakuj dokud není V'=V:
  - Vyber hranu (u,v) z E s minimální cenou tak, že *u* je ve V' a *v* není ve V'
  - Přidej *v* do V', přidej (u,v) do E'
- Výstup: G(V',E') je minimální kostra grafu

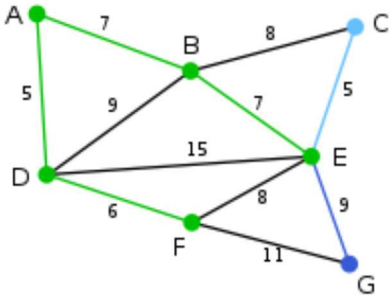
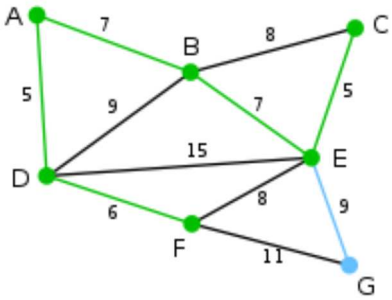
Časová složitost

Datová struktura s ohodnocením hran	Celková časová složitost
matice sousednosti	V^2
binární halda (v pseudokódu níže) a *seznam sousedů*	O((V + E) log(V)) = E log(V)
Fibonacciho halda a *seznam sousedů*	E + V log(V)

Jednoduchá implementace s použitím reprezentace grafu pomocí matice sousednosti a prohledáváním pole cen má časovou složitost $O(V^2)$. S použitím binární haldy a seznamu sousedů dosáhneme složitosti $O(E \log V)$, kde E je počet hran a V je počet vrcholů. S použitím sofistikované Fibonacciho haldy složitost snížíme až na $O(E + V \log V)$, což je obzvláště rychlé u grafů, kde E je $\Omega(V log V)$.

Příklad

Obrázek	Popis	Dosud neviděn	Sousedé	Dosavadní řešení
	Toto je náš původní ohodnocený graf. Není to strom, protože definice stromu požaduje, aby v grafu nebyly žádné kružnice a tento graf kružnice obsahuje. Čísla poblíž hran označují jejich cenu. Žádná hrana zatím není označena a vrchol **D** byl vybrán náhodně jako startovní vrchol.	C, G	A, B, E, F	D
	Druhý vybraný vrchol je nejbližší k vrcholu **D**: cena hrany do **A** je 5, do **B** je 9, do **E** je 15 a **F** je 6. Cena hrany do bodu **A** (5) je nejnižší, použijeme tedy (a v našem diagramu vyznačíme) tuto hranu.	C, G	B, E, F	A, D
	Dalším vybraným vrcholem je nejbližší vrchol buď k **D** nebo k **A**. Cena hrany z **D** do **B** je 9 a z **A** do **B** je 7, do **E** je 15 a do **F** je 6. Cena poslední jmenované hrany je nejnižší, zvolíme tedy hranu z **D** do **F**.	C	B, E, G	A, D, F
	V tomto případě je nejkratší hrana z **A** do **B**.	žádný	C, E, G	A, D, F, B
	V tomto případě je nejkratší hrana z **B** do **E**. Další dvě hrany obarvujeme na červeno, protože je už nebudeme moci použít. Nechceme, aby vznikla kružnice.	žádný	C, G	A, D, F, B, E

	Jediné zbývající vrcholy jsou **C** a **G**. Hrana z **E** do **C** váží 5 a hrana z **E** do **G** váží 9. Vybíráme tedy hranu do **C**. Hranu z **B** do **C** obarvujeme na červeno.	žádný	G	A, D, F, B, E, C
	Zbývá nám už jenom vrchol **G**. Hrana do **F** váží 11 a do **E** 9. Vybíráme tedy hranu do **E**. Všechny vrcholy jsou už součástí stromu, získali jsme tedy minimální kostru grafu (na diagramu je obarvená zelenou). Celková váha kostry je 39.	žádný	žádný	A, D, F, B, E, C, G

Implementace v pseudokódu

S použitím haldy

Inicializace

vstupy: Graf, funkce vracející ohodnocení hran a startovní vrchol

na začátku jsou všechny vrcholy nastaveny na status *dosud neviděn*, startovní vrchol je umístěn do grafu a všechny hrany do haldy tak, aby vracela hranu s nejnižší vahou.

```
Pro každý vrchol v grafu
   nastav min_vzdálenost vrcholu na ∞
   nastav otec vrcholu na null
   nastav min_seznam_sousedů vrcholu na prázdný_seznam
   nastav je_v_Q vrcholu na true
nastav vzdálenost startovního vrcholu na nula
přidej do haldy Q všechny vrcholy v grafu.
```

Algoritmus

V popisu algoritmu výše,

nejbližší vrchol je Q[0]
okraj je *v* v Q, kde vzdálenost *v* < ∞ poté, co je nejbližší vrchol vyjmut z haldy
dosud neviděn je *v* v Q, kde vzdálenost *v* = ∞, co je nejbližší vrchol vyjmut z haldy

Cyklus selže pokud halda vrátí nulu. Seznam sousedů je vytvořen tak, aby mohl vrátit orientovaný graf.

časová složitost: V na cyklus, log(V) na vyjmutí hrany z haldy

```
Dokud poslední_přidaný = vrať minimum v Q
```

```
nastav je_v_Q čeho poslední přidaný na false
přidej poslední_přidaný na (min_seznam_sousedu čeho (otec čeho poslední_přidaný))
přidej (otec čeho poslední_přidaný) do (min_seznam_sousedů čeho poslední_přidaný)
```

časová složitost: E/V, průměrný počet vrcholů

```
pro každý soused čeho poslední_přidaný
Jestliže (je_v_Q čeho soused) a zároveň (váha(poslední_přidaný, soused) < min_vzdálenost čeho soused)
    nastav otec čeho soused na poslední_přidaný
    nastav min_vzdálenost čeho soused na váha(poslední_přidaný, soused)
```

časová složitost: log(V), výška haldy

```
    uprav soused v Q, podle min_vzdálenost
```

Důkaz správnosti

Nechť *P* je souvislý, ohodnocený graf. S každou iterací Jarníkova algoritmu je přidána hrana, která spojuje vrchol v podgrafu s vrcholem mimo podgraf. Jelikož je *P* souvislý, musí existovat cesta mezi všemi dvojicemi vrcholů. Nechť výstup Jarníkova algoritmu je *Y* a Y_1 je minimální kostra grafu *P*. Jestliže Y_1=*Y*, pak *Y* je minimální kostra grafu. V opačném případě, nechť *e* je první hrana přidaná během konstrukce *Y*, která není v Y_1 a *V* je množina vrcholů spojených hranami přidanými před *e*. Pak jeden konec hrany *e* je v *V* a druhý není. Jelikož Y_1 je kostra grafu *P*, pak musí existovat cesta v Y_1 spojující oba konce hrany. Někde v této cestě se musí objevit hrana *f* spojující vrchol ve *V* s vrcholem, který není ve *V*. Ve chvíli, kdy je *e* přidána k *Y* by mohla být místo ní přidána také *f*, pokud by ovšem vážila méně. Jelikož ale byla přidána *e*, můžeme soudit, že

$$w(f) \geq w(e).$$

Nechť Y_2 je graf získaný odstraněním *f* a přidáním *e* z Y_1. Je snadné ukázat, že Y_2 je souvislý, má stejný počet hran jako Y_1 a celková váha hran není vyšší než u Y_1. Y_2 je tudíž minimální kostra grafu *P* a obsahuje *e* a všechny hrany přidané předtím při konstrukci *V*. Opakováním těchto kroků nakonec zjistíme, že minimální kostra grafu *P* je identická s *Y*. Tímto jsme tedy dokázali, že *Y* je minimální kostra grafu.

Reference

V tomto článku byl použit překlad textu z článku Prim's algorithm (https://en.wikipedia.org /wiki/Prim%27s_algorithm?oldid=135413690) na anglické Wikipedii.

Literatura

- V. Jarník: *O jistém problému minimálním*, Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57-63.
- R. C. Prim: *Shortest connection networks and some generalisations*. In: *Bell System Technical Journal*, 36 (1957), pp. 1389–1401 (anglicky)
- D. Cherition and R. E. Tarjan: *Finding minimum spanning trees*. In: *SIAM Journal of Computing*, 5 (Dec. 1976), pp. 724–741 (anglicky)

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 23.2: The algorithms of Kruskal and Prim, pp.567–574. (anglicky)

Externí odkazy

- Obrázky, zvuky či videa k tématu **Jarníkův algoritmus** ve Wikimedia Commons
- Problém minimální kostry grafu: Jarníkův algoritmus (http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Prim.shtml)
- Animovaný příklad Jarníkova algoritmu (http://students.ceid.upatras.gr/~papagel/project/prim.htm)
- Jarníkův algoritmus (Java Applet) (http://www.mincel.com/java/prim.html)
- Prim's algorithm code (http://www.algorithm-code.com/wiki/Prim%27s_algorithm)