

Hledání optimálních sledů

Hledání optimálních sledů se využívá k řešení problémů formalizovatelných hranově ohodnoceným grafem, kde hrany mohou představovat délku, šířku, cenu, dobu, propustnost atd. (jde o hranová ohodnocení).

Pak nás zajímají odpovědi na následující otázky:

- jak nejrychleji?
- jak nejlevněji?
- jak nejvíce?
- jak nejbezpečněji?

Někdy mají své ohodnocení i uzly (nebo-li uzlové ohodnocení), toto ohodnocení může limitovat hledané řešení, např. propustnost uzlu, cena za průchod uzlem atd.

Uzlové hodnoty lze snadno převést na hranové:



Hledání nejkratší cesty v grafu

Algoritmy pro hledání nejkratších cest v grafu můžeme rozdělit do několika tříd:

- z daného uzlu grafu do všech ostatních uzlů grafu (např. *Bellmanův-Fordův algoritmus*, *Dijkstrův algoritmus*,

algoritmus pro acyklické grafy, Moorův algoritmus pro neohodnocené grafy atd.),

- ze všech uzlů do daného uzlu,
- mezi dvojicí uzlů,
- mezi všemi dvojicemi uzlů grafu (např. *algoritmus násobení matic, Floydův-Warshallův algoritmus, Johnsonův algoritmus*).

Záporný cyklus je **cyklus**, ve kterém je součet ohodnocení hran záporný.

Záporný cyklus má smysl uvažovat pouze u grafů, které připouštějí záporné ohodnocení hran. Takovéto grafy je možné nalézt např. v ekonomických aplikacích, kde hrana reprezentuje určitou akci s finančním ohodnocením. Kladné ohodnocení pak může znamenat zisk, záporné ztrátu, případně naopak.

Společné vlastnosti uvedených algoritmů pro hledání nejkratších cest v grafu:

- žádný z **algoritmů** nelze použít na **graf**, který obsahuje **záporný cyklus** !
- po skončení algoritmu získáme nejen cestu ze startovního do cílového uzlu grafu, ale také ze startovního uzlu do všech ostatních uzlů grafu,
- kromě nejkratší cesty (sled v grafu) získáme také hodnotu této cesty (součet ohodnocení hran nejkratší cesty).

Délka nejkratší cesty

Nechť $G = (V, H, f)$ je graf a $u, v \in V$ libovolné dva jeho uzly. Definujme **délku nejkratší cesty** z uzlu u do uzlu v jako:

- $\delta(u, v) = \infty$, jestliže neexistuje cesta z uzlu u do uzlu v ,
- $\delta(u, v) = -\infty$, jestliže nějaká cesta z u do v obsahuje uzel patřící zápornému cyklu,
- $\delta(u, v) = \min\{w(p)\}$ jinak, přičemž $w(p)$ je **délka sledu** p z uzlu u do uzlu v .

Reprezentace nejkratších cest

Nechť $s \in V$ je počáteční uzel při hledání nejkratších cest v grafu $G = (U, H, f)$.

Pro každý uzel $v \in U$ grafu $G = (U, H, f)$ budeme udržovat hodnotu $p[v]$, ve které bude uložen předchůdce uzlu v (tj. uzel, ze kterého jsme do v přišli při hledání nejkratší cesty). Na počátku bude pro každý uzel tato hodnota prázdná, tedy $p[v] = \text{NULL}$.

Dále pro každý uzel v budeme udržovat hodnotu $d[v]$, ve které bude uložená aktuální délka cesty z počátečního uzlu s do uzlu v . Iniciální hodnota je pro $v \neq s$ rovna $d[v] = \infty$ a $d[s] = 0$.

Na konci výpočtu je $d[v] = \delta(s, v)$.

Moorův algoritmus

Algoritmus je založen na prohledávání do šířky.

Každý uzel má značku (p, d) , kde d je délka cesty (počet hran) a p je předcházející uzel. Počáteční uzel s dostane značku $(-, 0)$, ostatní $(-, \text{nekonečno})$. $V_0 = \{s\}$, $k = 0$

Pro $\forall i \in V_k$ uděláme

- každého neoznačovaného následníka uzlu i
 - označujeme $(i, k+1)$
 - a vložíme jej do množiny V_{k+1} ,
- Zvýšíme k o 1, a pokud V_k je neprázdná, opakujeme.

Výsledkem je distanční rozklad množiny uzlů podle počtu hran ležících na nejkratší (vzhledem k počtu hran) cestě od počátečního uzlu.

Bellmanův-Fordův algoritmus

Bellmanův-Fordův algoritmus je založený na postupném zlepšování hodnot $d[u]$. Jakmile uzlu u zlepšíme hodnotu $d[u]$,

musíme následně prozkoumat všechny hrany $(u,v) \in H$, a pokud to bude možné, zlepšit hodnotu $d[v]$ (operace RELAXACE). Algoritmus vrátí hodnotu *true* právě tehdy, když graf neobsahuje cyklus záporné délky dosažitelný z počátečního vrcholu *s*.

Vstupem algoritmu je graf $G = (U, H, f)$ s hranovým ohodnocením *w*, který neobsahuje záporný cyklus (může se ale jednat o graf, který obsahuje cyklus, i graf, který obsahuje záporné délky).

Celý postup zachycuje následující pseudoalgoritmus:

BELLMAN-FORD(*G, w, s*)

1. INICIALIZACE(*G, s*)
2. **for** *i* = 1 to $|U| - 1$ **do**
3. **for** každou hranu $(u,v) \in H$ **do** RELAXACE(*u,v*)
4. **for** každou hranu $(u,v) \in H$ **do**
5. **if** $d[v] > d[u] + w(u,v)$ **then return** false
6. **return** true

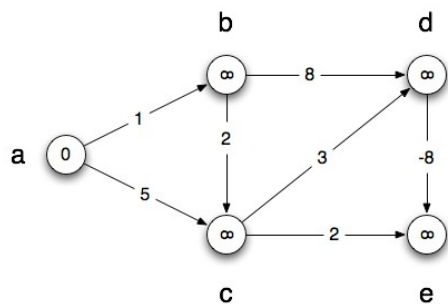
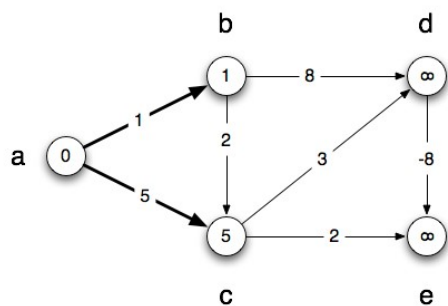
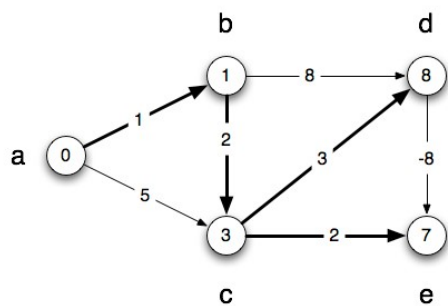
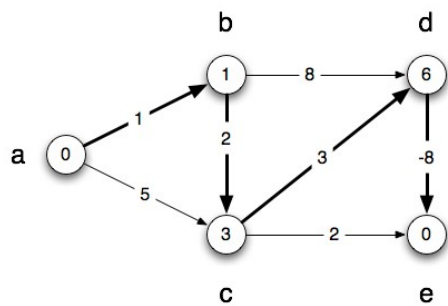
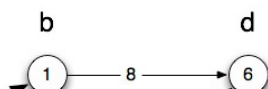
INICIALIZACE(*G,s*)

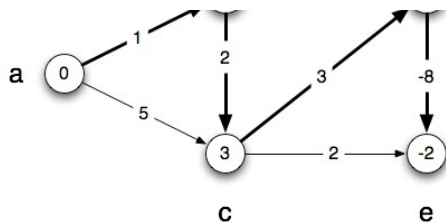
1. **for** každý uzel $v \in U$ **do**
2. $d[v] := \infty$;
3. $p[v] := \text{NULL}$
4. $d[s] := 0$

RELAXACE(*u,v*)

1. **if** $d[v] > d[u] + w(u,v)$
2. **then** $d[v] := d[u] + w(u,v)$;
3. $p[v] := u$;

Následující obrázky zachycují průběh Bellmanův-Fordova algoritmu pro graf s kořenem *a*. V každé iteraci cyklu relaxujeme hrany v pořadí (a,b), (a,c), (b,c), (c,d), (b,d), (d,e) a (c,e).





Program pro vizualizaci Bellman-Fordova algoritmu si můžete stáhnout například z [těchto stránek](#).

Časová složitost Bellmanova-Fordova algoritmu je $O(v \cdot e)$, kde v je počet uzlů a e je počet hran.

Dijkstrův algoritmus

Dijkstrův algoritmus řeší problém nejkratších cest z kořene s (startovací uzel, ze kterého začínáme) do ostatních uzlů grafu pro grafy s nezáporným ohodnocením hran. Algoritmus udržuje množinu uzlů S , pro které už je vypočítána délka nejkratší cesty. Algoritmus opakovaně vybírá vrchol $u \in V - S$, do nějž vede z počátečního uzlu nejkratší cesta a relaxuje hrany vycházející z uzlu u .

Celý postup zachycuje následující pseudoalgoritmus:

DIJSKTRA(G, w, s)

1. INICIALIZACE(G, s)
2. $S := \emptyset$; $Q := V$;
3. **while** $Q \neq \emptyset$ **do**
4. $u := (u \in Q \wedge d[u] = \min\{d[x] \mid x \in Q\})$
5. $S := S \cup \{u\}$
6. **for** každý uzel v takový, že $(u,v) \in H$ **do**
7. RELAXACE(u,v)

Pokud vás zajímá, jaká je hodnota nejkratší cesty z libovolného města v České republice do jiného, podívejte se na [tyto stránky](#). Applet, který zde najdete, pracuje na principu **Dijkstrova algoritmu**.

Časová složitost Dijkstrova algoritmu je $O(n^2)$, kde n je počet uzlů v grafu.

Floydův-Warshallův algoritmus

Floydův-Warshallův algoritmus porovnává všechny možné cesty v grafu mezi všemi dvojicemi uzlů. Pracuje tak, že postupně vylepšuje odhad na nejkratší cestu do té doby, než je zřejmé, že odhad je optimální.

Mějme graf G s uzly V očíslovanými 1 až n . Dále mějme funkci **nejkratsiCesta(i, j, k)**, která vrací nejkratší možnou cestu z i do j s použitím pouze uzlů 1 až k jako mezivrcholů. Pomocí této funkce chceme najít nejkratší cestu mezi všemi dvojicemi i a j s použitím mezivrcholů 1 až $k + 1$.

Na nejkratší cestu máme dva kandidáty:

- buď je nejkratší cesta v množině uzlů $(1 \dots k)$,
- nebo existuje cesta jdoucí z i do $k + 1$, a poté z $k + 1$ do j , která je lepší (kratší) než ta stávající.

Nejlepší cesta z i do j používající pouze uzly 1 až k je definována funkcí **nejkratsiCesta(i, j, k)**. Délka nejlepší cesty z i do $k + 1$ a poté do j je pak zřejmě součtem délek nejkratší cesty z i do $k + 1$ a nejkratší cesty z $k + 1$ do j .

Funkci **nejkratsiCesta(i, j, k)** pak můžeme rekurzivně definovat takto:

$$\begin{aligned} \text{nejkratsiCesta}(i, j, k) &= \min(\text{nejkratsiCesta}(i, j, k - 1), \text{nejkratsiCesta}(i, k, k - 1) + \text{nejkratsiCesta}(k, j, k - 1)); \\ \text{nejkratsiCesta}(i, j, 0) &= \text{delkaHrany}(i, j); \end{aligned}$$

Algoritmus nejprve spočte **nejkratsiCesta($i, j, 0$)** pro všechny dvojice i a j , poté pro všechny dvojice spočte **nejkratsiCesta($i, j, 1$)** atd. dokud nedosáhne $k = n$, kdy jsme našli nejkratší cesty pro všechny dvojice vrcholů i a j v grafu G . Časová složitost je vzhledem k počtu uzlů kubická.

Na [této stránce](#) naleznete applet, pomocí kterého lze snadno pochopit fungování výše zmíněného algoritmu.

Hledání nejdelší cesty v grafu

Motivační příklad:

Máme dané území, v němž chceme uspořádat cyklistický závod tak, aby byl co nejatraktivnější, tj. vedl co nejobtížnějšími úseky.

Jak převést problém nejdelší cesty na problém nejkratší cesty (tento problém totiž umíme řešit)?

- zřejmě nelze k délkám hran přičítat konstantu ani je od konstanty odečítat, ať už by se jednalo o konstanty sebevětší.

Řešení: Graf nesmí obsahovat cyklus kladné délky, pak lze **hranové ohodnocení vynásobit -1** a najít nejkratší cestu, ta pak odpovídá nejdelší cestě v původním grafu.

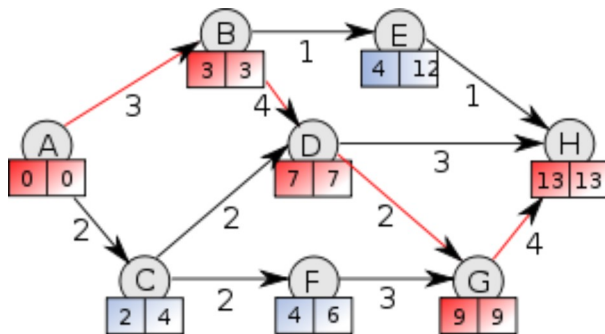
S hledáním nejdelší cesty v grafu úzce souvisí tzv. **kritická cesta**. Metoda kritické cesty (anglicky zkráceno **CPM**) je matematický algoritmus plánování průběhu množiny činností projektu. Je to jeden z důležitých nástrojů řízení projektů (analýza trvání projektu).

Jak tedy postupujeme při analýze trvání projektu?

Postup:

- Popíšeme projekt grafem:
 - činnosti odpovídají hranám (ohodnocení = doba trvání),
 - uzly odpovídají návaznostem (počátek označíme s, konec t),
- Najdeme nejdelší cestu z s do t, tj. kritickou cestu,
 - délka kritické cesty = délka celého projektu,
- prodloužení činností na kritické cestě = prodloužení celého projektu.

Příklad: červeně je vyznačena kritická cesta ABDGH.



Vedlejší efekty CPM

Analýzou grafu projektu získáme pro každou činnost 4 čísla:

- ES = earliest start time - nejdřívejší možný začátek,
- EF = earliest finish time - nejdřívejší možný konec,

- LS = latest start time - nejpozdější možný začátek,
- LF = latest finish time - nejpozdější možný konec.

Zřejmě platí, že $(EF - ES) = (LF - LS) = t$, kde t je doba trvání činnosti (ohodnocení hrany). Dále získáme ST (slack time) jako rozdíl $LS - ES = LF - EF$, nebo-li čas, o který je možné činnost prodloužit, aby neovlivnila dobu trvání projektu. Pro činnosti na kritické cestě je evidentně $ST = 0$.

Hledání nejbezpečnější cesty

Hranové ohodnocení odpovídá pravděpodobnosti bezpečného průchodu hranou, tj. pro každou hranu platí, že $w(h) \in \langle 0,1 \rangle$

Hledáme cestu s **maximálním součinem** hranových ohodnocení.

Přepočítáme hranové ohodnocení: $h' = -\log h$

- logaritmus převede součin na součet,
- obrácení znaménka převádí hledání maxima na hledání minima.

V nově ohodnoceném grafu hledáme nejkratší cestu, což už umíme pomocí výše popsaných algoritmů.

Hledání nejširší cesty

Ohodnocení hran odpovídají šířce.

Šířka cesty je minimum šířek hran na této cestě.

Algoritmus pro hledání nejširší cesty je podobný Dijkstrovu algoritmu (viz výše)

- uzly dostávají značky (p, b) ,

- p = předchůdce,
- b = šířka cesty z počátku do daného uzlu,
- značky jsou dočasné a trvalé,
- v každém kroku označíme jeden uzel trvale.

Inicializace:

- $S := \{s\}$, $\check{S} = U - \{s\}$,
- Označuj počátek $(-, \infty)$,
- Označuj následovníky počátku (s, b) , kde b je šířka příslušné hrany,
- Zbývající vrcholy označuj $(s, 0)$.

Iterace:

- Najdi uzel $j \in \check{S}$, do nějž se lze z některého uzlu $i \in S$ dostat po nejširší cestě, tj. má maximální hodnotu $\min\{b_i, b_j\}$,
- Pro všechny následovníky uzlu j bez trvalé značky přepočítáme značku, pokud jsme našli širší cestu.