Algoritmy pro hledání nejkratší cesty v grafu

Výukový materiál

1 Úvod

Tento dokument popisuje dva základní algoritmy pro hledání nejkratší cesty z jednoho zdroje (Single-Source Shortest Path) v ohodnoceném grafu $G = (V, E)$ s váhovou funkcí $w : E \to \mathbb{R}$. Předpokládáme, že $v_0 \in V$ je počáteční (zdrojový) uzel.

Nechť $p = \langle v_0, v_1, \ldots, v_k \rangle$ je sled v grafu G ze startovního uzlu v_0 do cílového uzlu v_k. **Délka sledu** p, kterou značíme jako $w(p)$, je definována jako součet vah všech hran, které tento sled tvoří:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Z této definice přímo vyplývá důležitá vlastnost: pokud k existujícímu sledu p z uzlu v_0 do uzlu v_k přidáme (napojíme) hranu (v_k, v_{k+1}), vznikne nový, prodloužený sled p'. Délka tohoto nového sledu je jednoduše součtem délky původního sledu a váhy přidané hrany:

$$w(p') = w(p) + w(v_k, v_{k+1})$$

Tento princip (tzv. vlastnost optimální podstruktury) je klíčový pro všechny algoritmy. Cílem výpočtu je najít sled p ze zdroje v_0 do uzlu v takový, že jeho délka $w(p)$ je minimální ze všech možných sledů z v_0 do v. Tuto skutečnou minimální vzdálenost označíme $\delta(v_0, v)$.

Algoritmy v průběhu výpočtu pro každý uzel $v \in V$ udržují aktuální nejlepší odhad této vzdálenosti v proměnné $d[v]$ a zároveň si pamatují předchůdce v tomto optimálním sledu v proměnné $\pi[v]$.

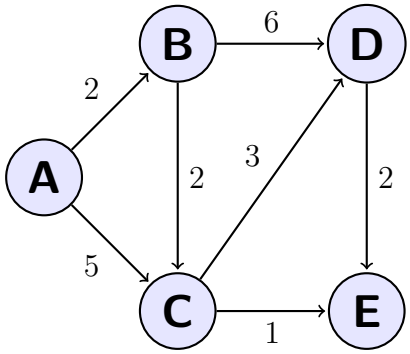
2 Vysvětlení pojmů na příkladu z praxe

Abychom lépe porozuměli abstraktním matematickým definicím z předchozí kapitoly, ukažme si vše na konkrétním příkladu ohodnoceného grafu $G = (V, E)$ s váhovou funkcí w. Můžeme si jej představit jako zjednodušenou mapu obcí, kde hrany představují silnice a jejich váhy odpovídají reálným vzdálenostem.

2.1 Základní prvky a délka sledu

Zvolme si v našem grafu uzel A a předpokládejme, že to je náš počáteční (zdrojový) uzel v_0. Nyní si představme sled p z uzlu A do uzlu C přes uzel B. Délka tohoto sledu, označovaná jako $w(p)$, je definována jako součet vah všech hran, které tento sled tvoří. Výpočet pro náš konkrétní sled by vypadal takto:

$$w(p) = w(A, B) + w(B, C) = 2 + 2 = 4$$



Obrázek 1: Příklad orientovaného ohodnoceného grafu s pěti uzly.

Na tomto sledu můžeme demonstrovat vlastnost optimální podstruktury. Pokud k existujícímu sledu p (vedoucímu z A do C) přidáme hranu (C, E), vznikne nový, prodloužený sled p'. Délka tohoto nového sledu je pak jednoduše součtem délky původního sledu a váhy nově přidané hrany:

$$w(p') = w(p) + w(C, E) = 4 + 1 = 5$$

2.2 Hledání nejkratší cesty a pracovní proměnné

Cílem výpočtu je pochopitelně najít sled ze zdroje v_0 do nějakého cílového uzlu v takový, že jeho délka $w(p)$ je absolutně minimální ze všech možných sledů z v_0 do v. Například pro cestu z A do C existuje přímá hrana o váze 5, ale my už víme, že cesta přes uzel B má délku pouze 4. Tuto skutečnou minimální vzdálenost označíme $\delta(A, C)$.

Algoritmy pro hledání nejkratších cest si v průběhu výpočtu pro každý uzel $v \in V$ udržují aktuální nejlepší odhad této vzdálenosti v proměnné $d[v]$ a zároveň si pamatují předchůdce v tomto optimálním sledu v proměnné $\pi[v]$.

Uzel v	**Skutečná vzdálenost** $\delta(v_0, v)$	**Odhad** $d[v]$ **v průběhu**	**Předchůdce** $\pi[v]$
A (v_0)	0	0	NIL
B	2	2	A
C	4	$5 \rightarrow 4$	A $\rightarrow$ B

Tabulka 1: Ukázka vývoje pracovních proměnných. Odhad $d[C]$ se v průběhu výpočtu sníží z 5 (přímá cesta) na 4 (nalezená kratší cesta přes B).

Odhad vzdálenosti $d[v]$ si můžeme představit jako hodnotu zapsanou tužkou. Na začátku zkusí algoritmus jít z A přímo do C a zapíše si odhad $d[C] = 5$. Později ale objeví cestu přes uzel B, která stojí pouze 4. Proto hodnotu 5 „vygumuje“ a aktualizuje na lepší hodnotu. Proměnná $\pi[v]$ (tedy předchůdce) zase slouží jako navigační značky. Díky nim můžeme na konci dokráčet z cíle pozpátku až do startu a odhalit tak kompletní nejkratší cestu.

3 Relaxace hrany

Srdcem všech zmíněných algoritmů pro hledání nejkratší cesty je klíčová a společná operace, kterou nazýváme **relaxace hrany** (často se překládá jako „uvolnění“). Tento krok

přímo staví na poznatcích a odvozeních, která jsme si detailně rozebrali v předchozí kapitole.

Její princip je velmi intuitivní: pro konkrétní zkoumanou hranu vedoucí z uzlu u do uzlu v zjišťujeme, zda nemůžeme vylepšit naši dosud nejlepší nalezenou cestu do cílového uzlu v tím, že zvolíme trasu „oklikou“ právě přes uzel u.

Prakticky to znamená, že vezmeme náš aktuální nejlepší odhad vzdálenosti do uzlu u (tedy hodnotu $d[u]$) a zkusíme k němu pomyslně „přidat hranu“ (u, v). Získaný nový součet, který odpovídá délce cesty přes uzel u, tedy $d[u] + w(u, v)$, následně porovnáme s naším dosavadním odhadem vzdálenosti do uzlu v (s hodnotou $d[v]$). Pokud zjistíme, že nová cesta je kratší (nový součet je ostře menší než dosavadní $d[v]$), slavíme úspěch – našli jsme kratší sled a musíme provést aktualizaci našich pracovních proměnných.

Celý tento proces můžeme formálně zapsat pomocí jednoduchého pseudokódu:

Algoritmus 1 Procedura relaxace hrany

```
1: Relax(u, v, w)      ▷ Relaxujeme hranu uv, délku sledu počítáme z váhové funkce w.
2: if d[v] > d[u] + w(u, v) then      ▷ Pokud je cesta přes uzel u kratší,
3:     d[v] ← d[u] + w(u, v)          ▷ přepíšeme délku cesty
4:     π[v] ← u                       ▷ i předchozí vrchol.
5: end if
```

3.1 Proč se operace nazývá „relaxace“?

Název „relaxace“ (neboli uvolnění) vychází z fyzikální analogie. Představte si, že odhady vzdáleností $d[v]$ představují napětí v gumičkách natahovaných přes mapu. Když najdeme kratší cestu (zkratku), přehnaně napnutá gumička povolí – její napětí se „zrelaxuje“ na nižší a přesnější hodnotu. V řeči našeho algoritmu to znamená, že se náš odhad délky nejkratší cesty zmenší.

3.2 Ukázka relaxace na našem grafu

Abychom si proces lépe představili, vezměme si náš ukázkový graf z předchozí kapitoly a ukažme si dva základní scénáře, které mohou při volání procedury **Relax** nastat.

3.2.1 Scénář 1: Úspěšná relaxace (Našli jsme zkratku)

Předpokládejme, že jsme nejprve objevili přímou cestu ze startovního uzlu A do uzlu C s váhou 5. Náš pracovní odhad je tedy $d[C] = 5$ a předchůdce $\pi[C] = A$. Následně prozkoumáme cestu z A do B, čímž zjistíme, že $d[B] = 2$. Nyní algoritmus přikročí k tomu, že chce **relaxovat hranu** (B, C) s váhou $w(B, C) = 2$.

Zkontrolujeme podmínku z našeho pseudokódu (řádek 2):

$$d[C] > d[B] + w(B, C)$$

$$5 > 2 + 2$$

$$5 > 4$$

Podmínka je splněna! Našli jsme kratší cestu přes uzel B. Provedeme proto aktualizaci hodnot (řádky 3 a 4):

- $d[C] \leftarrow 4$ (přepíšeme odhad vzdálenosti na novou, menší hodnotu)
- $\pi[C] \leftarrow B$ (přesměrujeme navigační ukazatel na nového předchůdce)

3.2.2 Scénář 2: Neúspěšná relaxace (Cesta oklikou je delší)

Zkusme nyní situaci naopak. Představte si, že po úspěšné aktualizaci máme $d[C] = 4$ a chtěli bychom zpětně relaxovat původní přímou hranu z (A, C), abychom se ujistili, jestli nám přece jen nepomůže. Váha hrany $w(A, C)$ je 5, hodnota $d[A] = 0$ (protože A je náš start).

Opět dosadíme do podmínky:

$$d[C] > d[A] + w(A, C)$$

$$4 > 0 + 5$$

$$4 > 5$$

Podmínka tentokrát neplatí. Hodnota 4 rozhodně není větší než 5. Oklika v tomto případě nedává žádný smysl, protože aktuálně již známe lepší cestu. Proměnné $d[C]$ ani $\pi[C]$ se tedy **neupraví** a zůstanou beze změny. Tímto způsobem algoritmus bezpečně ignoruje „slepé“ a neefektivní odbočky a ponechává si pouze ty nejlepší nalezené trasy.

4 Orientace grafu a záporné hrany

Při hledání nejkratších sledů hraje klíčovou roli to, zda je graf orientovaný či neorientovaný, a jaké znaménko mohou nabývat váhy jeho hran.

4.1 Nezáporné hrany

Pokud graf obsahuje pouze hrany s nezáporným ohodnocením ($w(u, v) \geq 0$ pro všechna $(u, v) \in E$), můžeme bez problémů pracovat s **orientovanými i neorientovanými grafy**. Neorientovanou hranu mezi uzly u a v s váhou w si algoritmy jednoduše představují (a v paměti často reprezentují) jako dvojici orientovaných hran (u, v) a (v, u), přičemž obě mají stejnou váhu w.

4.2 Záporné hrany u orientovaných grafů

U orientovaných grafů připouštíme i hrany se záporným ohodnocením (v praxi reprezentují např. zisk energie, peněz nebo časový bonus). Dokud v takovém grafu neexistuje tzv. **záporný cyklus** (orientovaný cyklus, jehož součet vah je menší než nula), má problém hledání nejkratšího sledu dobře definované řešení a algoritmy (jako Bellman-Ford) jej dokážou nalézt.

4.3 Problém záporných hran u neorientovaných grafů

Zcela jiná situace ovšem nastává, pokud bychom připustili záporné hrany v **neorientovaném** grafu. Proč to matematicky a algoritmicky nedává smysl?

Představme si neorientovanou hranu mezi uzly A a B se zápornou váhou, například $w = -10$. Jak jsme si řekli dříve, neorientovanou hranu lze chápat jako dvě orientované hrany: $A \to B$ za -10 a $B \to A$ za -10. Pokud vyrazíme z uzlu A do uzlu B a okamžitě se po stejné hraně vrátíme zpět do uzlu A, vznikne triviální sled $\langle A, B, A \rangle$. Délka tohoto sledu je:

$$w(A, B) + w(B, A) = (-10) + (-10) = -20$$

Tímto způsobem **každá neorientovaná záporná hrana automaticky vytváří záporný cyklus délky 2**. Pokud takový cyklus v grafu existuje, pojem „nejkratší sled“ ztrácí smysl. Kdykoliv se k této hraně dostaneme, můžeme po ní donekonečna přecházet tam a zpět, čímž bude celková délka sledu neomezeně klesat k $-\infty$. Úloha pak pro takový graf nemá konečné řešení.

5 Dijkstrův algoritmus

Dijkstrův algoritmus řeší problém nejkratší cesty pro grafy s nezáporným ohodnocením hran ($w(u,v) \geq 0$ pro všechna $(u,v) \in E$). Využívá „hladový“ přístup pomocí prioritní fronty.

Algoritmu uvedeme dvakrát. První obsahuje jen stručné komentáře zatímco ve druhém je podrobně okomentovám každý řádek.

Algoritmus 2 Dijkstrův algoritmus

Vstup: Graf $G = (V, E)$, váhy w, start v_0
Výstup: Pole vzdáleností d a předchůdců π

for každý uzel $v \in V$ **do** ▷ Inicializace proměnných.
    $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow$ NIL
end for
$d[v_0] \leftarrow 0$
$Q \leftarrow V$ ▷ Vytvoříme min-prioritní frontu řazenou podle hodnot d.
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Extract-Min(Q)
    for každý uzel $v \in$ Adj[u] **do**
        Relax(u, v, w) ▷ Pokud se $d[v]$ změní, zaktualizujeme i frontu Q.
    end for
end while

Algoritmus 3 Dijkstrův algoritmus s podrobným komentářem

Vstup: Graf $G = (V, E)$, váhy w, start v_0
Výstup: Pole vzdáleností d a předchůdců π

for každý uzel $v \in V$ **do** ▷ Pro každý vrchol inicializujeme
    $d[v] \leftarrow \infty$ ▷ vzdálenost od v_0 na nekonečno
    $\pi[v] \leftarrow$ NIL ▷ a předchozí vrchol na neznámý.
end for
$d[v_0] \leftarrow 0$ ▷ Počátečnímu vrcjolu v_0 nastavíme vzdálenost na nulu.
$Q \leftarrow V$ ▷ Vytvoříme min-prioritní frontu uzlů z množiny vrcholů grafu V. Vrcholy řadíme podle hodnot d.
while $Q \neq \emptyset$ **do** ▷ Dokud je prioritní fronta neprázdná,
    $u \leftarrow$ Extract-Min(Q) ▷ Odebereme z ní vrchol u s nejmenší hodnotou d.
    for každý uzel $v \in$ Adj[u] **do** ▷ Pro všechny sousedy v vrcholu u,
        Relax(u, v, w) ▷ zrelaxujeme hranu uv a pokud se $d[v]$ změní, zaktualizuje se i priorita vrcholu v ve frontě Q.
    end for
end while

6 Bellman-Fordův algoritmus

Bellman-Fordův algoritmus je robustnější a dokáže pracovat i s grafy obsahujícími hrany se zápornou váhou. Navíc umí detekovat existenci záporných cyklů dosažitelných ze zdroje v_0.

Algoritmus 4 Bellman-Fordův algoritmus

Vstup: Graf $G = (V, E)$, váhy w, start v_0
Výstup: TRUE pokud neexistuje záporný cyklus, jinak FALSE

```
1:  for každý uzel v ∈ V do
2:      d[v] ← ∞
3:      π[v] ← NIL
4:  end for
5:  d[v_0] ← 0
6:  for i ← 1 to |V| − 1 do                    ▷ Opakujeme (|V| − 1)-krát
7:      for každou hranu (u, v) ∈ E do
8:          Relax(u, v, w)
9:      end for
10: end for
11: for každou hranu (u, v) ∈ E do             ▷ Detekce záporného cyklu
12:     if d[v] > d[u] + w(u, v) then
13:         return FALSE
14:     end if
15: end for
16: return TRUE
```

7 Společný (generický) algoritmus

Oba algoritmy lze zapsat pomocí jediného generického schématu. Zásadní rozdíl, který určuje chování a asymptotickou složitost algoritmu, spočívá v **použité datové struktuře** $\mathcal{D}$ pro udržování množiny „aktivních“ uzlů.

Algoritmus 5 Generický algoritmus pro nejkratší cestu

Vstup: Graf $G = (V, E)$, váhy w, start v_0

```
1:  Inicializuj d a π pro všechny uzly, d[v_0] ← 0
2:  D.Insert(v_0)                          ▷ Vlož startovní uzel do datové struktury
3:  while D není prázdná do
4:      u ← D.Extract()                    ▷ Vyber uzel dle pravidel struktury
5:      for každý uzel v ∈ Adj[u] do
6:          staré_d ← d[v]
7:          Relax(u, v, w)
8:          if d[v] < staré_d and v ∉ D then
9:              D.Insert(v)                ▷ Uzel v byl zrelaxován, musíme ho prozkoumat
10:         end if
11:     end for
12: end while
```

Vliv datové struktury $\mathcal{D}$:

- **Min-Prioritní fronta (Dijkstrův algoritmus):** Pokud je $\mathcal{D}$ prioritní fronta uspořádaná vzestupně podle odhadů vzdáleností d, získáme Dijkstrův algoritmus. Za předpokladu nezáporných hran je každý uzel vložen a vybrán právě jednou.
- **Standardní FIFO fronta (Optimalizovaný Bellman-Ford / SPFA):** Pokud je $\mathcal{D}$ obyčejná fronta prvního příchozího (First-In-First-Out), vznikne algoritmus Shortest Path Faster Algorithm (SPFA). Uzel může být do fronty přidán vícekrát (pokaždé, když se jeho vzdálenost zmenší). V nejhorším případě tato varianta degraduje na klasického Bellman-Forda se stejnou časovou složitostí, ale v praxi na náhodných grafech funguje podstatně rychleji.

8 Zdůvodnění správnosti

8.1 Správnost Dijkstrova algoritmu

Správnost Dijkstrova algoritmu se opírá o předpoklad nezáporných vah hran ($w \geq 0$). Lze ji dokázat matematickou indukcí podle počtu navštívených uzlů.

- **Invariant:** V okamžiku, kdy je uzel u vybrán z prioritní fronty Q operací Extract-Min, platí, že jeho aktuální odhad $d[u]$ je roven skutečné nejkratší vzdálenosti $\delta(s, u)$.
- **Důkaz sporem:** Předpokládejme, že u je první uzel, pro který při výběru z Q platí $d[u] > \delta(s, u)$. Nechť p je skutečný nejkratší sled z v_0 do u. Na tomto sledu p musí existovat uzel y, který je v Q (ještě nebyl trvale vyřízen), a jeho předchůdce x na sledu p již byl vyřízen. Protože x bylo vyřízeno dříve, cesta k y přes x již byla zrelaxována, a tedy $d[y] = \delta(s, y)$. Protože hrany jsou nezáporné, délka podsledu z y do u je ≥ 0. Z toho plyne, že $\delta(s, y) \leq \delta(s, u)$. Dostáváme:

  $$d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$$

  To by ale znamenalo, že $d[y] < d[u]$. Dijkstrův algoritmus by z prioritní fronty vybral uzel y **před** uzlem u, což je spor s naším předpokladem, že byl vybrán uzel u. Invariant tedy platí a algoritmus je korektní.

8.2 Správnost Bellman-Fordova algoritmu

Správnost Bellman-Fordova algoritmu vychází z vlastnosti nejkratších sledů. Nejkratší sled v grafu bez záporných cyklů nesmí obsahovat žádný cyklus, a proto je to jednoduchá cesta obsahující maximálně $|V| - 1$ hran.

- **Důkaz indukcí:** Dokazujeme, že po i-té iteraci vnějšího cyklu platí pro všechny uzly v, ke kterým existuje nejkratší sled obsahující nejvýše i hran, že $d[v] = \delta(s, v)$.
- **Báze ($i = 0$):** Před prvním průchodem platí tvrzení pouze pro zdroj v_0, kde $d[v_0] = 0 = \delta(v_0, v_0)$ a sled má délku 0 hran.
- **Krok:** Předpokládejme, že to platí pro sledy délky i. Nechť nejkratší sled do uzlu v má $i + 1$ hran a končí hranou (u, v). Z principu optimality podproblémů musí být podsled do uzlu u nejkratším možným sledem délky i hran. Podle indukčního

předpokladu tedy po i-té iteraci platí $d[u] = \delta(v_0, u)$. V $(i+1)$-té iteraci algoritmus projde všechny hrany, a tedy i hranu (u, v). Provede se relaxace, čímž $d[v]$ nabyde hodnoty $d[u] + w(u, v) = \delta(v_0, v)$.

- **Závěr:** Jelikož žádný nejkratší sled bez záporných cyklů nemůže mít více než $|V|-1$ hran, po provedení $|V| - 1$ iterací algoritmus zaručeně nalezne skutečné minimální vzdálenosti do všech uzlů. Závěrečná kontrolní iterace pak odhalí případnou existenci záporného cyklu, pokud lze nějakou vzdálenost stále zkrátit.

9 Rozbor časové složitosti

9.1 Složitost Dijkstrova algoritmu

Časová složitost Dijkstrova algoritmu závisí téměř výhradně na implementaci min-prioritní fronty Q. Každý z $|V|$ uzlů je do fronty vložen a vybrán operací Extract-Min právě jednou. Pro každou z $|E|$ hran grafu se může zavolat operace Decrease-Key (aktualizace priority).

- **Lineární pole:** Extract-Min trvá $\mathcal{O}(|V|)$, Decrease-Key $\mathcal{O}(1)$. Celkově $\mathcal{O}(|V|^2 + |E|) = \mathcal{O}(|V|^2)$. Tato implementace je vhodná pro velmi husté grafy ($|E| \approx |V|^2$).
- **Binární halda:** Extract-Min trvá $\mathcal{O}(\log |V|)$, Decrease-Key $\mathcal{O}(\log |V|)$. Celkově $\mathcal{O}((|V| + |E|) \log |V|)$. Standardní a nejpoužívanější implementace, ideální pro řídké grafy.
- **Fibonacciho halda:** Extract-Min trvá amortizovaně $\mathcal{O}(\log |V|)$, Decrease-Key amortizovaně $\mathcal{O}(1)$. Celkově $\mathcal{O}(|V| \log |V| + |E|)$. Teoreticky nejrychlejší známý přístup pro hustší grafy.

9.2 Složitost Bellman-Fordova algoritmu

Základní složitost Bellman-Fordova algoritmu je velmi přímočará k analýze, neboť se skládá ze statických vnořených cyklů.

- **Inicializace:** Nastavení počátečních hodnot trvá $\mathcal{O}(|V|)$.
- **Hlavní cyklus:** Vnější cyklus proběhne přesně $(|V| - 1)$-krát. Uvnitř se prochází přesně všech $|E|$ hran grafu a pro každou se provede relaxace v čase $\mathcal{O}(1)$. Tento krok zabere $\mathcal{O}(|V| \cdot |E|)$.
- **Detekce cyklů:** Závěrečný průchod všech hran trvá $\mathcal{O}(|E|)$.

Celková časová složitost Bellman-Fordova algoritmu je tedy nekompromisně $\mathcal{O}(|V| \cdot |E|)$. V nejhorším případě pro úplný graf ($|E| \approx |V|^2$) dosahuje kubické složitosti $\mathcal{O}(|V|^3)$. To je daň za jeho schopnost pracovat se zápornými hranami.