1 Nejkratší cesty

- Algoritmus Bellmana a Forda
- Acyklické grafy
- Dijkstrův algoritmus
- Lineární nerovnice

Cesta v grafu

cesta v grafu $G = (V, E)$ je posloupnost vrcholů $p = \langle v_0, v_1, \ldots, v_k \rangle$ taková, že $(v_{i-1}, v_i) \in E$ pro $i = 1, \ldots, k$

jednoduchá cesta je cesta, která neobsahuje dva stejné vrcholy

terminologie

cesta	jednoduchá cesta
path	simple path
walk	path
sled	cesta

$p = \langle v_0, v_1, \ldots, v_k \rangle$ je cestou z vrcholu u do vrcholu v právě když $v_0 = u$ a $v_k = v$

v je dosažitelný z u, $u \rightsquigarrow v$

Délka cesty

graf $G = (V, E)$, váhová funkce (*ohodnocení, délka hran*) $w : E \to \mathbb{R}$

délka cesty $p = \langle v_0, v_1, \ldots, v_k \rangle$ je součet délek hran cesty

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

délka nejkratší cesty z vrcholu u do vrcholu v je definovaná předpisem

$$\delta(u, v) \stackrel{\text{def}}{=} \begin{cases} \min\{w(p) \mid u \overset{p}{\rightsquigarrow} v\} & \text{existuje cesta z } u \text{ do } v \\ \infty & \text{jinak} \end{cases}$$

nejkratší cesta z vrcholu u do vrcholu v je libovolná cesta p z u do v pro kterou $w(p) = \delta(u, v)$

pro neohodnocené grafy se délka cesty definuje jako počet hran cesty

Varianty problému nejkratší cesty

nejkratší cesty z daného vrcholu do všech vrcholů

single source shortest path, **SSSP**

tato přednáška

nejkratší cesty ze všech vrcholů do daného vrcholu

pro neorientované grafy totožné s SSSP

pro orientované grafy redukce na SSSP změnou orientace hran

nejkratší cesta mezi danou dvojicí vrcholů speciální případ SSSP, nejsou známy asymptoticky rychlejší algoritmy než pro SSSP

tato přednáška

nejkratší cesty mezi všemi dvojicemi vrcholů

řešení opakovanou aplikací algoritmu pro SSSP

existují efektivnější algoritmy

ADS II

nejdelší, nejširší, nejspolehlivější . . . cesty

viz literatura

Algoritmy pro SSSP

orientované grafy

neohodnocený graf

průzkum do šířky, BFS

acyklický graf

průzkum do hloubky

tato přednáška

graf s nezáporným ohodnocením hran

Dijkstrův algoritmus a jiné

tato přednáška

obecný graf

algoritmus Bellmana Forda a jiné

tato přednáška

Algoritmy pro SSSP

neorientované grafy

neohodnocený graf

- průzkum do šířky, BFS

graf s nezáporným ohodnocením hran

- hranu nahradíme dvojicí orientovaný hran a převedeme na úlohu v orientovaném grafu

obecný graf

- nahrazením hrany se záporným ohodnocením dvojicí orientovaných hran vznikne cyklus záporné délky
- pokud původní graf obsahuje hrany záporné délky, ale žádný cyklus záporné délky, lze takovou úlohu převést na hledání nejlevnějšího perfektního párování
- když obsahuje cyklus záporné délky, problém je NP-těžký a umíme ho řešit pouze algoritmy exponenciální složitosti
- viz literatura

Nejkratší cesta vs nejkratší jednoduchá cesta

graf neobsahuje hrany záporné délky

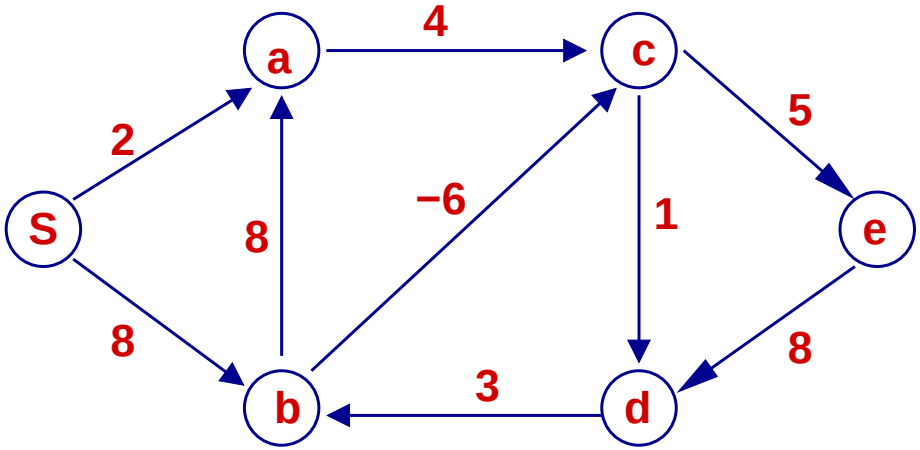
mezi každou dvojicí vrcholů existuje nejkratší cesta, která je jednoduchá

nechť p je nejkratší cesta, která není jednoduchá, tj. obsahuje cyklus

- cyklus nemůže mít zápornou délku (*spor s předpokladem neexistence hran záporné délky*
- cyklus nemůže mít kladnou délku (*spor s předpokladem, že cesta je nejkratší*
- cyklus má nulovou délku - cyklus můžeme z cesty vypustit a dostaneme jednoduchou nejkratší cestu

Nejkratší cesta vs nejkratší jednoduchá cesta

graf obsahuje hrany záporné délky



- cyklus $\langle b, c, d, b \rangle$ má délku -2
- jestliže nějaká cesta z x do y obsahuje cyklus záporné délky, tak žádná cesta z x do y nemůže být nejkratší cestou, $\delta(x, y) = -\infty$

v případě, že graf obsahuje hrany se zápornou délkou, problém nejkratší cesty je formulovaný jako

1. rozhodni, zda graf obsahuje cyklus záporné délky
2. když ne, tak najdi nejkratší (jednoduché) cesty

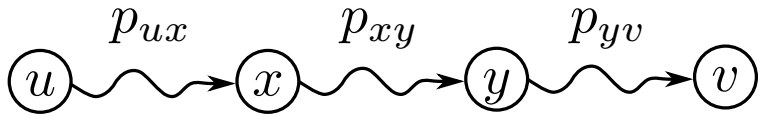
Struktura nejkratších cest

Lemma 1

Každá podcesta nejkratší cesty je nejkratší cestou.

- nechť p je nejkratší cesta z u do v

  $w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$



- předpokládejme, že existuje kratší cesta z x do y, $w(p'_{xy}) < w(p_{xy})$
- zkonstruujeme novou cestu $p' = u \overset{p_{ux}}{\rightsquigarrow} x \overset{p'_{xy}}{\rightsquigarrow} y \overset{p_{yv}}{\rightsquigarrow} v$

$$\begin{aligned} w(p') &= w(p_{ux}) + w(p'_{xy}) + w(p_{yv}) \\ &< w(p_{ux}) + w(p_{xy}) + w(p_{yv}) \\ &= w(p) \end{aligned}$$

  což je spor s předpokladem, že p je nejkratší cesta z u do v

Reprezentace nejkratších cest

strom nejkratších cest

pozor na rozdíl mezi stromem nejkratších cest a nejlevnější kostrou grafu

Reprezentace stromu nejkratších cest z vrcholu s

atribut vzdálenost *distance*, $v.d$

- iniciální nastavení $v.d = \infty$
- hodnota $v.d$ se v průběhu výpočtu snižuje
- hodnota $v.d$ je horním odhadem délky nejkratší cesty, $v.d \geq \delta(s, v)$
- na konci výpočtu je $v.d = \delta(s, v)$

atribut předchůdce *parent*, $v.\pi$

- iniciální nastavení $v.\pi = Nil$
- vrchol $v.\pi$ je předchůdcem vrcholu v na cestě z s do v délky $v.d$
- na konci výpočtu je $v.\pi$ předchůdce vrcholu v na nejkratší cestě z s do v, resp. $v.\pi = Nil$ když neexistuje cesta z s do v

graf předchůdců $G_p = (V_p, E_p)$ je definovaný hodnotami $.\pi$

$$V_p = \{v \in V \mid v.\pi \neq Nil\} \cup \{s\}$$

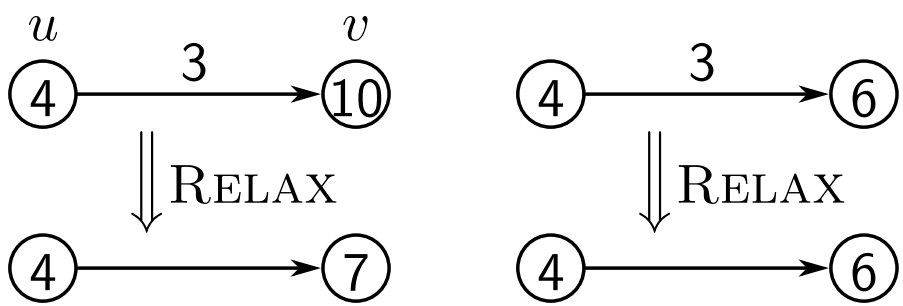
$$E_p = \{(v.\pi, v) \mid v \in V_p \setminus \{s\}\}$$

strom nejkratších cest na konci výpočtu je G_p stromem nejkratších cest

- s je kořen stromu, V_p je množina vrcholů dosažitelných z vrcholu s
- pro každý vrchol $v \in V_p$, (jediná) cesta z s do v v G_p je nejkratší cestou z s do v v G

Relaxace

- technika, kterou využívají algoritmy pro hledání nejkratších cest
- relaxací hrany (u, v) rozumíme test, zda je možné zkonstruovat kratší cestu do v tak, že projedeme přes vrchol u
- když aktuálně známá cesta do vrcholu u prodloužená o hranu (u, v) je kratší než aktuálně známá cesta do v ($u.d + w(u, v) < v.d$), tak jsme našli novou - kratší cestu do v a podle toho aktualizujeme hodnoty $v.d$ a $v.\pi$ ($v.d \leftarrow u.d + w(u, v)$ a $v.\pi \leftarrow u$)



- hranu (u, v) nazýváme napjatou právě když $u.d + w(u, v) < v.d$

Generický SSSP algoritmus

Init_SSSP(G, s)

```
1 foreach v ∈ V do v.d ← ∞; v.π ← Nil od
2 s.d ← 0
```

Relax(u, v, w)

```
1 v.d ← u.d + w(u, v)
2 v.π ← u
```

Generic_SSSP(G, w, s)

```
1 INIT_SSSP(G, s)
2 S ← s
3 while S ≠ ∅ do
4         vyber u z S
5         foreach  hranu (u, v) ∈ E do
6           if v.d > u.d + w(u, v)
7             then RELAX(u, v, w)
8                   S ← S ∪ {v} fi od
9 od
```

Korektnost generického SSSP algoritmu

- pro každý vrchol v platí, že hodnota $v.d$ je buď ∞, anebo je rovna délce nějaké cesty z s do v
  důkaz indukcí na počet relaxací
- když graf neobsahuje cyklus záporné délky, tak hodnota $v.d$ je buď ∞, anebo je rovna délce nějaké jednoduché cesty z s do v
  důkaz viz diskuse nejkratší cesta vs nejkratší jednoduchá cesta

  důsledek: **generický algoritmus skončí, protože v grafu existuje jenom konečný počet jednoduchých cest**

- když žádná hrana grafu není napjatá, tak hodnota $v.d$ je rovna délce cesty $s \to \cdots \to v.\pi.\pi \to v.\pi \to v$
- když žádná hrana grafu není napjatá, tak cesta $s \to \cdots \to v.\pi.\pi \to v.\pi \to v$ je nejkratší cestou z s do v
  důkaz indukcí na počet relaxací

  důsledek: **když výpočet generického algoritmu skončí, tak G_p je strom nejkratších cest**

Složitost generického SSSP algoritmu

závisí od toho

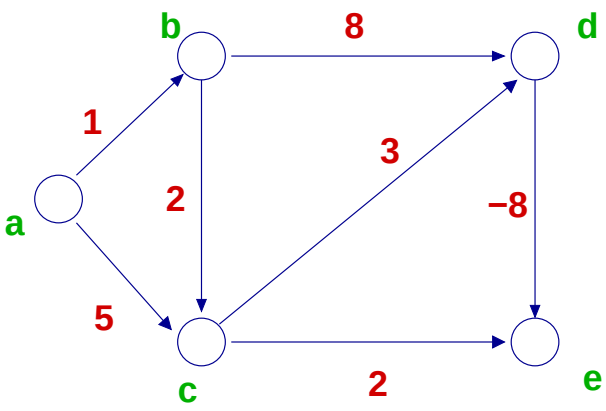
- jakou datovou strukturu použijeme pro reprezentaci množiny S obsahující vrcholy určené k prozkoumání
  (tj. vrcholy, u kterých byla změněna hodnota $.d$)
- v jakém pořadí budeme prozkoumávat vrcholy z množiny S

Bellmanův - Fordův algoritmus

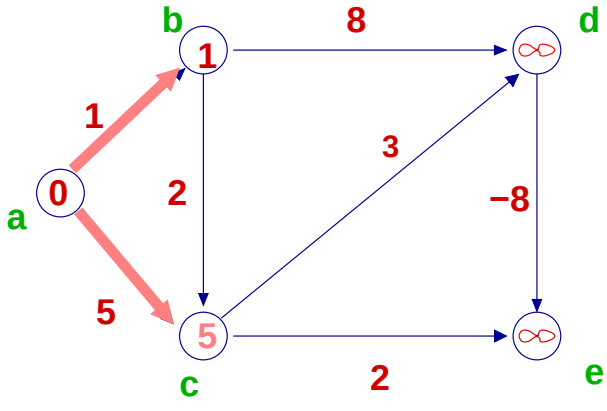
- pro reprezentaci množiny vrcholů určených k prozkoumání využívá **frontu** (FIFO)
- pro přehlednější analýzu složitosti a důkaz korektnosti rozdělujeme výpočet do iterací; v jedné iteraci se relaxují všechny hrany grafu

Bellman-Ford(G, w, s)

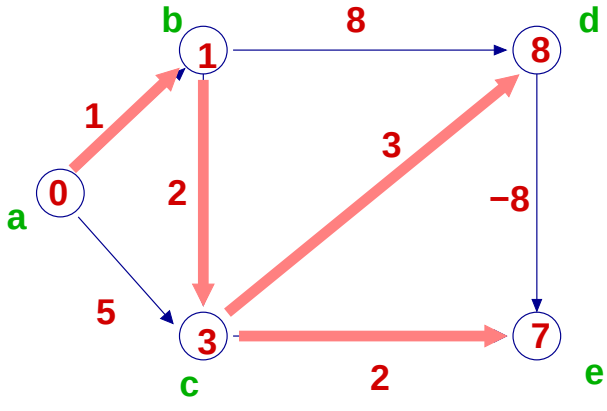
```
1  INIT_SSSP(G, s)
2  for i = 1 to |V| − 1 do
3      foreach (u, v) ∈ E do
4          if v.d > u.d + w(u, v) then RELAX(u, v, w) fi
5      od
6  od
7  foreach (u, v) ∈ E do
8      if v.d > u.d + w(u, v) then return FALSE fi
9  od
10 return TRUE
```



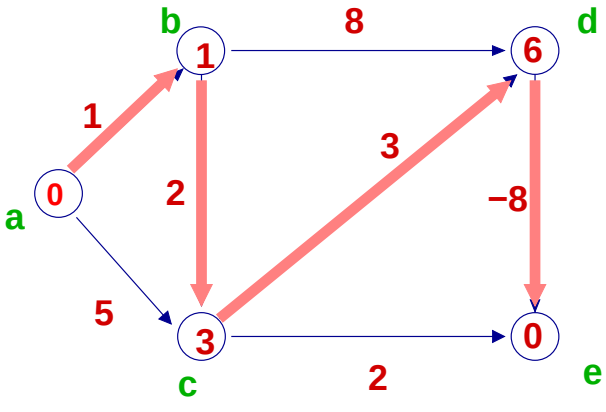
(a)



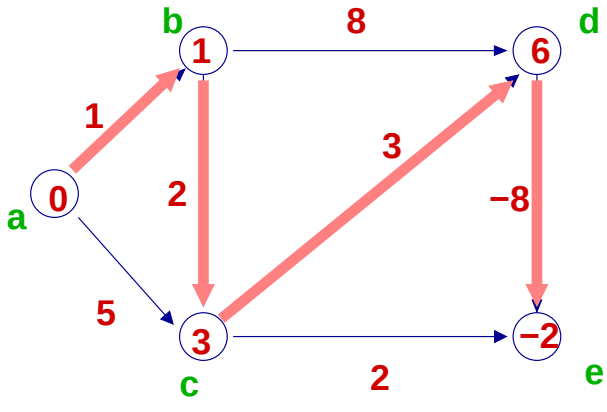
(b)



(c)



(d)



(e)

graf G_p

výpočet algoritmu BELLMAN-FORD, hledáme nejkratší cesty z vrcholu a

v každé iteraci cyklu 2 - 6 relaxujeme hrany v pořadí (c, e), (d, e), (b, d), (c, d), (b, c), (a, c), (a, b)
barevně jsou vyznačeny hrany grafu předchůdců G_p

Korektnost 1

Lema 2

Když graf G neobsahuje cyklus záporné délky dosažitelný z s, tak po $|V| - 1$ iteracích cyklu 3 - 5 pro všechny vrcholy v platí

$$v.d = \delta(s, v).$$

- nechť v je dosažitelný z s a nechť $p = \langle v_0, v_1, \ldots, v_k \rangle$ je nejkratší jednoduchá cesta z s do v ($v_0 = s$, $v_k = v$)
- protože cesta p neobsahuje cyklus, má nanejvýš $|V| - 1$ hran, tj. $k \leq |V| - 1$
- v každé iteraci cyklu se relaxují všechny hrany grafu, speciálně
  - v první iterace se relaxuje hrana (v_0, v_1)
  - v druhé iteraci se relaxuje hrana (v_1, v_2)
  - ...
  - v k-té iteraci se relaxuje hrana (v_{k-1}, v_k)
- indukcí ověříme, že po i-té iteraci platí $v_i.d = \delta(s, v_i)$

Korektnost 2

Lema 3

Když graf G neobsahuje cyklus záporné délky dosažitelný z s, tak algoritmus vrátí hodnotu TRUE.

- po ukončení výpočtu platí pro každou hranu $(u, v) \in E$

$$\begin{aligned} v.d &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \\ &= u.d + w(u, v) \end{aligned}$$

- žádná hrana není napjatá a test na řádku 8 nevrátí hodnotu FALSE

Korektnost 3

Lema 4

Když graf G obsahuje cyklus záporné délky dosažitelný z s, tak algoritmus vrátí hodnotu FALSE.

- nechť $c = \langle v_0, v_1, \ldots, v_k \rangle$, $v_0 = v_k$ je cyklus záporné délky dosažitelný z s, $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$
- předpokládejme, že algoritmus vrátí hodnotu TRUE, tj. pro každou hranu cyklu platí $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$
- sumací přes všechny vrcholy cyklu

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i)) = \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- každý vrchol se vyskytuje právě jednou v součtech $\sum_{i=1}^{k} v_{i-1}.d$ a $\sum_{i=1}^{k} v_i.d$
- $$\sum_{i=1}^{k} v_{i-1}.d = \sum_{i=1}^{k} v_i.d \quad \Longrightarrow \quad 0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$$
- spor s předpokladem o délce cyklu c

Složitost

složitost algoritmu Bellmana a Forda

- inicializace má složitost $\Theta(V)$
- relaxace hrany má konstantní složitost
- cyklus 3 - 5 má složitost $\Theta(E)$; počet jeho opakování je $V - 1$
- celková složitost je $\mathcal{O}(VE)$
  jiný zápis: $\mathcal{O}(mn)$, kde n je počet vrcholů a m je počet hran grafu

existuje efektivnější řešení?

ne lehce najdeme graf a takové pořadí relaxace jeho hran, pro které je nutných $V - 1$ iterací

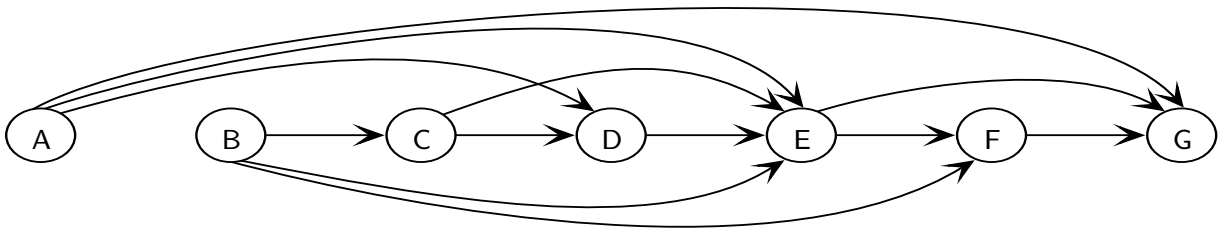
??? otázka vhodného pořadí relaxace hran

ano vhodné pořadí hran dokážeme určit pro speciální typy grafů

- acyklické grafy
- grafy bez záporných hran

Nejkratší cesty v orientovaném acyklickém grafu

- optimálně pořadí relaxace hran v Bellmanově - Fordově algoritmu je takové, že vždy relaxujeme hranu (u, v) pro kterou $u.d = \delta(s, u)$
- pro obecný graf určit pořadí relaxací tak, aby byla dodržena uvedená podmínka, může být stejně náročné jako vypočíst nejkratší cesty
- speciálně pro acyklické grafy se toto pořadí dá vypočíst jednoduše: požadovanou vlastnost má topologické uspořádání vrcholů grafu



A < B < C < D < E < F < G

Nejkratší cesty v orientovaném acyklickém grafu

Dag(G, w, s)

```
1 najdi topologické uspořádání vrcholů grafu G
2 INIT_SSSP(G, s)
3 foreach vrchol u v topologickém uspořádání do
4   foreach (u, v) ∈ E do
5     if v.d > u.d + w(u, v) then RELAX(u, v, w) fi
6   od
7 od
```

- časová složitost $\Theta(V + E)$
- topologické uspořádání garantuje, že hrany *každé* cesty jsou relaxované v pořadí, v jakém se vyskytují na cestě

Dijkstrův algoritmus

- pro reprezentaci množiny vrcholů určených k prozkoumání využívá **prioritní frontu**, kde priorita vrcholu v je určena hodnotou $v.d$
- *Dijkstrův algoritmus můžeme nahlížet i jako efektivní implementaci prohledávání grafu do šířky, na rozdíl od BFS neukládáme vrcholy, které mají být prozkoumané, do fronty, ale do prioritní fronty*

- řeší problém SSSP pro grafy **s nezáporným ohodnocením hran**

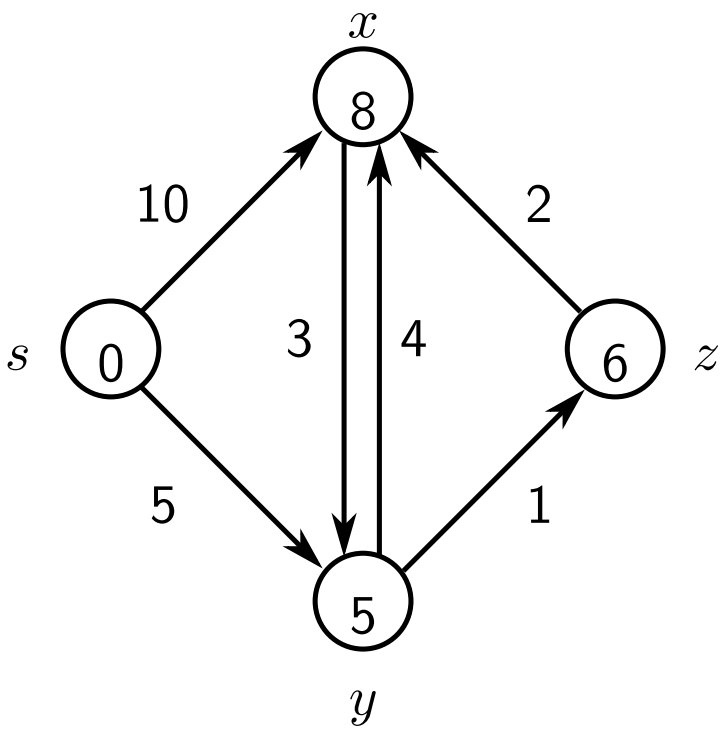
Dijkstrův algoritmus

Dijkstra(G, w, s)

```
1  INIT_SSSP(G, s)
2  S ← ∅
3  Q ← V
4  while Q ≠ ∅ do
5        u ← EXTRACT_MIN(Q)
6        S ← S ∪ {u}
7        foreach (u, v) ∈ E do
8           if v.d > u.d + w(u, v) then RELAX(u, v, w) fi
9        od
10 od
```

- algoritmus udržuje dvě množiny
  - S vrcholy, pro které je již vypočtena délka nejkratší cesty
  - Q prioritní fronta, $Q = V \setminus S$
- algoritmus vybírá vrchol $u \in Q$ s nejmenší hodnotou $u.d$ a relaxuje hrany vycházející z u

Dijkstrův algoritmus - příklad



vrcholy se přidávají do množiny S v pořadí s, y, z, x

Korektnost

Lema 5 (Invariant cyklu while)

Na začátku iterace **while** *cyklu platí* $v.d = \delta(s, v)$ *pro všechny vrcholy* $v \in S$.

Inicializace na začátku $S = \emptyset$ a tvrzení platí triviálně

Ukončení na konci $Q = \emptyset$, tj. $S = V$ a $v.d = \delta(s, v)$ pro všechny $v \in V$

Iterace potřebujeme prokázat, že když u přesuneme do S, tak $u.d = \delta(u.s)$

- když u není dosažitelný z s tak tvrzení platí protože $u.d = \delta(s, u) = \infty$
- v opačném případě nechť p je nejkratší cesta z s do u; cestu p můžeme dekomponovat na dvě cesty $s \overset{p_1}{\rightsquigarrow} x \to y \overset{p_2}{\rightsquigarrow} u$ tak, že bezprostředně před zařazením u do S všechny vrcholy cesty p_1 patří do S a $y \notin S$
- $x \in S \implies x.d = \delta(s, x)$
- při zařazení x do S byla relaxovaná hrana (x, y)) a $s \overset{p_1}{\rightsquigarrow} x \to y$ je nejkratší cesta do $y \implies y.d = \delta(s, y)$
- hrany mají nezápornou délku $\implies \delta(s, y) \leq \delta(s, u)$
- v dané iteraci jsme vybrali vrchol $u \implies u.d \leq y.d$
- spojením dostáváme $u.d \leq y.d = \delta(s, y) \leq \delta(s, u) \implies u.d \leq \delta(s, u)$

Složitost Dijkstrova algoritmu

$\Theta(V)$ operací INSERT (do fronty přidá nový objekt)
$\Theta(V)$ operací EXTRACT_MIN (z fronty odstraní objekt s minimálním klíčem)
$\Theta(E)$ operací DECREASE_KEY (objektu ve frontě sníží hodnotu klíče)

složitost algoritmu závisí od způsobu implementace prioritní fronty Q

pole složitost $\Theta(V \cdot V + E \cdot 1) = \Theta(V^2)$
- INSERT má složitost $\Theta(1)$
- EXTRACT_MIN má složitost $\Theta(V)$
- DECREASE_KEY má složitost $\Theta(1)$

binární halda složitost $\Theta(V \log V + E \log V)$
- INSERT má složitost $\Theta(\log V)$
- EXTRACT_MIN má složitost $\Theta(\log V)$
- DECREASE_KEY má složitost $\Theta(\log V)$

Fibonacciho halda složitost $\Theta(V \log V + E)$
- INSERT má složitost $\Theta(1)$
- EXTRACT_MIN má složitost $\Theta(\log V)$
- DECREASE_KEY má složitost $\Theta(1)$

Optimalizace Dijkstrova algoritmu pro hledání nejkratší cesty mezi dvěma vrcholy s a t

optimalizace 1

- výpočet ukončíme když vrchol t odebereme z prioritní fronty

Optimalizace Dijkstrova algoritmu pro hledání nejkratší cesty mezi dvěma vrcholy s a t

optimalizace 2 - dvousměrné hledání (*bidirectional search*)

- současně spouštíme (dopředný) výpočet Dijkstrova algoritmu z vrcholu s a (zpětný) výpočet z vrcholu t, vždy jednu iteraci každého výpočtu
- dopředný výpočet používá frontu Q_f a přiřazuje vrcholům hodnoty $.d_f$ a $.\pi_f$, zpětný frontu Q_b a přiřazuje hodnoty $.d_b$ a $.\pi_b$
- výpočet ukončíme když nějaký vrchol w je odstraněn z obou front Q_f a Q_b
- po ukončení najdeme vrchol x s minimální hodnotou $x.d_f + x.d_b$ (*pozor, nemusí to být vrchol* w)
- využitím atributů $.\pi_f$ a $.\pi_b$ najdeme nejkratší cestu z s do x a nejkratší cestu z x do t; jejich spojením dostaneme nejkratší cestu z s do t

Optimalizace Dijkstrova algoritmu pro hledání nejkratší cesty mezi dvěma vrcholy s a t

optimalizace 3 - heuristika A*

- pokud bychom dovedli spolehlivě zjistit, že nejkratší cesta z s do t nepovede přes vrchol v, mohli bychom zpracování vrcholu v a hran s ním incidentních přeskočit $\Longrightarrow$ pracovali bychom s menší grafem, a tedy rychleji
- jestliže dva vrcholy jsou stejně daleko od s, chceme při průzkumu preferovat ten, který je blíže k cílovému vrcholu t
- pro odhad preferencí používáme ohodnocení vrcholů – potenciál $h : V \to \mathbb{R}$
- Dijkstrův algoritmus s heuristikou se od klasického liší v tom, že při výběru vrcholu z prioritní fronty vybíráme vrchol s nejnižší hodnotou $v.d + h(v)$
- *jak volit potenciál a proč může urychlit výpočet?*

Heuristika A* - přípustný potenciál

Potenciál je přípustný právě když pro každou hranu $(u, v) \in E$ splňuje podmínku

$$h(u) \leq w(u, v) + h(v)$$

a pro vrchol t platí $h(t) = 0$.

- pro libovolnou cestu $p = \langle v_0 = u, v_1, \ldots, v_k = t \rangle$ z u do t a přípustný potenciál platí

$$\begin{aligned} w(p) &= \sum_{i=0}^{k-1} w(v_i, v_{i+1}) \\ &\geq h(v_0) - h(v_1) + h(v_1) - h(v_2) + \ldots + h(v_{k-1}) - h(v_k) \\ &= h(u) - h(t) = h(u) \end{aligned}$$

t.j. potenciál vrcholu u je dolním odhadem délky cesty z u do t

Heuristika A* - úprava ohodnocení grafu pomocí potenciálu

- vytvoríme nové ohodnocení grafu $w' : E \to \mathbb{R}$, které definujeme jako

$$w'(u, v) = w(u, v) - h(u) + h(v)$$

- když h je přípustný potenciál, tak pro nové ohodnocení grafu a pro každou hranu grafu platí

$$w'(u, v) \geq 0$$

  t.j. pro výpočet nejkratších cest můžeme použít Dijkstrův algoritmus

- nové ohodnocení w' nemění nejkratší cesty, protože pro každou cestu p z s do t platí

$$w'(p) = w(p) + h(t) - h(s)$$

  a potenciálový rozdíl medzi s a t je pro všechny cesty z s do t stejný

Heuristika A* - úprava ohodnocení grafu pomocí potenciálu

- aby hodnoty h měly příznivý vliv na rychlost výpočtu, měli by hodnoty $h(v)$ být dolním odhadem vzdálenosti z vrcholu v do cílového vrcholu t; čím je odhad přesnější, tím je výpočet rychlejší
- Dijkstrův algoritmus s heuristikou se od klasického liší v tom, že při výběru vrcholu z prioritní fronty vybíráme vrchol s nejnižší hodnotou $v.d + h(v)$
- za předpokladu, že ohodnocení vrcholů h splňuje pro všechny hrany (x, y) grafu podmínku $w(x, y) \geq h(x) - h(y)$, Dijkstrův algoritmus s heuristikou je korektní
  důkaz probíhá analogicky jako pro Dijkstrův algoritmus

Dijkstrův algoritmus a obecné grafy

graf se záporně ohodnocenými hranami ale bez cyklů záporné délky

- algoritmus najde korektní řešení
  (po každé změně hodnoty $u.d$ vrátíme vrchol u do fronty)
- složitost výpočtu může být až exponenciální vůči velikosti grafu
- v praxi někdy rychlejší než algoritmus Bellmana a Forda

graf s cykly záporné délky

- výpočet algoritmu není konečný

Úloha lineárního programování

pro danou $m \times n$ matici A a vektory $b = (b_1, \ldots, b_m)$ a $c = (c_1, \ldots, c_n)$ najít vektor $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, který optimalizuje hodnotu účelové funkce $\sum_{i=1}^{n} c_i x_i$ při splnění ohraničení $Ax \leq b$

příklad

minimalizovat $-2x_1 - 3x_2 - x_3$

při splnění ohraničení

$$\begin{aligned} -x_1 - x_2 - x_3 &\leq 3 \\ 3x_1 + 4x_2 - 2x_3 &\leq 10 \\ 2x_1 - 4x_2 &\leq 2 \\ 4x_1 - x_2 + x_3 &\leq 0 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

přípustné řešení (0, 0, 0)

optimální řešení (0, 5, 5)

Úloha lineárního programování

význam

- mnoho problémů dokážeme vyjádřit jako úlohu lineárního programování (*např. problém nejkratší cesty*)
- pro řešení těchto problémů potom stačí použít algoritmus pro řešení úlohy lineárního programování [1]

algoritmická složitost

- existují polynomiální algoritmy pro řešení úlohy lineárního programování
- pro řešení speciálních případů úlohy lineárního programování existují mnohem rychlejší algoritmy
- problém lineárních ohraničení je příkladem takovéto speciální úlohy
- ukážeme postup založený na SSSP

[1] viz kurz IA101 Algoritmika pro těžké problémy

Problém lineárních nerovnic

- je daná množina nerovnic tvaru $x_i - x_j \leq b_k$, kde

  x jsou proměnné, $1 \leq, i, j \leq n$

  b jsou konstanty, $1 \leq k \leq m$

- úkolem je najít takové hodnoty proměnných x, které splňují všechny nerovnice (tzv. přípustné řešení; v případě, že neexistuje žádné přípustné řešení, tak výstupem je $\textsc{False}$

příklad

$$\begin{aligned} x_1 - x_2 &\leq 5 \\ x_1 - x_3 &\leq 6 \\ x_2 - x_4 &\leq -1 \\ x_3 - x_4 &\leq -2 \\ x_4 - x_1 &\leq -3 \end{aligned}$$

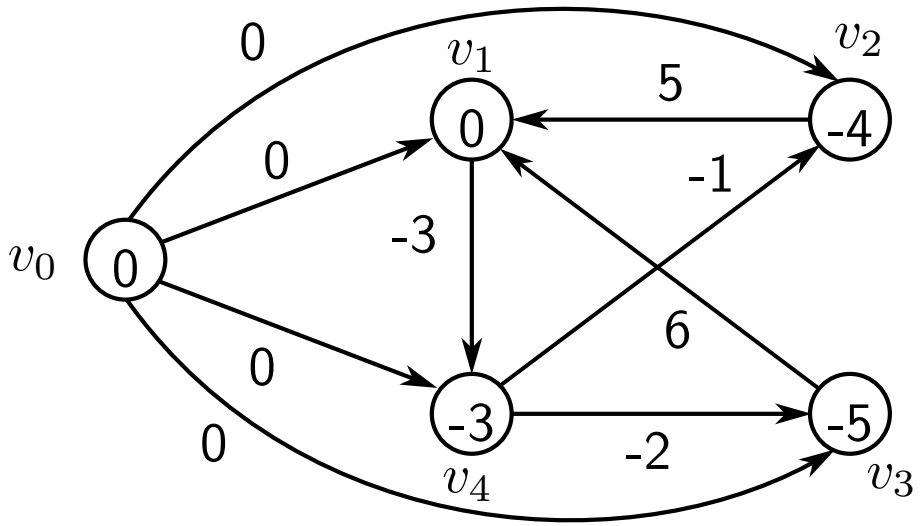
přípustné řešení $x = (0, -4, -5, -3)$

Graf lineárních nerovnic

ohodnocený, orientovaný graf $G = (V, E)$

- $V = \{v_0, v_1, \ldots, v_n\}$
  (*jeden vrchol pro každou proměnnou plus vrchol* v_0)
- $E = \{(v_i, v_j) \mid x_j - x_i \leq b_k$ je nerovnica $\} \cup$
  $\cup \{(v_0, v_1), (v_0, v_2), \ldots, (v_0, v_n)\}$
- $w(v_0, v_j) = 0$ pro všechny j
- $w(v_i, v_j) = b_k$ když $x_j - x_i \leq b_k$

$$
\begin{aligned}
x_1 - x_2 &\leq 5 \\
x_1 - x_3 &\leq 6 \\
x_2 - x_4 &\leq -1 \\
x_3 - x_4 &\leq -2 \\
x_4 - x_1 &\leq -3
\end{aligned}
$$



Nejkratší cesty v grafu lineárních nerovnic

Věta 6

Pro daný systém lineárních nerovnic a k němu odpovídající graf lineárních nerovnic $G = (V, E)$ platí:

1. *když G nemá cyklus záporné délky, tak*

$$x = (\delta(v_0, v_1), \delta(v_0, v_2), \ldots, \delta(v_0, v_n))$$

   je přípustným řešením systému nerovnic;

2. *když G má cyklus záporné délky, tak systém nemá žádné přípustné řešení.*

1. z neexistence cyklu záporné délky plyne

$$\delta(v_0, v_j) \quad \leq \quad \delta(v_0, v_i) + w(v_i, v_j)$$

   po dosazení

$$\begin{aligned} x_j &\leq x_i + b_k \\ x_j - x_i &\leq b_k \end{aligned}$$

Nejkratší cesty v grafu lineárních nerovnic

Věta 7

Pro daný systém lineárních nerovnic a k němu odpovídající graf lineárních nerovnic $G = (V, E)$ platí:

1.
2. *když G má cyklus záporné délky, tak systém nemá žádné přípustné řešení.*

1.
2. nechť $c = \langle v_1, v_2, \ldots, v_k \rangle$, kde $v_1 = v_k$, je cyklus záporné délky
   hrany cyklu c odpovídají nerovnostem

$$\begin{aligned} x_2 - x_1 &\leq w(v_1, v_2) \\ x_3 - x_2 &\leq w(v_2, v_3) \\ &\vdots \\ x_k - x_{k-1} &\leq w(v_{k-1}, v_k) \end{aligned}$$

   přípustné řešení x musí splňovat všechny tyto nerovnosti
   po sečtení všech nerovností dostaneme $0 \leq w(c)$, což je spor s předpokladem o záporné délce cyklu c

Algoritmus pro problém lineárních nerovnic

1. vytvoř graf lineárních nerovnic
   - $n + 1$ vrcholů
   - $m + n$ hran
   - časová složitost $\Theta(m + n)$

2. najdi nejkratší cesty z vrcholu v_0 algoritmem Bellmana a Forda
   - časová složitost $\mathcal{O}((n + 1)(m + n)) = \mathcal{O}(n^2 + nm)$

3. když algoritmus vrátí hodnotu $\textsc{False}$, tak problém lineárních nerovnic nemá přípustné řešení když algoritmus vrátí hodnotu $\textsc{True}$, tak přípustným řešením je $x = (\delta(v_0, v_1), \delta(v_0, v_2), \ldots, \delta(v_0, v_n))$