Grafové algoritmy

Zbyněk Křivka, Tomáš Masopust

23. října 2012

Obsah

1 Úvod **3**

2 Grafy **6**

3 Reprezentace grafů **11**

4 Prohledávání do šířky **14**

5 Prohledávání do hloubky **18**
5.1 Topologické uspořádání 22
5.2 Silně souvislé komponenty (Strongly Connected Components) 24

6 Minimální kostry **27**

7 Nejkratší cesty z jednoho do všech uzlů **31**
7.1 Nejkratší cesty z jednoho do všech uzlů v acyklických grafech 34

8 Nejkratší cesty ze všech uzlů do všech ostatních uzlů **38**

9 Toky v síti **45**

10 Řez v síti **50**

11 Maximální párování v bipartitním grafu **53**

12 Barvení grafů **55**
12.1 Hranové barvení grafů 55
12.2 (Vrcholové) barvení grafů 59
12.3 Chromatický polynom 59

13 Eulerovské tahy **63**

14 Hamiltonovské cesty a kružnice **66**

Literatura

- Cormen, Leiserson, Rivest, Stein: *Introduction to algorithms.* The MIT Press and McGraw-Hill, 2001.

- Gibbons: *Algorithmic Graph Theory.* Cambridge University Press, 1985.

1 Úvod

Základní pojmy

- Neformálně, algoritmus je dobře definovaná výpočetní procedura, která přijímá nějaké hodnoty (množinu hodnot) jako vstup a produkuje nějakou hodnotu (množinu hodnot) jako výstup.
- Algoritmus je tedy posloupnost výpočetních kroků, které transformují vstup na výstup.
- Datová struktura je způsob, jak uložit a organizovat data tak, aby k nim bylo co nejvíce ulehčeno přistupování a modifikování.

Požadavky kladené na algoritmy

- Algoritmus skončí pro každý (správně zadaný) vstup.
- Výstup algoritmu je korektní.
- Čas ani paměť nejsou neomezeny.
- Existuje mnoho řešení určitého problému, ty se však dramaticky liší svojí efektivností.
- Velký důraz kladen na časovou a prostorovou efektivnost.

Složitost algoritmu

Časová složitost algoritmu:

- Maximální počet "primitivních" kroků vykonaných daným algoritmem nad všemi vstupy stejné "velikosti", tj. počet kroků v nejhorším případě.
- Popsána funkcí závislou na velikosti vstupu, $T(n)$.

Prostorová složitost algoritmu:

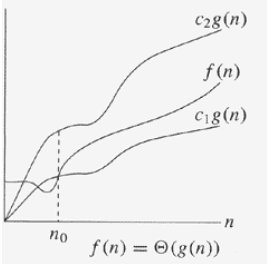
- Maximální velikost obsazené paměti během výpočtu algoritmu nad všemi vstupy stejné "velikosti".
- Popsána funkcí závislou na velikosti vstupu, $S(n)$.

n může být obecně vektor hodnot.

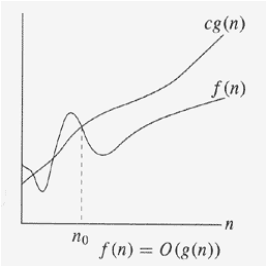
Θ-notace

Nechť $g(n)$ je funkce.

- $\Theta(g(n)) = \{f(n) :$ existují $c_0, c_1, n_0 > 0$ takové, že $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ pro $n \geq n_0\}$.
- Obvykle píšeme $f(n) = \Theta(g(n))$ místo $f(n) \in \Theta(g(n))$.
- $\frac{1}{2}n^2 - 3n = \Theta(n^2)$ – ověřte pro $c_1 = \frac{1}{14}, c_2 = \frac{1}{2}, n_0 = 7$.



Obrázek 1: Θ-notace.



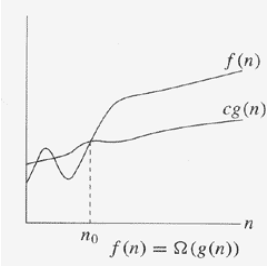
Obrázek 2: *O*-notace.

***O*-notace**
Nechť $g(n)$ je funkce.

- $O(g(n)) = \{f(n) :$ existují $c, n_0 > 0$ takové, že $0 \leq f(n) \leq cg(n)$ pro $n \geq n_0\}$.
- $\Theta(g(n)) \subseteq O(g(n))$.
- $n = O(n^2)$, ale $n \neq \Theta(n^2)$.

Ω-notace
Nechť $g(n)$ je funkce.

- $\Omega(g(n)) = \{f(n) :$ existují $c, n_0 > 0$ takové, že $0 \leq cg(n) \leq f(n)$ pro $n \geq n_0\}$.



Obrázek 3: Ω-notace.

Theorem 1. *Pro libovolné funkce $f(n)$ a $g(n)$ platí $f(n) = \Theta(g(n))$, právě když $f(n) = O(g(n))$ a $f(n) = \Omega(g(n))$.*

o-notace a ω-notace

Nechť $g(n)$ je funkce.

- $o(g(n)) = \{f(n) :$ pro každé $c > 0$ existuje $n_0 > 0$ takové, že $0 \leq f(n) < cg(n)$ pro $n \geq n_0\}$.
- $\omega(g(n)) = \{f(n) :$ pro každé $c > 0$ existuje $n_0 > 0$ takové, že $0 \leq cg(n) < f(n)$ pro $n \geq n_0\}$.
- $f(n) \in \omega(g(n))$, právě když $g(n) \in o(f(n))$.
- $2n = o(n^2)$, ale $2n^2 \neq o(n^2)$.
- $f(n) = o(g(n))$ pokud $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $n^2/2 = \omega(n)$, ale $n^2/2 \neq \omega(n^2)$.
- $f(n) = \omega(g(n))$ pokud $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

Vlastnosti

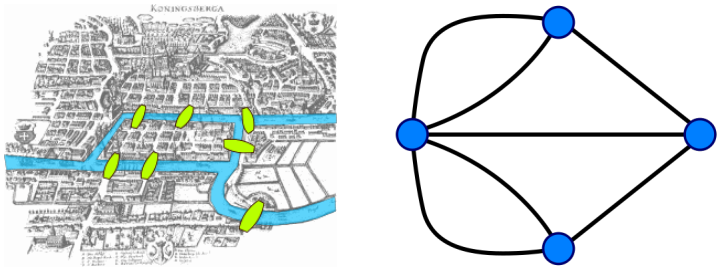
Nechť $f(n)$ a $g(n)$ jsou (asymptoticky kladné) funkce.

- *Tranzitivita* $f(n) = X(g(n))$ a $g(n) = X(h(n))$ implikuje $f(n) = X(h(n))$, pro $X \in \{\Theta, O, \Omega, o, \omega\}$.
- *Reflexivita* $f(n) = X(f(n))$, pro $X \in \{\Theta, O, \Omega\}$.
- *Symetrie* $f(n) = \Theta(g(n))$, právě když $g(n) = \Theta(f(n))$.
- *Transponovaná symetrie* $f(n) = O(g(n))$, právě když $g(n) = \Omega(f(n))$. $f(n) = o(g(n))$, právě když $g(n) = \omega(f(n))$.
- *Ne vždy porovnatelné* n a $n^{1+\sin(n)}$ jsou neporovnatelné.

2 Grafy

Grafy: Úvod, historie, definice

- Leonhard Euler, *Sedm mostů města Královce*, 1736.
- Otázka: Je možné přejít všechny mosty, ale každý pouze jednou?



Obrázek 4: Sedm mostů města Královce a jejich grafická reprezentace.

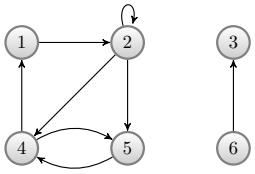
Orientovaný graf G je dvojice

$$G = (V, E),$$

kde

- V je konečná množina uzlů a
- $E \subseteq V^2$ je množina hran.

Hrana (u, u) se nazývá smyčka.
Pokud (u, v) je hrana, říkáme, že u a v jsou incidentní.



Obrázek 5: Orientovaný graf

Graf $G' = (V', E')$ je podgraf grafu $G = (V, E)$, jestliže

- $V' \subseteq V$ a $E' \subseteq E$.

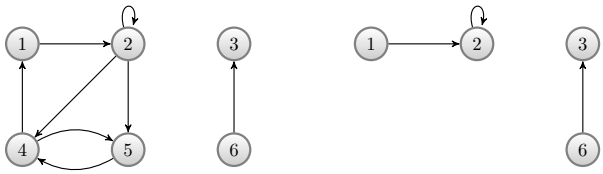
Mějme $V'' \subseteq V$. Podgraf indukovaný V'' je graf $G'' = (V'', E'')$, kde

- $E'' = \{(u, v) \in E : u, v \in V''\}$.

Neorientovaný graf G je dvojice

$$G = (V, E),$$

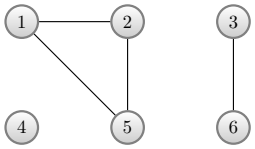
kde



Obrázek 6: Graf a jeho podgraf indukovaný množinou $\{1, 2, 3, 6\}$.

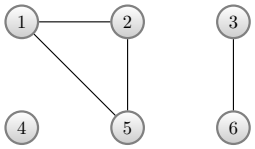
- V je konečná množina uzlů a
- $E \subseteq \binom{V}{2}$ je množina hran.

Poznámka
Hrana je množina $\{u, v\}$, kde $u, v \in V$ a $u \neq v$. I v neorientovaném případě však budeme hrany značit (u, v) a považovat (u, v) a (v, u) za stejnou hranu. Smyčky nejsou povoleny.



Obrázek 7: Neorientovaný graf

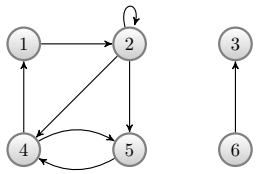
- Stupeň uzlu u v neorientovaném grafu je počet uzlů s ním incidentních, značí se $d(u)$.
- $d(1) = d(2) = d(5) = 2$, $d(3) = d(6) = 1$, $d(4) = 0$.
- Pokud $d(u) = 0$, nazývá se u izolovaný uzel.



Obrázek 8: Neorientovaný graf

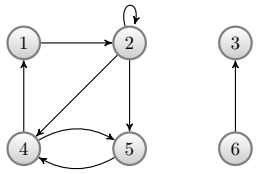
- Výstupní stupeň uzlu u v orientovaném grafu je počet hran z něj vycházejících, značí se $d_-(u)$.
- Vstupní stupeň uzlu u je počet hran do něj vstupujících, značí se $d_+(u)$.
- Stupeň uzlu u je součet výstupního a vstupního stupně.
- $d_-(2) = 3$, $d_+(2) = 2$, $d(2) = 5$.

- Posloupnost uzlů $\langle v_0, v_1, v_2, \ldots, v_k \rangle$, kde $(v_{i-1}, v_i) \in E$ pro $i = 1, 2, \ldots, k$, se nazývá sled délky k z v_0 do v_k.



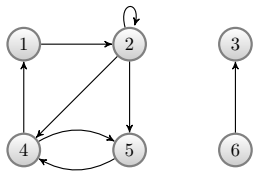
Obrázek 9: Orientovaný graf

- Sled je tedy určen uzly $v_0, v_1, \ldots, v_k$ a hranami (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k). Délka sledu je počet jeho hran.
- Rovněž se připouští sled délky 0 z u do u, pro všechny uzly u.
- Pokud existuje sled s z u do u', říkáme, že u' je dosažitelný z u sledem s, značeno $u \overset{s}{\rightsquigarrow} u'$.
- Tah je sled, ve kterém se neopakují hrany.
- Cesta je sled, ve kterém se neopakují uzly.

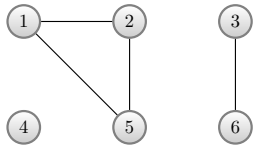


- $\langle 1, 2, 5, 4 \rangle$ je cesta
- $\langle 2, 5, 4, 5 \rangle$ je sled, který není cestou (jde o tah?)
- Podsled (podtah,podcesta) sledu $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ je podposloupnost sousedících uzlů, tj. $\langle v_i, v_{i+1}, v_{i+2}, \ldots, v_j \rangle$, pro $0 \leq i \leq j \leq k$.

- Sled (tah, cesta) $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ se nazývá uzavřený, pokud obsahuje alespoň jednu hranu a $v_0 = v_k$.
- Pro neorientovaný graf požadujeme $k \geq 3$.
- Uzavřená cesta se nazývá kružnice. Orientovaná kružnice se nazývá cyklus.

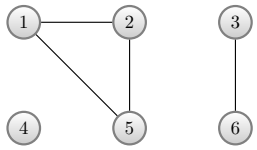


- $\langle 2, 2 \rangle$ je cyklus délky 1
- $\langle 1, 2, 4, 1 \rangle$ je cyklus
- $\langle 1, 2, 4, 5, 4, 1 \rangle$ je ale uzavřený sled (i tah), který není cyklus



- $\langle 1, 2, 5, 1 \rangle$ je kružnice
- $\langle 3, 6, 3 \rangle$ není kružnice, nebo ano?

- Orientovaný graf se nazývá prostý, pokud neobsahuje smyčky.
- Graf bez cyklů (kružnic) se nazývá acyklický.
- Neorientovaný graf se nazývá souvislý, pokud mezi libovolnými dvěma uzly existuje cesta.
- Souvislé komponenty grafu jsou třídy ekvivalence množiny uzlů podle relace "je dosažitelný z".

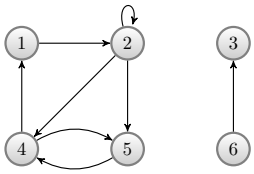


Graf má tři souvislé komponenty:

- $\{1, 2, 5\}$
- $\{3, 6\}$
- $\{4\}$
- Orientovaný graf se nazývá silně souvislý, pokud mezi libovolnými dvěma uzly existuje orientovaná cesta.
- Silně souvislé komponenty grafu jsou třídy ekvivalence množiny uzlů podle relace "jsou vzájemně dosažitelné".

Graf má tři silně souvislé komponenty:

- $\{1, 2, 4, 5\}$
- $\{3\}$
- $\{6\}$


1
2
3
4
5
6

3 Reprezentace grafů

Nechť $G = (V, E)$ je graf. Zavedeme následující značení:

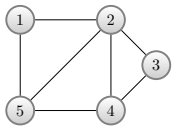
- $n = |V|$
- $m = |E|$.

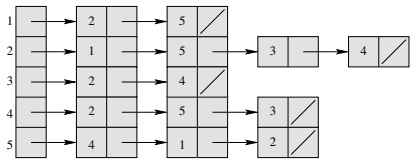
1. *Seznam sousedů*
   - preferovaná reprezentace,
   - výhodnější zejména pro řídké grafy, tj. takové grafy, kde $m \ll n^2$,
   - většina dále uvedených algoritmů používá právě tuto reprezentaci.
2. *Matice sousednosti*
   - může být výhodnější pro husté grafy, tj. ty, kde m je skoro n^2
   - nebo když potřebujeme rychle rozhodnout, zda graf obsahuje hranu spojující dva dané uzly.

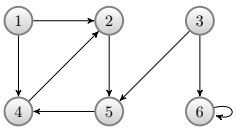
Seznam sousedů

Seznam sousedů grafu $G = (V, E)$ sestává z

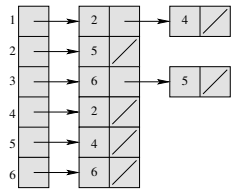
- pole $Adj[1 \dots n]$ mající n seznamů, jeden pro každý uzel z V,
- kde $Adj[u]$ obsahuje všechny uzly v takové, že $(u, v) \in E$.







- Pokud G je orientovaný graf, pak součet délek seznamů seznamu sousedů je m, protože každá hrana tvaru (u, v) je reprezentována v vyskytujícím se v $Adj[u]$.
- Pokud G je neorientovaný graf, pak součet délek seznamů seznamu sousedů je $2m$, protože každá hrana (u, v) je reprezentována v vyskytujícím se v $Adj[u]$ a u vyskytujícím se v $Adj[v]$.
- Třída paměťové složitosti je v obou případech stejná, $\Theta(m + n)$.



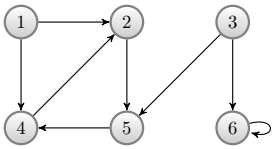
Ohodnocený graf

- Ohodnocený graf je takový graf, kde každá jeho hrana má přiřazenu nějakou hodnotu, typicky danou *váhovou funkcí* $w : E \to \mathbb{R}$.
- Seznam sousedů se snadno rozšíří pro ohodnocené grafy tak, že hodnota $w(u, v)$ je uložena s uzlem v v seznamu $Adj[u]$.
- Nevýhoda: zjistit, zda je hrana (u, v) přítomná v grafu znamená hledat v v celém seznamu $Adj[u]$.

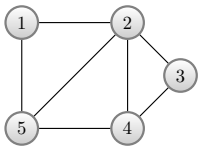
Matice sousednosti

Nechť $G = (V, E)$ je graf a předpokládejme, že uzly jsou nějak očíslovány čísly $1, 2, \ldots, n$. Matice sousednosti $A = (a_{ij})$ je čtvercová matice řádu n taková, že

$$a_{ij} = \begin{cases} 1 & \text{pokud } (i, j) \in E, \\ 0 & \text{jinak.} \end{cases}$$



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

- Množství obsazené paměti je bez ohledu na počtu hran $\Theta(n^2)$.
- Lepší nebo horší?
- *Transponovaná* matice k $A = (a_{ij})$ je matice $A^T = (a_{ij}^T)$, kde $a_{ij}^T = a_{ji}$.
- Pro matici sousednosti A neorientovaného grafu platí $A = A^T$.
- Stačí tedy uložit do paměti pouze část matice nad diagonálou.
- Nechť $G = (V, E)$ je ohodnocený graf, pak

$$a_{ij} = \begin{cases} w(i,j) & \text{pokud } (i,j) \in E, \\ NIL & \text{jinak,} \end{cases}$$

kde NIL je nějaká unikátní hodnota, většinou 0 či ∞.

4 Prohledávání do šířky

Prohledávání do šířky (BFS)

- Vstup: (ne)orientovaný graf $G = (V, E)$ a uzel $s \in V$.
- Prochází všechny uzly dostupné z s a počítá jejich vzdálenost (počet hran) od s.
- Vytváří *strom prohledávání do šířky* s kořenem s obsahující všechny uzly dosažitelné z s. Cesta z s do v je nejkratší cestou v grafu.
- Při procházení grafu obarvuje uzly bílou, šedou a černou barvou.
- Reprezentace grafu – seznam sousedů.
- $color[u] \in \{\text{WHITE, GRAY, BLACK}\}$.
- $\pi[u]$ je předchůdce u na cestě z s.
- $d[u]$ je vzdálenost u od s (počet hran, dále bude rozšířeno na ohodnocené grafy).

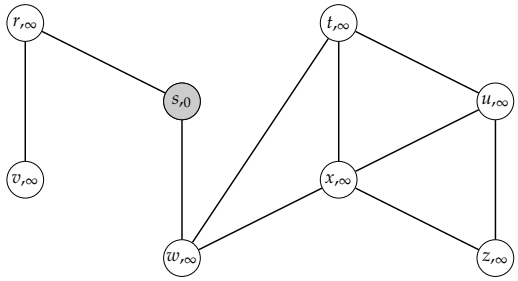
```
BFS(G, s)
 1  for každý uzel u ∈ V − {s}
 2      do color[u] ← WHITE
 3         d[u] ← ∞
 4         π[u] ← NIL
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      do u ← DEQUEUE(Q)
12         for každý v ∈ Adj[u]
13             do if color[v] =WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u] + 1
16                         π[v] ← u
17                         ENQUEUE(Q, v)
18         color[u] ← BLACK
```

BFS – příklad

Analýza BFS

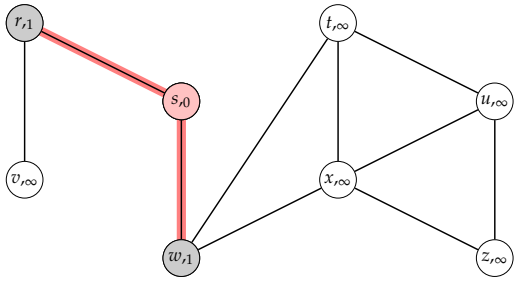
- Ve **while**-cyklu již není možno obarvit uzel na bílo.
- Řádek 13 proto zaručuje, že každý uzel bude zařazen do fronty (a následně vybrán z fronty) nejvýše jednou.
- Operace vkládání a vybírání prvku z fronty je konstantní, tj. $O(1)$, takže celkový čas na operace fronty je $O(n)$.

BFS – příklad



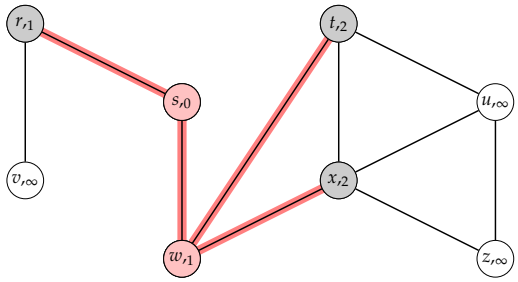
$Q = s$

BFS – příklad



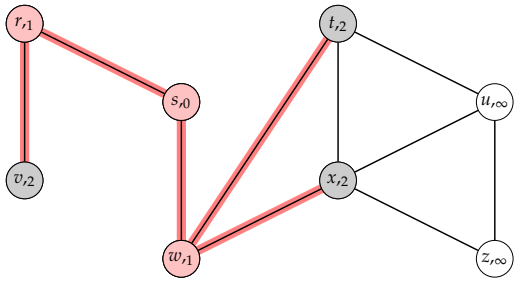
$Q = wr$

BFS – příklad



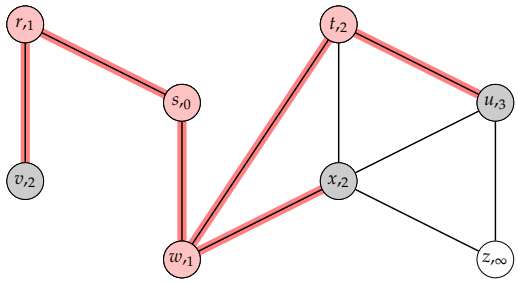
$Q = rtx$

BFS – příklad



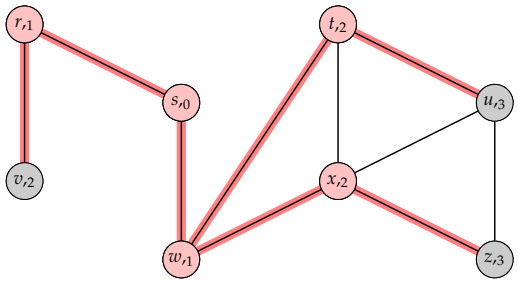
$Q = txv$

BFS – příklad



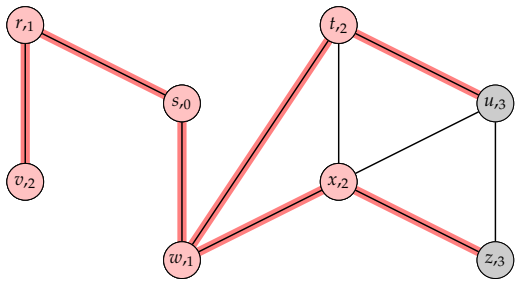
$Q = xvu$

BFS – příklad



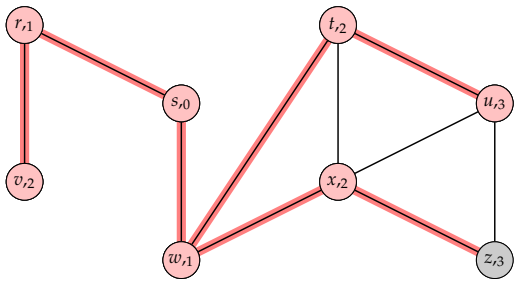
$Q = vuz$

BFS – příklad

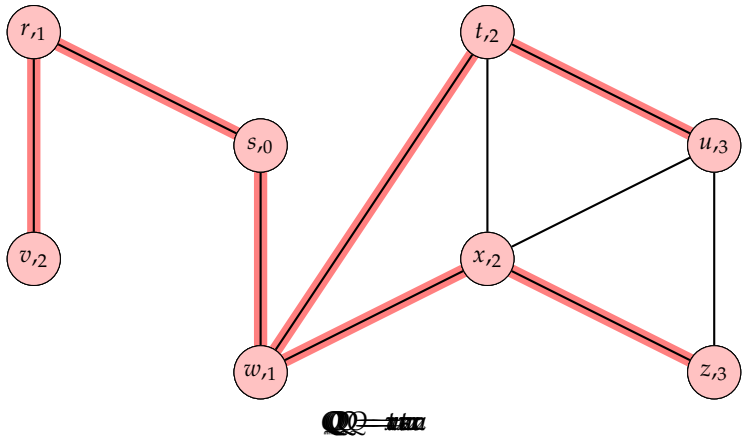


$Q = uz$

BFS – příklad



$Q = z$



- Protože se seznam sousedů daného uzlu prochází pouze při jeho vybrání z fronty, je seznam skenován nejvýše jednou.
- Jelikož je suma délek těchto seznamů rovna $\Theta(m)$, je celkový čas skenování seznamu sousedů $O(m)$.
- Inicializace zabere čas $O(n)$, proto je celkový čas algoritmu $O(m+n)$, tj. lineární vzhledem k velikosti reprezentace grafu G.

Nejkratší cesty

- Jak již bylo řečeno, BFS rovněž určí vzdálenost všech uzlů od daného vstupního uzlu s.
- Definujme *nejkratší vzdálenost* $\delta(s,v)$ z s do v jako minimální počet hran na jakékoliv cestě z s do v. Pokud cesta neexistuje, je $\delta(s,v) = \infty$.
- Cesta délky $\delta(s,v)$ z s do v je *nejkratší cesta* z s do v.

Lemma 2. *Nechť $G = (V,E)$ je orientovaný či neorientovaný graf a nechť $s \in V$ je libovolný uzel. Pak pro každou hranu $(u,v) \in E$ platí, že*

$$\delta(s,v) \leq \delta(s,u) + 1\,.$$

Důkaz.
- Pokud je uzel u dosažitelný z s, pak je rovněž uzel v dosažitelný z s. To ale znamená, že nejkratší cesta z s do v nemůže být delší než nejkratší cesta z s do u následovaná hranou (u,v). Nerovnost tedy platí.
- Pokud uzel u není dosažitelný z s, pak $\delta(s,u) = \infty$ a nerovnost opět platí.

□

Lemma 3. *Nechť $G = (V,E)$ je orientovaný či neorientovaný graf a přepokládejme, že BFS je spuštěno na G z uzlu $s \in V$. Pak po ukončení BFS platí, že $d[v] \geq \delta(s,v)$ pro každé $v \in V$.*

Důkaz.
- Indukcí vzhledem k počtu ENQUEUE operací. IP je $d[v] \geq \delta(s,v)$ pro všechna $v \in V$.
- Báze: $d[s] = 0 = \delta(s,s)$ a $d[v] = \infty \geq \delta(s,v)$, $v \in V - \{s\}$.
- Nechť v je bílý uzel prozkoumaný během prohledávání z u. Z IP máme $d[u] \geq \delta(s,u)$. Z řádku 15 BFS, IP a předchozího lemmatu máme

  $$d[v] = d[u] + 1 \geq \delta(s,u) + 1 \geq \delta(s,v)\,.$$

  Uzel v je poté vložen do fronty a už nikdy není vložen znovu do fronty, protože je šedý a řádky 14–17 se vykonávají pouze pro bílé uzly, tj. hodnota $d[v]$ se již nezmění.

□

Lemma 4. *Nechť během provádění BFS na grafu $G = (V, E)$ obsahuje fronta Q uzly $\langle v_1, v_2, \ldots, v_r \rangle$, kde v_1 je první prvek Q (začátek fronty) a v_r je poslední prvek Q (konec fronty). Pak $d[v_r] \leq d[v_1] + 1$ a $d[v_i] \leq d[v_{i+1}]$ pro $i = 1, 2, \ldots, r-1$.*

Důkaz.

- Indukcí vzhledem k počtu operací fronty. Na počátku je $Q = \langle s \rangle$ a tvrzení platí. Tvrzení platí po obou frontových operacích:
- v_1 odebráno, pak v_2 nový první prvek (pokud se fronta vyprázdní, trvzení platí triviálně). Z IP máme $d[v_1] \leq d[v_2]$. Pak ale $d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$, zbytek nerovností je nezměněn.
- v_{r+1} vloženo do Q (řádek 17). V tu dobu je již z Q vyjmut uzel u, jehož seznam se prochází. Z IP máme $d[u] \leq d[v_1]$. Tedy, $d[v_{r+1}] = d[u] + 1 \leq d[v_1] + 1$. Proto $d[v_r] \leq_{IP} d[u] + 1 = d[v_{r+1}]$. Zbytek nerovností je nezměněn.

□

Corollary 5. *Nechť uzly v_i a v_j jsou vloženy do fronty během provádění BFS a že v_i je vloženo před v_j. Pak $d[v_i] \leq d[v_j]$ v okamžiku vložení v_j do fronty.*

Důkaz. Z předchozího lemmatu a vlastnosti, že každý uzel obdrží konečnou hodnotu d nejvýše jednou během výpočtu. □

Theorem 6 (Korektnost algoritmu BFS)**.** *Nechť $G = (V, E)$ je (ne)orientovaný graf a $s \in V$. Pak $BFS(G, s)$ prozkoumá všechny uzly $v \in V$ dosažitelné z s a po ukončení platí, že $d[v] = \delta(s, v)$ pro všechna $v \in V$. Pro každý uzel $v \neq s$ dosažitelný z s navíc platí, že jedna z nejkratších cest z s do v je nejkratší cesta z s do $\pi[v]$ následovaná hranou $(\pi[v], v)$.*

Důkaz.

- Sporem. Nechť v je uzel s minimální $\delta(s, v)$ takový, že $d[v] \neq \delta(s, v)$. Zřejmě $v \neq s$.
- Lemma 3 říká, že $d[v] \geq \delta(s, v)$, proto $d[v] > \delta(s, v)$. Navíc, v musí být dosažitelný z s, protože jinak je $\delta(s, v) = \infty \geq d[v]$.
- Nechť u je uzel bezprostředně předcházející v na nejkratší cestě z s do v, tj. $\delta(s, v) = \delta(s, u) + 1$. Protože $\delta(s, u) < \delta(s, v)$ a vzhledem k volbě v, platí $d[u] = \delta(s, u)$.
- Celkem tedy $d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1$.

Důkaz (pokračování).

- Uvažme nyní BFS v době, kdy u je vybíráno z Q na řádku 11, tj. v je buď bílý, šedý, nebo černý.
- v je bílý, pak řádek 15 nastavuje $d[v] = d[u] + 1$ – spor.
- v je černý, pak v již bylo vybráno z Q a podle Důsledku 5 je $d[v] \leq d[u]$ – spor.
- v je šedý, pak v bylo obarveno při výběru nějakého uzlu w, jenž byl vybrán z Q dříve než u. Navíc, $d[v] = d[w] + 1$. Podle Důsledku 5 je $d[w] \leq d[u]$, tj. $d[v] \leq d[u] + 1$ – spor.
- Tedy $d[v] = \delta(s, v)$ pro všechna $v \in V$. Navíc, všechny uzly dosažitelné z s musí být prozkoumány, protože jinak je jejich d hodnota nekonečno.
- Konečně, všimněme si, že když $\pi[v] = u$, pak $d[v] = d[u] + 1$, tj. nejkratší cestu z s do v můžeme získat přidáním hrany $(\pi[v], v)$ k nejkratší cestě z s do $\pi[v]$.

□

Strom prohledávání do šířky (BFS strom)

- Nechť π je pole předchůdců vypočtené procedurou $BFS(G, s)$ na nějakém grafu $G = (V, E)$ a uzlu $s \in V$.
- *Podgraf předchůdců* grafu G je graf $G_\pi = (V_\pi, E_\pi)$, kde
- $V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$ a
- $E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}\}$.
- Graf G_π je *BFS strom*, pokud V_π obsahuje pouze uzly dosažitelné z s a pro všechna $v \in V_\pi$ existuje pouze jediná cesta z s do v, která je zároveň nejkratší cestou.
- BFS strom je strom, neboť platí, že je souvislý a $|E_\pi| = |V_\pi| - 1$.

Lemma 7. *Nechť G je orientovaný či neorientovaný graf. Procedura BFS konstruuje π tak, že G_π je BFS strom.*

Důkaz.
- Řádek 16 BFS nastavuje $\pi[v] = u$, právě když $(u, v) \in E$ a $\delta(s, v) < \infty$.
- V_π tedy obsahuje pouze uzly dosažitelné z s.
- Protože G_π je strom, obsahuje pouze jednu cestu z s do každého z ostatních uzlů.
- Induktivní aplikací Teorému 6 dostáváme, že každá tato cesta je nejkratší.

□

Tisk nejkratší cesty z s do v

```
PRINT-PATH(G, s, v)
1  if v = s
2     then print s
3     else if π[v] = NIL
4             then print "Cesta z" s "do" v "neexistuje!"
5             else PRINT-PATH(G, s, π[v])
6                  print v
```

Složitost $O(n)$.

5 Prohledávání do hloubky

Prohledávání do hloubky (DFS)

- Vstup: (ne)orientovaný graf $G = (V, E)$.
- Na rozdíl od BFS prochází DFS všechny uzly.
- Při procházení grafu obarvuje uzly bílou, šedou a černou barvou.
- Používá pole předchůdců π.
- Vytváří *les prohledávání do hloubky* obsahující všechny uzly grafu, který je definován takto: $G_\pi = (V, E_\pi)$, kde
$$E_\pi = \{(\pi[v], v) : v \in V,\ \pi[v] \neq NIL\}.$$
- Reprezentace grafu – seznam sousedů.
- $color[u] \in \{\text{WHITE, GRAY, BLACK}\}$.
- $d[u]$ je časová známka prvního prozkoumání uzlu (obarvení na šedo).
- $f[u]$ je časová známka dokončení prozkoumávání seznamu sousedů uzlu u (začernění uzlu).
- $1 \leq d[u] < f[u] \leq 2n$.
- $color[u] = WHITE$ v čase před $d[u]$.
- $color[u] = GRAY$ v čase $d[u]$ až $f[u]$.
- $color[u] = BLACK$ v čase po $f[u]$.
- *time* je globální proměnná.

```
DFS(G)
1  for každý uzel u ∈ V
2      do color[u] ← WHITE
3         π[u] ← NIL
4  time ← 0
5  for každý uzel u ∈ V
6      do if color[u] = WHITE
7            then DFS-VISIT(u)
```

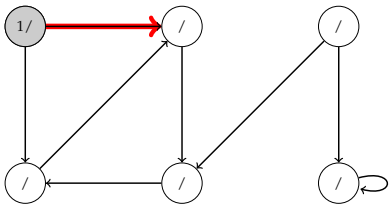
```
DFS-VISIT(u)
1  color[u] ← GRAY
2  time ← time + 1
3  d[u] ← time
4  for každý uzel v ∈ Adj[u]
5      do if color[v] = WHITE
6            then π[v] ← u
7                 DFS-VISIT(v)
8  color[u] ← BLACK
9  f[u] ← time ← time + 1
```

DFS – příklad

Časová složitost DFS

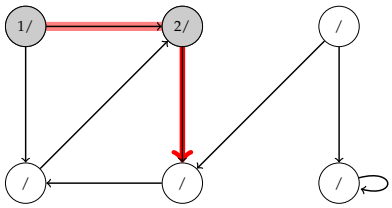
- Cykly na řádcích 1–3 a 5–7 bez volání funkce DFS-VISIT vezmou čas $\Theta(n)$.

DFS – příklad



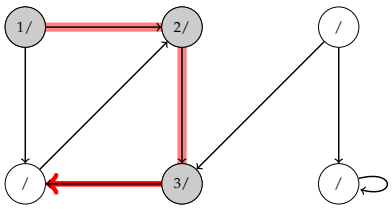
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



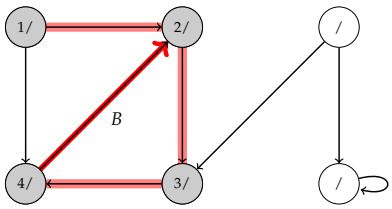
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



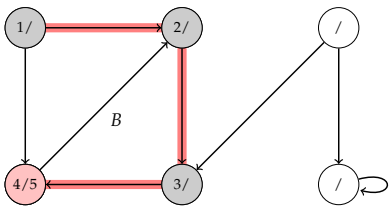
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



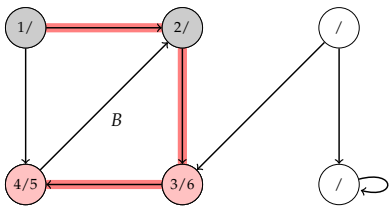
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



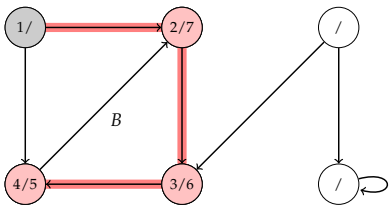
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



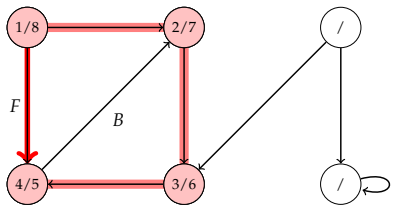
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



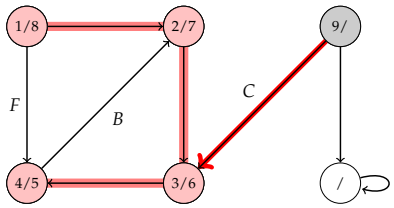
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



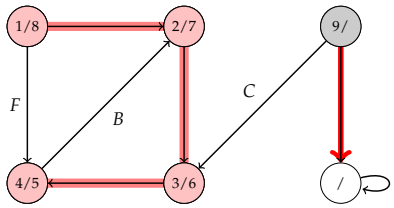
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



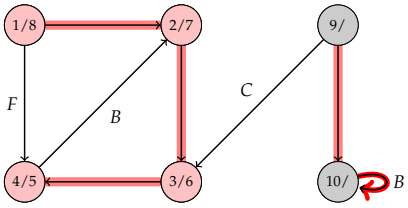
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



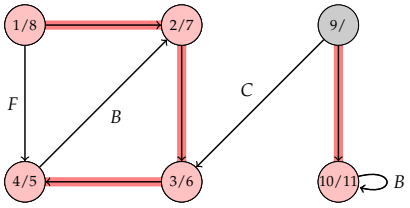
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



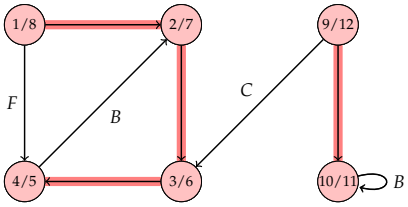
Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad

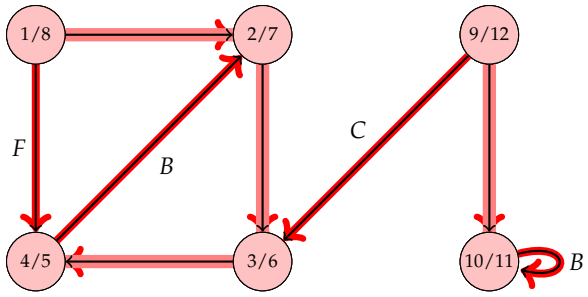


Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

DFS – příklad



Obrázek: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.



Obrázek 10: B značí zpětnou (Back) hranu, F značí dopřednou (Forward) hranu a C značí křížící (Cross) hranu.

Časová složitost DFS-VISIT

- Funkce je volána pouze na bílé uzly a první věc, kterou udělá, je jejich obarvení na šedo. Je tedy volána přesně jednou pro každý uzel $v \in V$.
- Pro každý uzel v je cyklus 4–7 prováděn $|Adj[v]|$-krát.
- Jelikož $\sum_{v \in V} |Adj[v]| = \Theta(m)$, je celková cena řádků 4–7 $\Theta(m)$.
- Celková složitost je tedy $\Theta(m + n)$.

Závorková věta

V prohledávání do hloubky (ne)orientovaného grafu $G = (V, E)$ pro libovolné dva uzly u a v platí právě jedno z následujících:

- intervaly $[d[u], f[u]]$ a $[d[v], f[v]]$ jsou disjunktní a ani u není následníkem v ani v není následníkem u v DFS lese,
- interval $[d[u], f[u]]$ je celý obsažen v intervalu $[d[v], f[v]]$ a u je následníkem v v nějakém DFS stromě, nebo
- interval $[d[v], f[v]]$ je celý obsažen v intervalu $[d[u], f[u]]$ a v je následníkem u v nějakém DFS stromě.

Důkaz pro $d[u] < d[v]$ (opačná nerovnost se dokáže podobně).

- $d[v] < f[u]$, pak v bylo prozkoumáno, když u bylo šedé. Protože v prozkoumáno později než u, je v dokončeno před u, tj. $f[v] < f[u]$.
- $f[u] < d[v]$, pak z vlastnosti $d[v] < f[v]$ plyne, že oba intervaly jsou disjunktní. Navíc, když je prozkoumáván uzel z jednoho intervalu, nejsou uzly druhého šedé, proto nemůže být jejich následníkem.

Corollary 8. *Uzel v je následníkem uzlu u v DFS lese grafu $G = (V, E)$ právě tehdy, když*

$$d[u] < d[v] < f[v] < f[u]\,.$$

Věta o bílé cestě

V DFS lese grafu $G = (V, E)$ je uzel v následníkem uzlu u právě tehdy, když v čase $d[u]$ existuje cesta z u do v obsahující pouze bílé uzly.

Důkaz. $\Rightarrow$: Nechť v je následníkem u. Nechť w je uzel na cestě z u do v v DFS lese. Protože w je následníkem u, je podle předchozího důsledku $d[u] < d[w]$. Tedy w je v čase $d[u]$ bílé.

$\Leftarrow$: Nechť v je nejbližším uzlem k u dosažitelným z u v čase $d[u]$ nějakou bílou cestou takový, že není následníkem u v DFS lese.

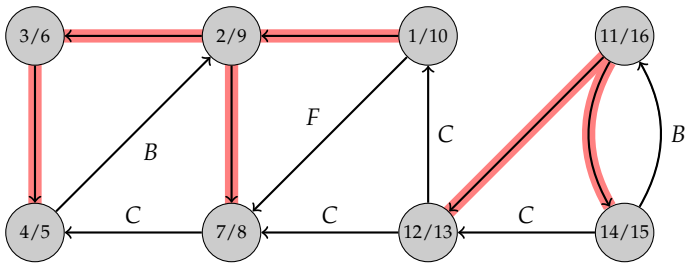
- Nechť w je předchůdce v na dané cestě. Pak w je následníkem u a podle předchozího důsledku je $f[w] \leq f[u]$ (w může být u).
- Protože v musí být objeveno po u, ale před ukončením w, je $d[u] < d[v] < f[w] \leq f[u]$.
- Závorková věta říká, že interval $[d[v], f[v]]$ je celý obsažen v intervalu $[d[u], f[u]]$. Předchozí důsledek dává, že v je následníkem u.

□

Klasifikace hran

1. **Stromové hrany** jsou hrany DFS lesa G_π. (u, v) je stromová hrana, jestliže v bylo poprvé objeveno prozkoumáváním hrany (u, v). Zvýrazněné hrany v předchozích příkladech.
2. **Zpětné hrany** jsou hrany (u, v) spojující u s předchůdcem v v DFS lese. Smyčky jsou zpětné hrany. V příkladech značeno B.
3. **Dopředné hrany** jsou hrany (u, v) spojující u s následníkem v v DFS lese. V příkladech značeno F.
4. **Cross hrany** jsou všechny ostatní hrany.

Příklad



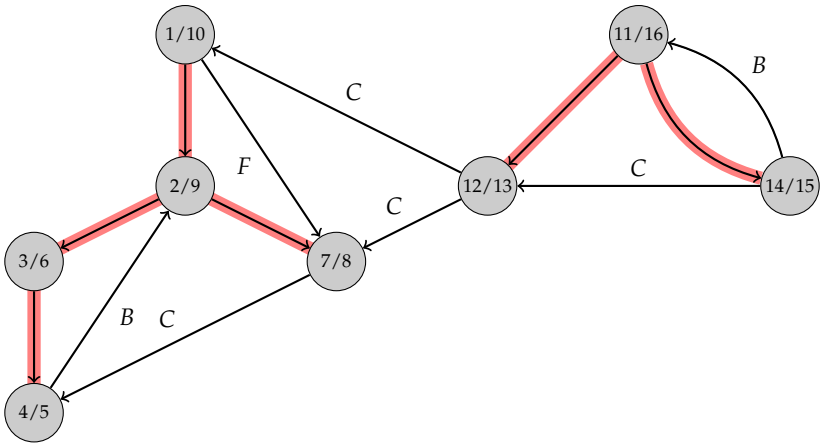
Nakreslení

Každý graf lze nakreslit tak, že stromové a dopředné hrany vedou dolů a zpětné hrany vedou nahoru.

DFS a klasifikace hran

Nechť (u, v) je hrana. Pak během provádění DFS je možno tuto hranu klasifikovat podle barvy uzlu v následovně:

1. bílá indikuje stromovou hranu,
2. šedá indikuje zpětnou hranu a
3. černá indikuje dopřednou nebo cross hranu.
   - (u, v) je dopředná hrana, jestliže $d[u] < d[v]$ a
   - cross hrana, jestliže $d[u] > d[v]$.



Neorientovaný graf

Theorem 9. *V prohledávání neorientovaného grafu G je každá hrana buď stromová, nebo zpětná.*

Důkaz.

- Nechť (u, v) je libovolná hrana grafu G a nechť $d[u] < d[v]$.
- Pak v se stane černý v době, kdy u je šedý.
- Pokud je hrana (u, v) poprvé prozkoumána ve směru z u do v, je v bílý – jinak bychom již tuto hranu prozkoumali ve směru od v do u. (u, v) je tedy stromová hrana.
- Pokud je hrana (u, v) poprvé prozkoumána ve směru z v do u, je u stále šedý – protože u je šedý v době, kdy je hrana poprvé prozkoumána. (u, v) je tedy zpětná hrana.

□

5.1 Topologické uspořádání

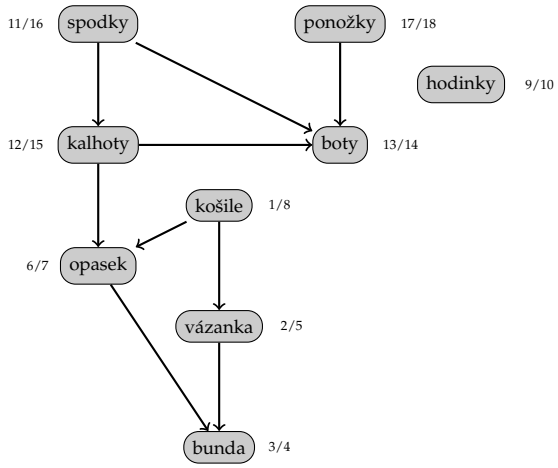
Topologické uspořádání

- Aplikace DFS
- *Topologické uspořádání* orientovaného acyklického grafu $G = (V, E)$ je lineární uspořádání všech jeho uzlů tak, že pokud $(u, v) \in E$, pak u předchází v v onom uspořádání.
- Pokud G není acyklický, pak žádné topologické uspořádání neexistuje.

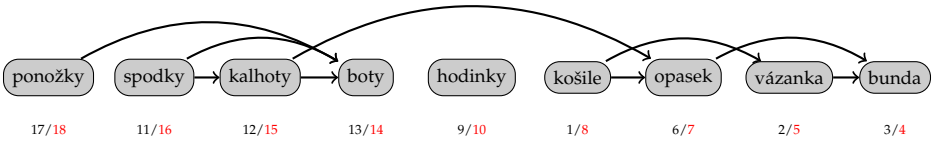
TOPOLOGICAL-SORT(G)
1 zavolej DFS(G) pro výpočet hodnot $f[v]$
2 každý dokončený uzel zařaď na začátek seznamu uzlů L
3 **return** L

- Složitost DFS je $\Theta(m + n)$, přidání uzlu do seznamu je konstantní, proto je celková složitost $\Theta(m + n)$.

Topologické uspořádání – příklad



Topologické uspořádání – příklad



Lemma 10. *Orientovaný graf G je acyklický, právě když* DFS(G) *nenajde žádnou zpětnou hranu.*

Důkaz. $\Rightarrow$: Nechť (u, v) je zpětná hrana. Pak u je následníkem v v DFS lese, tj. existuje cesta z v do u. Hrana (u, v) pak uzavírá cyklus.

$\Leftarrow$: Nechť G obsahuje cyklus c. Ukážeme, že pak DFS(G) obsahuje zpětnou hranu.

- Nechť v je první uzel c objeven procedurou DFS(G) a nechť (u, v) je předcházející hrana na cyklu c.
- V čase $d[v]$ tvoří hrany cyklu c bílou cestu z v do u.
- Podle věty o bílé cestě platí, že u je následníkem v v DFS lese. Proto je (u, v) zpětná hrana.

□

Theorem 11. *Procedura* TOPOLOGICAL-SORT(G) *topologicky uspořádá orientovaný acyklický graf* G.

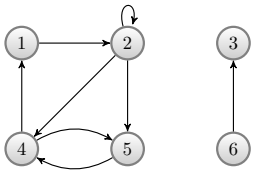
Důkaz.
- Nechť DFS je spuštěno na orientovaný acyklický graf $G = (V, E)$ a určí hodnoty $f[v]$.
- Stačí ukázat, že pokud $(u, v) \in E$, pak $f[v] < f[u]$.
- Nechť (u, v) je právě prozkoumávána procedurou DFS(G), pak v nemůže být šedý, protože jinak by byl v předchůdcem u a (u, v) by byla zpětná hrana – spor s předchozím lemmatem.
- v je bílý, pak v je následníkem u v DFS lese, a proto $f[v] < f[u]$.
- v je černý, pak $f[v]$ je již nastaveno. Jelikož u je stále prozkoumáváno, není $f[u]$ dosud nastaveno, proto $f[v] < f[u]$.

□

5.2 Silně souvislé komponenty (Strongly Connected Components)

Silně souvislé komponenty (SCC)

- Aplikace DFS
- $G = (V, E)$ orientovaný graf. Silně souvislá komponenta je maximální množina $C \subseteq V$ taková, že pro každé $u, v \in C$, $u \rightsquigarrow v$ (tedy i $v \rightsquigarrow u$).



Graf má tři silně souvislé komponenty:

- $\{1, 2, 4, 5\}$
- $\{3\}$
- $\{6\}$
- *Transponovaný graf* grafu $G = (V, E)$ je graf $G^T = (V, E^T)$, kde $E^T = \{(u, v) : (v, u) \in E\}$.

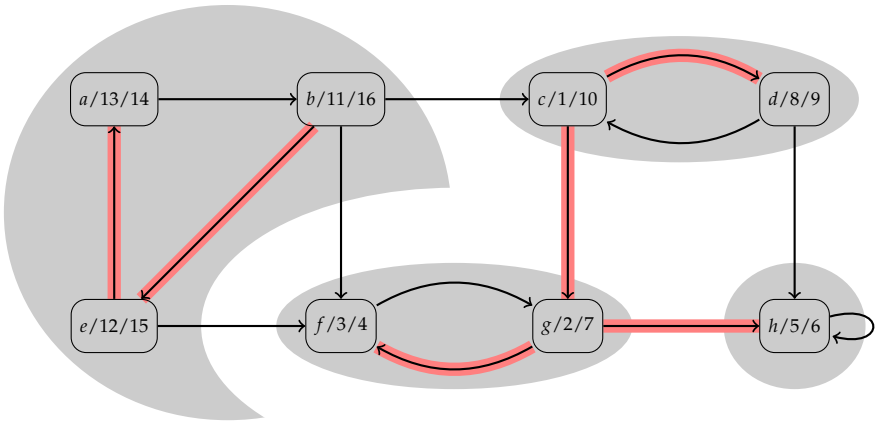
SCC(G)
1 zavolej DFS(G) pro výpočet hodnot $f[u]$
2 vypočítej G^T
3 zavolej DFS(G^T), ale v hlavním cyklu uvažuj uzly
v klesajícím pořadí podle hodnoty $f[u]$
4 na výstup dej uzly každého stromu z DFS lesa, určeného na řádku 3,
jako samostatnou silně souvislou komponentu

- Složitost $\Theta(m + n)$.
- Seznam sousedů G^T se dá určit ze seznam sousedů G v čase $O(m + n)$. Jak?
- G a G^T mají stejné silně souvislé komponenty – u a v jsou vzájemně dosažitelné v G, právě když jsou vzájemně dosažitelné v G^T.

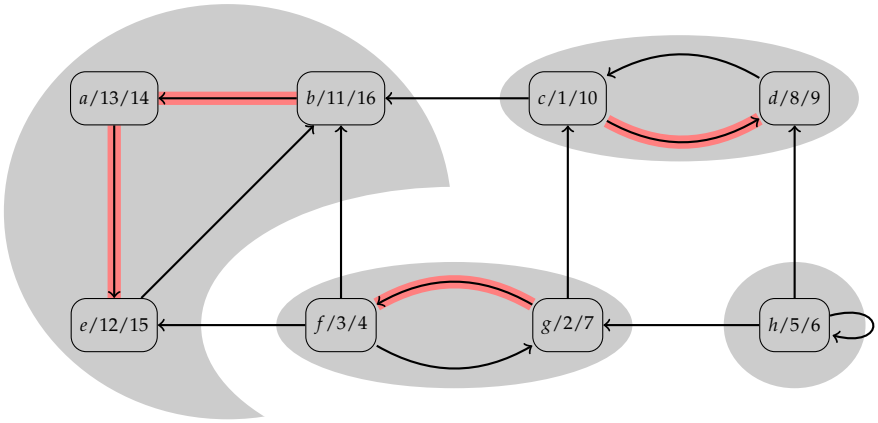
SCC – příklad

SCC – příklad

- *Graf komponent* grafu $G = (V, E)$ je graf $G^{scc} = (V^{scc}, E^{scc})$ definován následovně:
  - Nechť $C_1, C_2, \ldots, C_k$ jsou silně souvislé komponenty grafu G.
  - $V^{scc} = \{v_1, v_2, \ldots, v_k\} \subseteq V$, $V^{scc} \cap C_i \neq \emptyset$, $i = 1, 2, \ldots, k$.
  - $(v_i, v_j) \in E^{scc}$, pokud existují $x \in C_i$ a $y \in C_j$, $i \neq j$, takové, že $(x, y) \in E$.



Obrázek 11: Výsledek DFS(G). Vyznačeny stromové hrany a silně souvislé komponenty.



Obrázek 12: Graf G^T a výsledek řádku 3 procedury SCC. Uzly b, c, g a h jsou kořeny stromů DFS lesa. Každý strom odpovídá jedné silně souvislé komponentě.

Vlastnost grafu komponent

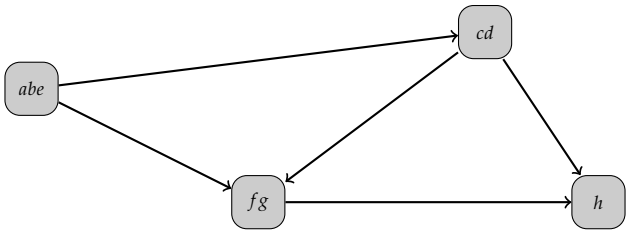
Lemma 12. *Nechť C a C' jsou různé silně souvislé komponenty orientovaného grafu $G = (V, E)$. Nechť $u, v \in C$, $u', v' \in C'$ a $u \rightsquigarrow u'$ v G. Pak NEplatí $v' \rightsquigarrow v$.*

Důkaz. Pokud $v' \rightsquigarrow v$, pak $u \rightsquigarrow u' \rightsquigarrow v'$ a $v' \rightsquigarrow v \rightsquigarrow u$, tj. u a v' jsou vzájemně dosažitelné – spor. □

- V následujícím uvažujeme pouze časy $d[u]$ a $f[u]$ vypočítané prvním voláním procedury DFS.
- $U \subseteq V$, pak $d(U) = \min_{u \in U}\{d[u]\}$ a $f(U) = \max_{u \in U}\{f[u]\}$.

Lemma 13. *Nechť C a C' jsou různé silně souvislé komponenty orientovaného grafu $G = (V, E)$. Nechť $(u, v) \in E$, $u \in C$, $v \in C'$. Pak $f(C) > f(C')$.*

Důkaz



- 1) $d(C) < d(C')$ – nechť x je první objevený v C. V čase $d[x]$ jsou všechy uzly $C \cup C'$ bílé. Pro $w \in C'$ existuje bílá cesta $x \rightsquigarrow u \rightarrow v \rightsquigarrow w$. Věta o bílé cestě říká, že všechy uzly z $C \cup C'$ jsou následníky x v DFS stromu. Důsledek závorkové věty říká, že $f[x] = f(C) > f(C')$.
- 2) $d(C) > d(C')$ – nechť y první objevený v C'. V čase $d[y]$ jsou všechy uzly C' bílé a existuje bílá cesta z y do každého uzlu C'. Věta o bílé cestě a důsledek závorkové věty dávají $f[y] = f(C')$. V čase $d[y]$ jsou všechy uzly C bílé. Z předchozího lemmatu máme, že neexistuje cesta z C' do C. Proto jsou uzly C bílé i v čase $f[y]$, tj. $f[w] > f[y]$, $w \in C$, což dává $f(C) > f(C')$.

Corollary 14. *Nechť C a C' jsou různé silně souvislé komponenty orientovaného grafu $G = (V, E)$. Nechť $(u, v) \in E^T$, $u \in C$, $v \in C'$. Pak $f(C) < f(C')$.*

Důkaz. $(u, v) \in E^T$ implikuje, že $(v, u) \in E$. Protože silně souvislé komponenty G a G^T jsou stejné, dává předchozí lemma $f(C) < f(C')$. □

Theorem 15. *Procedura* SCC(G) *je korektní.*

Důkaz

- Indukcí vzhledem k počtu DFS stromů nalezených na řádku 3 procedury. IP: Prvních k stromů nalezených procedurou na řádku 3 jsou silně souvislé komponenty. ZK: $k = 0$ je triviální.
- IK: Uvažme $(k+1)$-ní nalezený strom. Nechť u je jeho kořen a nechť u je v silně souvislé komponentě C.
- $f[u] = f(C) > f(C')$ pro libovolnou silně souvislou komponentu C' různou od C, která ještě nebyla navštívena.
- Podle IP jsou v čase $d_{G^T}[u]$ všechny uzly z C bílé. Věta o bílé cestě dává, že všechny ostatní uzly C jsou následníky u v DFS stromu.
- Podle IP a předchozího důsledku vede každá hrana v G^T jdoucí z C do již navštívené silně souvislé komponenty.
- Tedy žádný uzel z jiné komponenty než C nebude následníkem u během DFS G^T. Uzly stromu tedy tvoří silně souvislou komponentu.

6 Minimální kostry

Minimální kostra

- První algoritmus řešící tento problém navrhl brněnský matematik O. Borůvka, 1926.
- Nechť $G = (V, E)$ je souvislý neorientovaný graf s váhovou funkcí

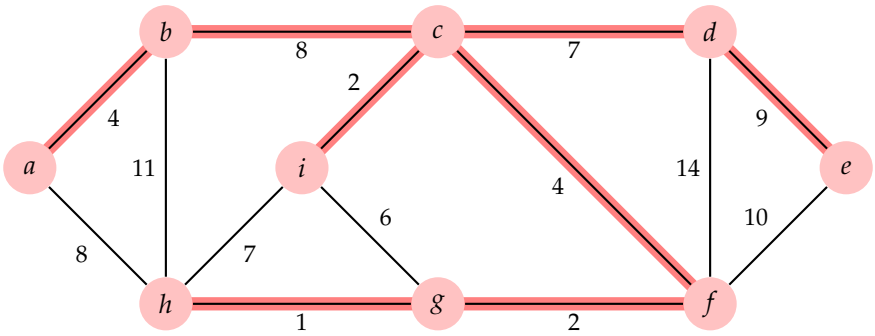
$$w : E \to \mathbb{R}\,.$$

- Úkolem je najít takovou množinu hran $T \subseteq E$, že podgraf (V, T) je souvislý, acyklický a

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

je minimální.

Příklad



Generický algoritmus

```
GENERIC-MST(G, w)
1  A ← ∅
2  while A netvoří kostru
3        do najdi hranu (u, v) ∈ E bezpečnou pro A
4           A ← A ∪ {(u, v)}
5  return A
```

- Invariant: Před každou iterací algoritmu je množina A podmnožinou nějaké minimální kostry.
- Hrana $(u, v) \in E$ je bezpečná pro A, pokud $A \cup \{(u, v)\}$ je podmnožinou nějaké minimální kostry.

Pomocné definice

- *Řez* grafu $G = (V, E)$ je dvojice $(S, V - S)$, $S \subseteq V$.
- Hrana $(u, v) \in E$ *kříží* řez $(S, V - S)$, pokud jeden její konec je v S a druhý ve $V - S$.
- Řez *respektuje* množinu A hran, pokud žádná hrana v A nekříží řez.

- Hrana se nazývá *lehká*, pokud kříží řez a její hodnota je minimální z hodnot všech hran, které kříží řez.

Theorem 16. • *Nechť $G = (V, E)$ je souvislý neorientovaný graf s reálnou váhovou funkcí w.*
- *Nechť $A \subseteq E$ je součástí nějaké minimální kostry G.*
- *Nechť $(S, V - S)$ je řez, který respektuje A.*
- *Nechť (u, v) je lehká hrana křížící $(S, V - S)$.*

Pak hrana (u, v) je bezpečná pro A.

Důkaz

- Nechť T je minimální kostra G, $A \subseteq T$, $(u, v) \notin T$.
- $u \rightsquigarrow v$ je cesta v T, tj. přidání (u, v) vytváří kružnici. Nechť např. $u \in S$ a $v \in V - S$.
- Nechť (x, y) je hrana z cesty $u \rightsquigarrow v$ v T křížící řez $(S, V - S)$. Protože řez respektuje A, $(x, y) \notin A$.
- $T' = (T - \{(x, y)\}) \cup \{(u, v)\}$ je kostra grafu G.

Důkaz. • (u, v) je lehká hrana křížící $(S, V - S)$ a (x, y) rovněž kříží řez, tedy $w(u, v) \leq w(x, y)$.
- Tedy, $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$.
- T je minimální kostra, proto $w(T) \leq w(T')$.
- Protože $A \subseteq T$ a $(x, y) \notin A$, $A \subseteq T'$.
- Konečně, $A \cup \{(u, v)\} \subseteq T'$. Protože T' je minimální kostra, je (u, v) bezpečná pro A.

□

Corollary 17. • *Nechť $G = (V, E)$ je souvislý neorientovaný graf s reálnou váhovou funkcí w.*
- *Nechť $A \subseteq E$ je součástí nějaké minimální kostry G.*
- *Nechť $C = (V_C, E_C)$ je souvislá komponenta (strom) v lese $G_A = (V, A)$.*

Pokud (u, v) je lehká hrana spojující C s jinou komponentou z G_A, pak (u, v) je bezpečná pro A.

Důkaz. Řez $(V_C, V - V_C)$ respektuje A a (u, v) je lehká hrana tohoto řezu. Proto je (u, v) bezpečná pro A. □

Kruskalův a Primův (Jarníkův) algoritmus

- Založeny na generickém algoritmu.
- Udávají pravidlo vybírající bezpečnou hranu, viz řádek 3 generického algoritmu.
- V Kruskalově algoritmu je A les a bezpečná hrana přidávaná do A je hrana s nejmenším ohodnocením spojující dvě různé komponenty.
- V Primově (Jarníkově) algoritmu je A strom a bezpečná hrana přidávaná do A je hrana s nejmenším ohodnocením spojující strom s uzlem, který není součástí stromu.

Kruskalův algoritmus

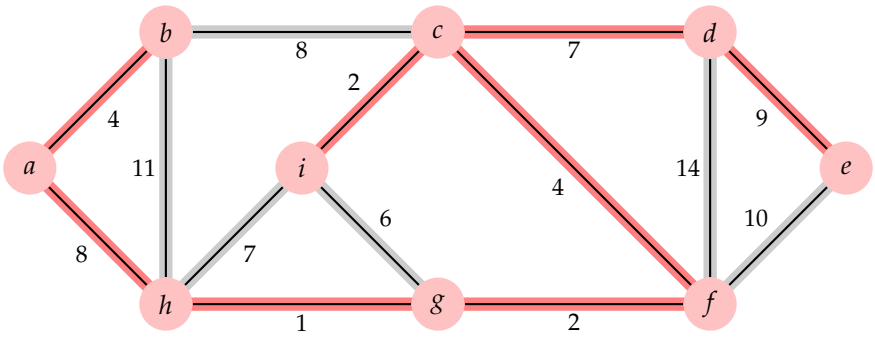
- MAKE-SET(v) vytvoří množinu obsahující v.
- FIND-SET(v) vrací reprezentanta množiny obsahující v.
- UNION(u, v) sjednotí dvě množiny obsahující u a v.

KRUSKAL-MST(G, w)
1 $A \leftarrow \emptyset$
2 **for** každý uzel $v \in V$
3 **do** MAKE-SET(v)
4 uspořádej hrany E do neklesající posloupnosti podle váhy w
5 **for** každou hranu $(u, v) \in E$ branou v neklesajícím uspořádání podle w
6 **do if** FIND-SET$(u) \neq$ FIND-SET(v)
7 **then** $A \leftarrow A \cup \{(u, v)\}$
8 UNION(u, v)
9 **return** A

Kruskalův algoritmus – Složitost

- Řádek 1: $O(1)$, Řádek 4: $O(m \log m)$. Řádky 2-3: n-krát složitost MAKE-SET. Řádky 5-8: $O(m)$-krát FIND-SET a UNION – závisí na implementaci
  - Implementace seznamem s heuristikou: celkem $O(m + n \log n)$.
  - Stromová implementace s váhami a zkratkami: celkem $O((m + n)\alpha(n))$, α je velmi pomalu rostoucí funkce ($\alpha(n) \leq 4$).
- G souvislý dává $m \geq n - 1$. Proto množinové operace berou $O(m\alpha(n))$. Protože $\alpha(n) = O(\log n) = O(\log m)$, celková složitost je $O(m \log m)$.
- Když si ještě všimneme, že $m < n^2$, je $\log m = O(\log n)$, proto celkem $O(m \log n)$.

Kruskalův algoritmus – příklad

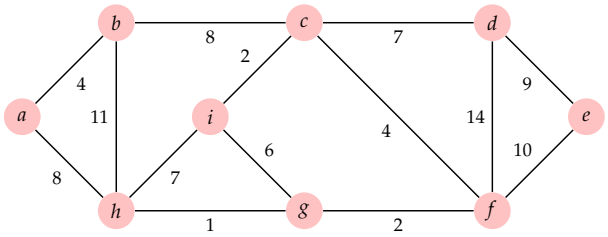


Primův algoritmus

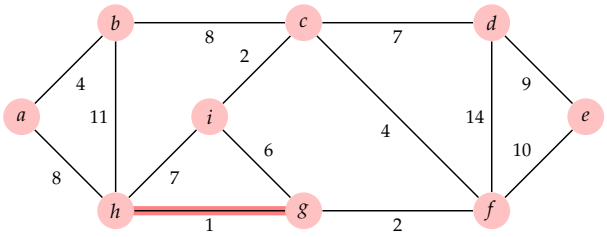
Invariant:

- $A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$.
- v v min. kostře, pak $v \in V - Q$.
- $v \in Q$. Pokud $\pi[v] \neq NIL$, pak $key[v] < \infty$ a $key[v]$ je hodnota lehké hrany $(v, \pi[v])$ spojující v s uzlem již v min. kostře.

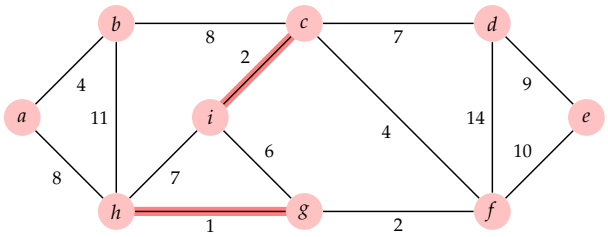
Kruskalův algoritmus – příklad



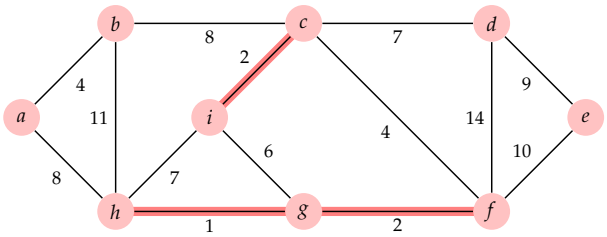
Kruskalův algoritmus – příklad



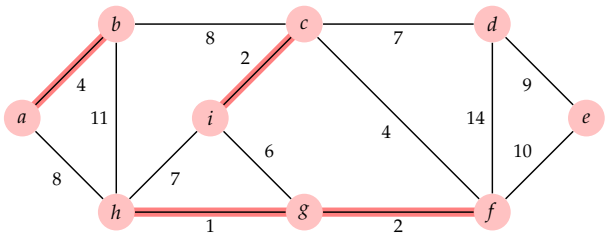
Kruskalův algoritmus – příklad



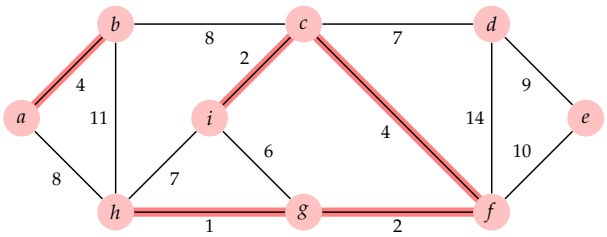
Kruskalův algoritmus – příklad



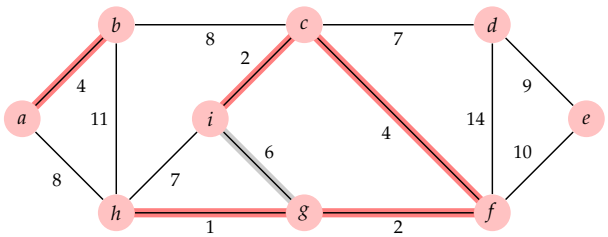
Kruskalův algoritmus – příklad



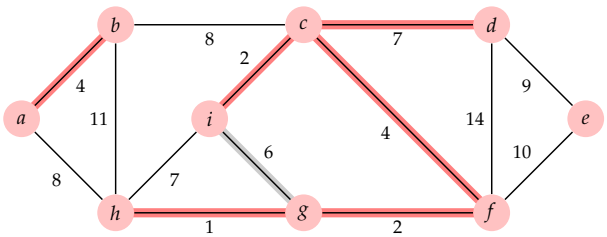
Kruskalův algoritmus – příklad



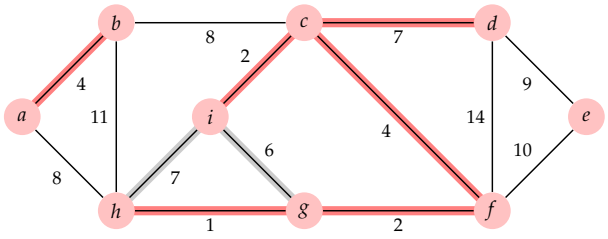
Kruskalův algoritmus – příklad



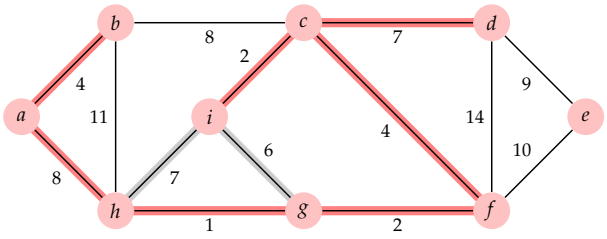
Kruskalův algoritmus – příklad



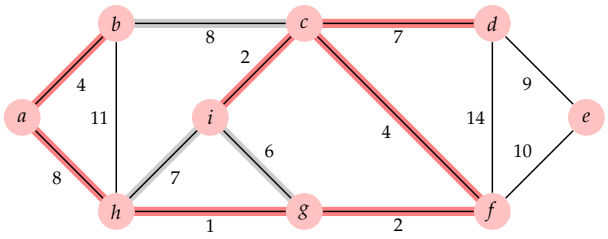
Kruskalův algoritmus – příklad



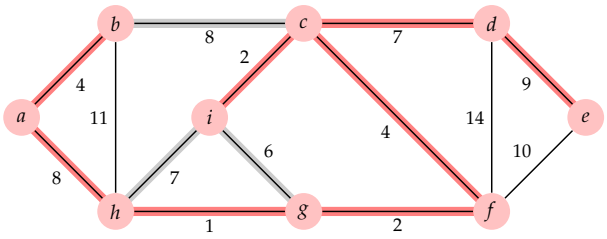
Kruskalův algoritmus – příklad



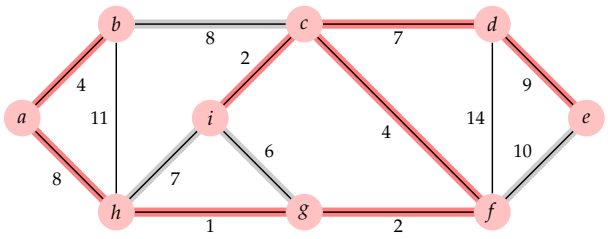
Kruskalův algoritmus – příklad



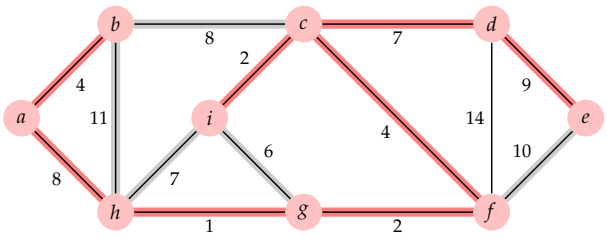
Kruskalův algoritmus – příklad



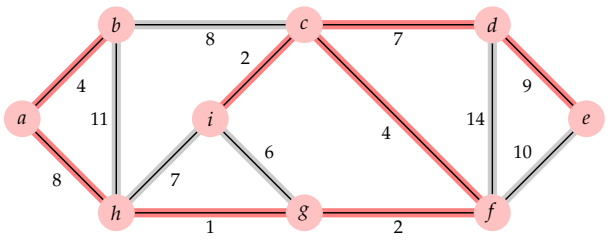
Kruskalův algoritmus – příklad



Kruskalův algoritmus – příklad



Kruskalův algoritmus – příklad



```
PRIM-MST(G, w, r)
1  for každý u ∈ V
2      do key[u] ← ∞
3         π[u] ← NIL
4  key[r] ← 0
5  Q ← V
6  while Q ≠ ∅
7        do u ← EXTRACT-MIN(Q)
8           for každý v ∈ Adj[u]
9               do if v ∈ Q a w(u, v) < key[v]
10                    then π[v] ← u
11                         key[v] ← w(u, v)
```

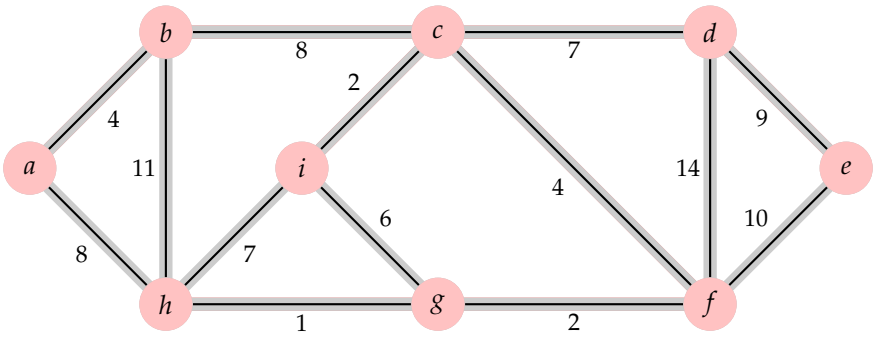
Primův algoritmus – Složitost

- Řádky 1-5: $O(n)$ za použití binární haldy.
- **while** cyklus se proved n-krát a protože EXTRACT-MIN bere $O(\log n)$, je celková složitost všech volání EXTRACT-MIN $O(n \log n)$.
- **for** cyklus se provede $O(m)$-krát, protože délka všech seznamů sousedů je dohromady $2m$.
- Řádek 9 se dá udělat v čase $O(1)$.
- Řádek 11 bere $O(\log n)$ – provést operaci DECREASE-KEY v Q.
- Celkem tedy $O(n \log n + m \log n) = O(m \log n)$.

Primův algoritmus – Složitost

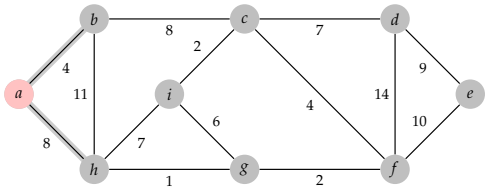
- Použitím Fibonacciho haldy se dá složitost vylepšit.
- Operace EXTRACT-MIN v čase $O(\log n)$
- Operace DECREASE-KEY v čase $O(1)$.
- Celková složitost je pak $O(m + n \log n)$.

Primův algoritmus – příklad



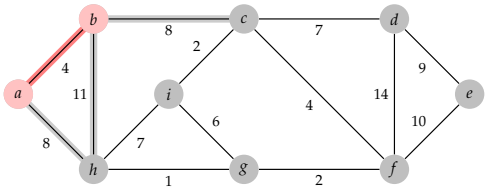
Obrázek 13: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



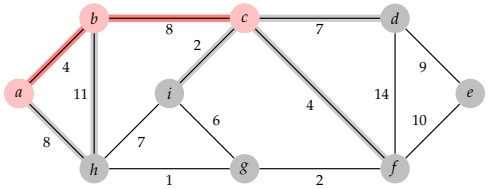
Obrázek: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



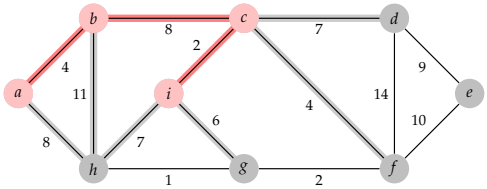
Obrázek: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



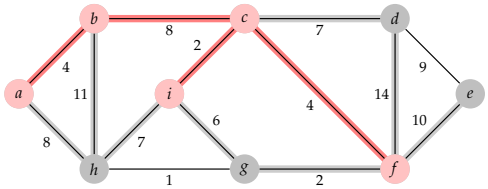
Obrázek: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



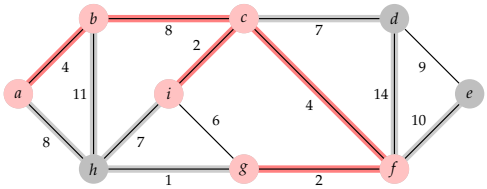
Obrázek: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



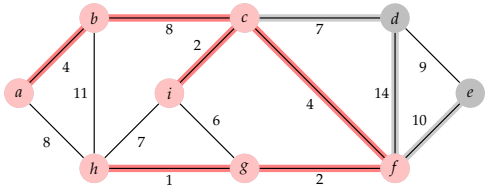
Obrázek: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



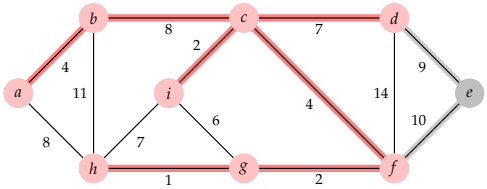
Obrázek: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



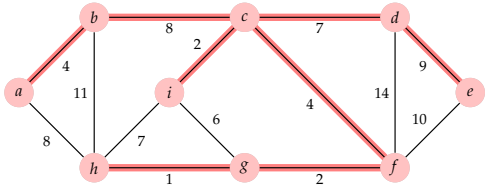
Obrázek: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



Obrázek: Šedé hrany jsou hrany řezu.

Primův algoritmus – příklad



Obrázek: Šedé hrany jsou hrany řezu.

7 Nejkratší cesty z jednoho do všech uzlů

Nejkratší cesty

- Daný ohodnocený orientovaný graf $G = (V, E)$ a
- váhová funkce $w : E \to \mathbb{R}$.
- Cena cesty $p = \langle v_0, v_1, \ldots, v_k \rangle$ je suma

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

- Cena nejkratší cesty z u do v je

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{pokud ex. cesta z } u \text{ do } v \\ \infty & \text{jinak} \end{cases}$$

- Nejkratší cesta z u do v je pak libovolná cesta p z u do v s $w(p) = \delta(u, v)$.

Nejkratší cesty – varianty

- Nejkratší cesty z jednoho do všech uzlů
- Nejkratší cesty ze všech uzlů do jednoho – převrácením orientace hran
- Z jednoho do jednoho – existuje asymptoticky rychlejší řešení než pomocí výše zmíněných?
- Ze všech do všech – existuje rychlejší řešení než výše zmíněné spuštěné pro každý uzel.

Podcesty nejkratších cest

Lemma 18. *Nechť $G = (V, E)$ je ohodnocený orientovaný graf s váhovou funkcí $w : E \to \mathbb{R}$. Nechť $p = \langle v_1, v_2, \ldots, v_k \rangle$ je nejkratší cesta z v_1 do v_k. Pro $1 \leq i \leq j \leq k$, $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ je podcesta cesty p z v_i do v_j. Pak p_{ij} je nejkratší cesta z v_i do v_j.*

Důkaz.

- p je $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, kde $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$.
- Nechť ex. p'_{ij} z v_i do v_j s $w(p'_{ij}) < w(p_{ij})$.
- Pak $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, kde $w(p_{1i}) + w(p'_{ij}) + w(p_{jk}) < w(p)$. Spor.

□

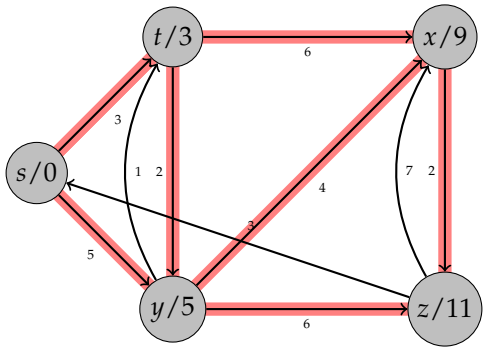
Záporné hrany

- Pokud G neobsahuje záporný cyklus dosažitelný ze zdrojového uzlu s, pak pro všechna $v \in V$, $\delta(s, v)$ zůstává dobře definována (i když má zápornou hodnotu).
- Pokud G obsahuje záporný cyklus dosažitelný ze zdrojového uzlu s, pak δ nezůstává dobře definována – stále prochází cyklem a snižuje hodnotu.
- Pokud ex. záporný cyklus na nějaké cestě z s do v, definujeme $\delta(s, v) = -\infty$.
- Pozn. nejkratší cesta vždy existuje, ne však nejkratší sled. Algoritmy pracují se sledy, proto výše zmíněný problém.

Reprezentace nejkratších cest

- $G = (V, E)$ graf.
- $\pi[v]$ označuje předchůdce v na nejkratší cestě.
- Pro vypsání můžeme použít proceduru PRINT-PATH(G, s, v) (viz dříve)
- Podgraf předchůdců $G_\pi = (V_\pi, E_\pi)$ je indukovaný hodnotami π
  - $V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$
- V okamžiku dokončení výpočtu algoritmu je G_π strom nejkratších cest. Tj. kořenový strom obsahující nejkratší cesty ze zdroje s do všech ostatních uzlů.

Nejedinečnost nejkratších cest – příklad



Obrázek 14: Nejkratší cesty.

Relaxace

- $d[v]$ – odhad nejkratší cesty

INITIALIZE-SINGLE-SOURCE(G, s)
1 **for** každý $v \in V$
2 **do** $d[v] \leftarrow \infty$
3 $\pi[v] \leftarrow$ NIL
4 $d[s] \leftarrow 0$

- Složitost $\Theta(n)$.

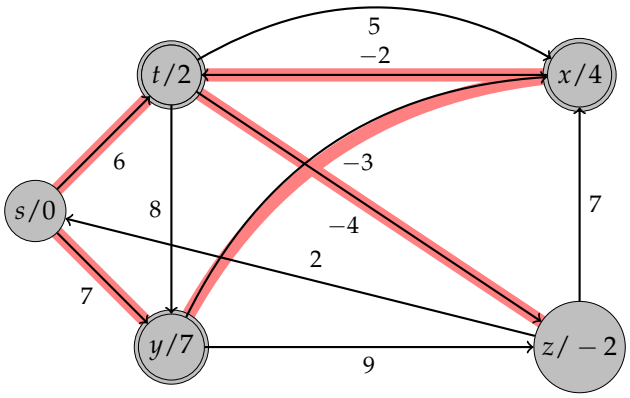
RELAX(u, v, w)
1 **if** $d[v] > d[u] + w(u, v)$
2 **then** $d[v] \leftarrow d[u] + w(u, v)$
3 $\pi[v] \leftarrow u$

Algoritmus Bellman-Ford

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to n − 1
3       do for každou hranu (u, v) ∈ E
4               do RELAX(u, v, w)
5  for každou hranu (u, v) ∈ E
6       do if d[v] > d[u] + w(u, v)
7             then return FALSE
8  return TRUE
```

- Pokud vrátí FALSE, G obsahuje zápornou kružnici.
- Pokud vrátí TRUE, má v π uloženy nejkratší cesty.

Bellman-Ford – příklad



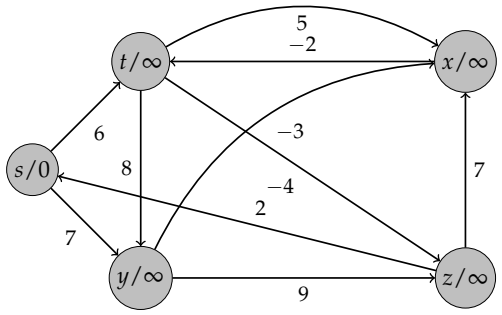
Obrázek 15: Práce algoritmu Bellman-Ford.

- Pokud $(u, v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí: $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

Algoritmus Bellman-Ford – Složitost

- Řádek 1 bere $\Theta(n)$.
- Řádky 2-4 berou $n - 1$-krát $\Theta(m)$.
- Řádky 5-7 berou $\Theta(m)$.
- Celkem $\Theta(mn)$.

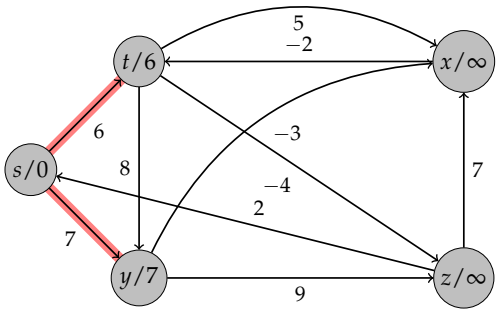
Bellman-Ford – příklad



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

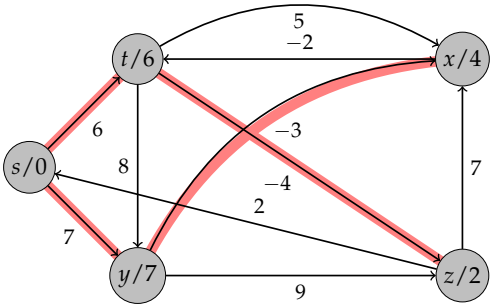
Bellman-Ford – příklad



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

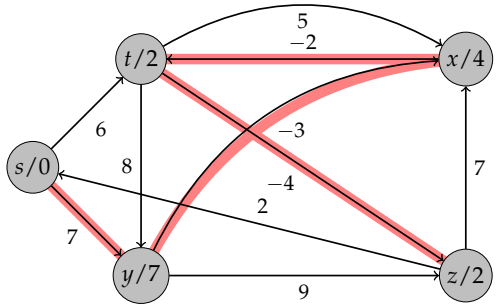
Bellman-Ford – příklad



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

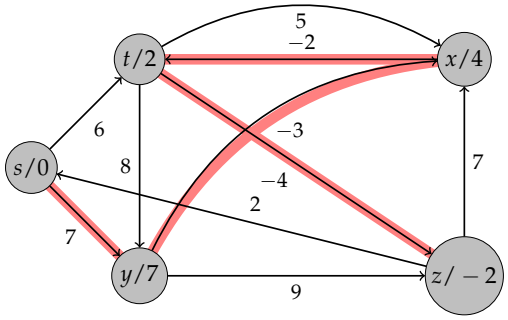
Bellman-Ford – příklad



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Bellman-Ford – příklad



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Algoritmus Bellman-Ford – Korektnost

Lemma 19. *Nechť $G = (V, E)$ je ohodnocený orientovaný graf se zdrojem s a váhovou funkcí $w : E \to \mathbb{R}$ a předpokládejme, že G neobsahuje žádný záporný cyklus dosažitelný z s. Pak po $n - 1$ opakování* **for***-cyklu na řádcích 2-4 $d[v] = \delta(s, v)$ pro všechny $v \in V$ dosažitelné z s.* **Pozn.** *$d[v] = \infty$, pak v nedosažitelný z s.*

Důkaz.

- Nechť $v \in V$ dosažitelný z s.
- Nechť $p = \langle v_0, v_1, \ldots, v_k \rangle$ je nejkratší cesta z s do v; $s = v_0$ a $v = v_k$.
- p má nejvýše $n - 1$ hran, proto $k \leq n - 1$.
- Každá z $n - 1$ iterací na řádcích 2-4 relaxuje všech m hran.
- Mezi hranami relaxovanými v i-tém kroku je i hrana (v_{i-1}, v_i) a po tomto kroku platí $d[v_i] = \delta(s, v_i)$. (Dokažte indukcí.)
- Tedy po k-té iteraci je $d[v_k] = \delta(s, v_k)$.

□

Algoritmus Bellman-Ford – Korektnost

Theorem 20 (Korektnost I)**.** *Pokud G neobsahuje záporný cyklus dosažitelný z s, algoritmus vrací* TRUE *a $d[v] = \delta(s, v)$ pro všechny $v \in V$.*

Důkaz.

- Nechť G neobsahuje záporný cyklus dosažitelný z s.
- Po ukončení algoritmu je $d[v] = \delta(s, v)$ pro všechny $v \in V$ (Lemma 19)
- Navíc, $d[v] = \delta(s, v) \leq \delta(s, u) + w(u, v) = d[u] + w(u, v)$. Proto algoritmus vrací TRUE.

□

Algoritmus Bellman-Ford – Korektnost

Theorem 21 (Korektnost II)**.** *Pokud G obsahuje záp. cyklus dosažitelný z s, alg. vrací* FALSE*.*

Důkaz.

- Záporný cyklus $c = \langle v_0, v_1, \ldots, v_k \rangle$, $v_0 = v_k$, dosažitelný z s.
- Pak $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$.
- Sporem – alg. vrací TRUE, tj. $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$ pro $i = 1, 2, \ldots, k$.
- Pak ale $\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$.
- Protože $v_0 = v_k$, máme $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$.
- Jelikož pro $i = 1, 2, \ldots, k$ je $d[v_i] < \infty$, máme $0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$. Spor.

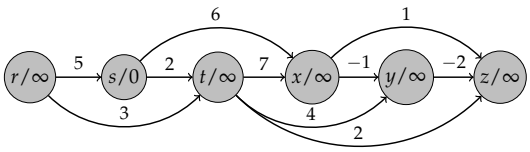
□

7.1 Nejkratší cesty z jednoho do všech uzlů v acyklických grafech

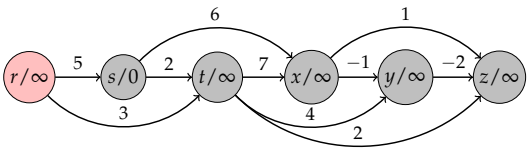
Nejkratší cesty v acyklických grafech

- Pro acyklické grafy existuje rychlejší metoda než Bellman-Ford.
- Časová složitost $\Theta(n + m)$.
  - Časová složitost topologického uspořádání je $\Theta(n + m)$.
  - Řádek dva má složitost $\Theta(n)$.
  - Řádky 3-5 projdou každou hranu právě jednou, tj. vnitřní cyklus se provede m-krát. RELAX je konstantní.

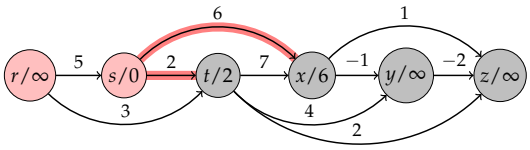
Příklad



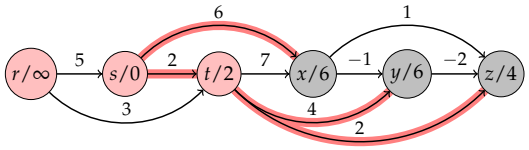
Příklad



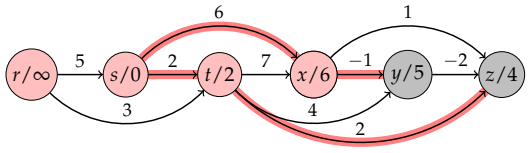
Příklad



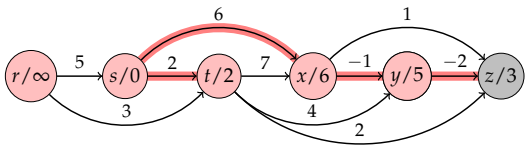
Příklad



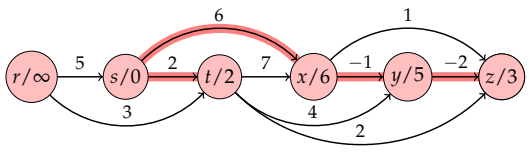
Příklad



Příklad

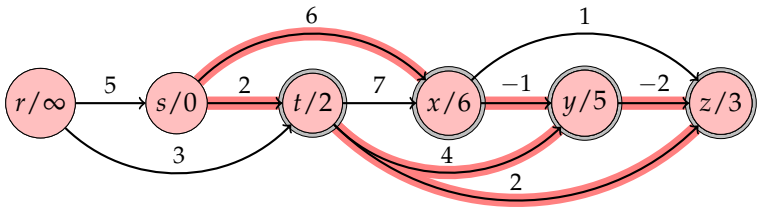


Příklad



DAG-SHORTEST-PATHS(G, w, s)
1 Topologicky uspořádej uzly grafu G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 **for** každý uzel u, brané podle topologického uspořádání
4 **do for** každý uzel $v \in Adj[u]$
5 **do** RELAX(u, v, w)

Příklad



Korektnost

Theorem 22. *Pokud ohodnocený orientovaný graf $G = (V, E)$ má zdrojový uzel s a žádný cyklus, pak* DAG-SHORTEST-PATHS *procedura vypočítá $d[v] = \delta(s, v)$ pro $v \in V$.*

Důkaz.

- Pokud v nedosažitelný z s, pak $d[v] = \delta(s, v) = \infty$.
- Nechť $p = \langle v_0, v_1, \ldots, v_k \rangle$, kde $s = v_0$ a $v = v_k$.
- Jelikož algoritmus prochází uzly podle topologického uspořádání, jsou hrany p relaxovány v pořadí $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$.
- Z toho plyne, že $d[v_i] = \delta(s, v_i)$ po ukončení algoritmu (dokažte).

□

Dijkstrův algoritmus

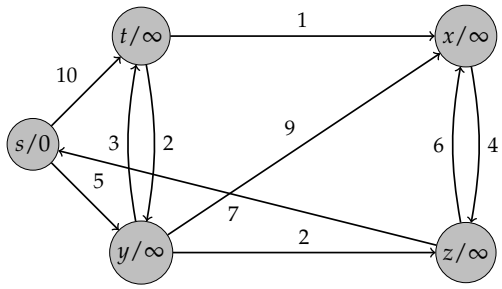
- Pouze pro ohodnocené orientované grafy bez záporných ohodnocení.
- $w(u, v) \geq 0$ pro každou hranu $(u, v) \in E$.
- Je možno jej naimplementovat tak, že jeho časová složitost je nižší než algoritmu Bellman-Ford.

Dijkstrův algoritmus

- S je množina uzlů, jejichž nejkratší vzdálenost od s již byla vypočtena.
- Q je prioritní fronta; uzel s min. d-hodnotou na vrcholu fronty.

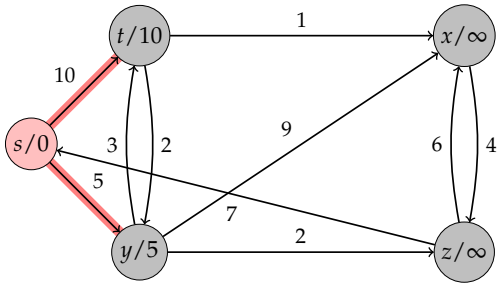
Dijkstrův algoritmus – příklad

Dijkstrův algoritmus – příklad



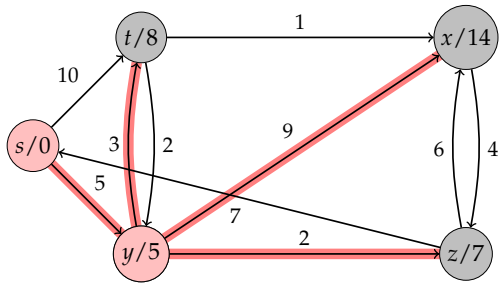
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Dijkstrův algoritmus – příklad



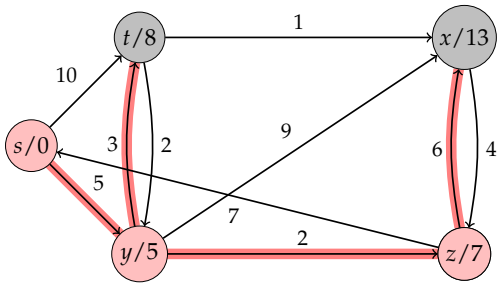
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Dijkstrův algoritmus – příklad



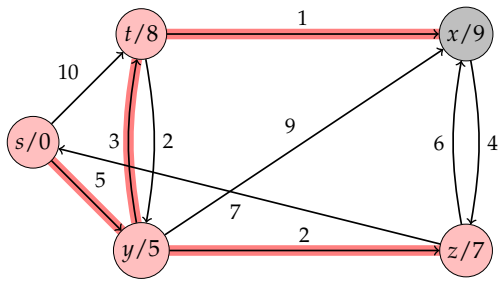
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Dijkstrův algoritmus – příklad



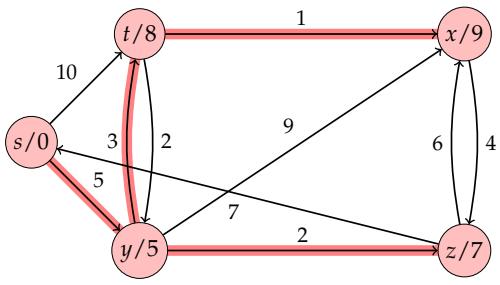
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Dijkstrův algoritmus – příklad



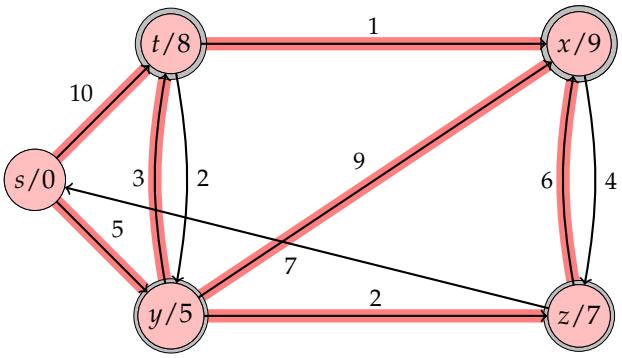
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Dijkstrův algoritmus – příklad



Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V
4  while Q ≠ ∅
5        do u ← EXTRACT-MIN(Q)
6           S ← S ∪ {u}
7           for každý uzel v ∈ Adj[u]
8              do RELAX(u, v, w)
```



Obrázek 16: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Korektnost

Theorem 23. *Dijkstrův algoritmus, spuštěn na ohodnoceném orientovaném grafu $G = (V, E)$ bez záporných hran se zdrojem s, skončí s $d[v] = \delta(s, v)$ pro $v \in V$.*

Důkaz.

- Invariant: Na začátku každého **while**-cyklu je $d[v] = \delta(s, v)$ pro $v \in S$.
- Platí pro $S = \emptyset$.
- Nechť u je první uzel, pro nějž $d[u] \neq \delta(s, u)$ v okamžiku přidání do S.
- Pak musí být $u \neq s$, protože s je přidaný jako první do S a $d[s] = \delta(s, s) = 0$ platí v okamžiku přidání s do S.
- Protože $u \neq s$, je $S \neq \emptyset$ (těsně) před přidáním u.
- Z předpokladu $d[u] \neq \delta(s, u)$ plyne existence cesty z s do u – jinak je $d[u] = \delta(s, u) = \infty$.
- Existuje tedy nejkratší cesta p z s do u.

□

Korektnost

Pokračování důkazu.

- Existuje tedy nejkratší cesta p z s do u.
- Těsně před přidáním u do S spojuje p uzel $s \in S$ s uzlem $u \in V - S$.
- Rozložme p následovně:

$$s \overset{p_1}{\rightsquigarrow} x \to y \overset{p_2}{\rightsquigarrow} u\,,$$

kde y je první uzel na cestě, který leží ve $V - S$ a x je jeho předchůdce na p.

- Podle přepokladu máme, že $d[x] = \delta(s, x)$ v okamžiku přidání x do S.
- Jelikož v tomto okamžiku byla hrana (x, y) relaxována, $d[y] = \delta(s, y)$ v okamžiku přidání u do S (dokažte).

□

Korektnost

Pokračování důkazu.

- $s \overset{p_1}{\rightsquigarrow} x \to y \overset{p_2}{\rightsquigarrow} u$, kde y je první uzel ležící ve $V - S$ a x je jeho předchůdce na p.
- $d[y] = \delta(s, y)$ v okamžiku přidání u do S.
- Jelikož y leží před u na nejkratší cestě z s do u a ohodnocení hran je nezáporné, máme $\delta(s, y) \leq \delta(s, u)$.
- Tedy, $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$.
- Protože však oba uzly $y, u \in V - S$ v okamžiku, kdy u byl vybrán, je $d[u] \leq d[y]$.
- Celkem tedy $d[u] = \delta(s, u)$. Spor – špatný předpoklad.
- V okamžiku ukončení je $Q = \emptyset$. Jelikož $Q = V - S$ (rozmyslete si), je $S = V$. Proto $d[v] = \delta(s, v)$ pro $v \in V$.
- Hotovo – uff....

□

Složitost Dijkstrova algoritmu Prioritní fronta implementována pomocí pole

- INSERT a DECREASE-KEY čas $O(1)$.
- EXTRACT-MIN čas $O(n)$, provede se pro každý uzel (řádek 5).
- RELAX na řádku 8 se provede m-krát.
- Celkem $O(n^2 + m) = O(n^2)$.
- Pro řídké grafy, $|E| = o(n^2 / \log n)$, lze dostat časovou složitost $O(m \log n)$ (pomocí binární haldy).
- Obecně, použitím Fibonacciho haldy dostaneme časovou složitost $O(n \log n + m)$.

8 Nejkratší cesty ze všech uzlů do všech ostatních uzlů

Nejkratší cesty

- Daný ohodnocený orientovaný graf $G = (V, E)$ a
- váhová funkce $w : E \to \mathbb{R}$.

- Možno n-krát použít algoritmus pro nalezení nejkratší cesty z daného uzlu do všech ostatních.
- Dijkstrův algoritmus: Čas $O(n^3 + nm) = O(n^3)$ pro pole, či $O(n^2 \log n + nm)$ pro Fibonacciho haldu.
- Pokud povolíme záporné hrany, musíme použít algoritmus Bellman-Ford, tj. čas $O(n^2 m)$, což je na hustých grafech $O(n^4)$.

Nejkratší cesty

- Na rozdíl od předchozí části zde používáme matici sousednosti $W = (w_{ij})$, kde

$$w_{ij} = \begin{cases} 0 & \text{pro } i = j, \\ w(i,j) & \text{pro } i \neq j \text{ a } (i,j) \in E, \\ \infty & \text{pro } j \neq j \text{ a } (i,j) \notin E \end{cases}$$

- Připouštíme záporné hrany.
- Zatím se omezíme na grafy bez záporných cyklů.

- Výsledek v matici $D = (d_{ij})$, kde $d_{ij} = \delta(i,j)$ po ukončení algoritmu.
- Matice předchůdců $\Pi = (\pi_{ij})$, kde π_{ij} je
  1. NIL, pokud $i = j$ nebo neexistuje cesta z i do j,
  2. předchůdce j na nějaké nejkratší cestě z i.

Výpis nejkratších cest

```
PRINT-ALL-SHORTEST-PATH(Π, i, j)
1  if i = j
2     then print i
3     else if π_ij = NIL
4             then print "Cesta z" i "do" j "neexistuje!"
5             else PRINT-ALL-SHORTEST-PATH(Π, i, π_ij)
6                  print j
```

Násobení matic – struktura nejkratších cest

- Reprezentace – matice sousednosti $W = (w_{ij})$.
- Nechť p je nejkratší cesta z i do j, která má m' hran.
- Pokud p nemá záporný cyklus, pak $m' < \infty$.
- Pro $i = j$ je $m' = 0$ a $w_{ij} = \delta(i, j) = 0$.
- Pro $i \neq j$ rozložme cestu p takto:

  $$i \overset{p'}{\rightsquigarrow} k \to j\,,$$

  kde p' má $m' - 1$ hran.
- p' je nejkratší cesta z i do k – rozmyslete – proto $\delta(i, j) = \delta(i, k) + w_{kj}$.

Násobení matic – rekurze

- Nechť $l_{ij}^{(m)}$ je minimální ohodnocení ze všech cest z i do j, které obsahují nejvýše m hran.
- $m = 0$, právě když $i = j$. Tedy $l_{ij}^{(0)} = \begin{cases} 0 & \text{pro } i = j \\ \infty & \text{pro } i \neq j \end{cases}$
- $l_{ij}^{(m)} = \min(l_{ij}^{(m-1)}, \min\limits_{1 \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\}) = \min\limits_{1 \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\}$.
- Nejkratší cesta z i do j má nejvýše $n - 1$ hran, proto

  $$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \ldots.$$

  (Když tam není záporný cyklus.)

Násobení matic – výpočet

- Vstupní matice $W = (w_{ij})$.
- Vypočteme matice $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$, kde pro $m = 1, 2, \ldots, n - 1$,

  $$L^{(m)} = (l_{ij}^{(m)})\,.$$

- $L^{(n-1)}$ pak obsahuje hodnoty nejkratších cest.
- $l_{ij}^{(1)} = w_{ij}$, tj. $L^{(1)} = W$.

Srdce algoritmu

- $rows[L]$ značí počet řádků L.
- Časová složitost $\Theta(n^3)$.

```
EXTEND-SHORTEST-PATHS(L, W)
1  n ← rows[L]
2  Nechť L' = (l'_ij) je matice řádu n
3  for i ← 1 to n
4      do for j ← 1 to n
5             do l'_ij ← ∞
6                for k ← 1 to n
7                    do l'_ij ← min(l'_ij, l_ik + w_kj)
8  return L'
```

Konečně souvislost s násobením matic

- Nechť $C = A \cdot B$, kde A a B jsou matice řádu n.
- Pak
$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$
- Pro srovnání
$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{l_{ik}^{(m-1)} + w_{kj}\}$$

Najděte 4 rozdíly (mimo názvu)

```
EXTEND-SHORTEST-PATHS(L, W)
1  n ← rows[L]
2  Nechť L' = (l'_ij) je matice řádu n
3  for i ← 1 to n
4      do for j ← 1 to n
5             do l'_ij ← ∞
6                for k ← 1 to n
7                    do l'_ij ← min(l'_ij, l_ik + w_kj)
8  return L'
```

```
MATRIX-MULTIPLY(A, B)
1  n ← rows[A]
2  Nechť C je matice řádu n
3  for i ← 1 to n
4      do for j ← 1 to n
5             do c_ij ← 0
6                for k ← 1 to n
7                    do c_ij ← c_ij + a_ik · b_kj
8  return C
```

Zase násobení matic

- Zápisem $X \cdot Y$ označme matici vypočtenou procedurou EXTEND-SHORTEST-PATHS(X, Y).

- Pak počítáme sekvenci matic

$$\begin{array}{rcccc} L^{(1)} & = & L^{(0)} \cdot W & = & W \\ L^{(2)} & = & L^{(1)} \cdot W & = & W^2 \\ L^{(3)} & = & L^{(2)} \cdot W & = & W^3 \\ & & \vdots & & \\ L^{(n-1)} & = & L^{(n-2)} \cdot W & = & W^{n-1} \end{array}$$

  kde W^{n-1} obsahuje hodnoty nejkratších cest.

Pomalá metoda

```
SLOW-ALL-SHORTEST-PATHS(W)
1  n ← rows[W]
2  L^(1) ← W
3  for m ← 2 to n − 1
4       do L^(m) ← EXTEND-SHORTEST-PATHS(L^(m−1), W)
5  return L^(n−1)
```

- Složitost $\Theta(n^4)$.

Rychlá (rychlejší) metoda

- Jak zrychlit?
- Pokud nemáme záporný cyklus, pak $L^{(m)} = L^{(n-1)}$ pro $m \geq n-1$.
- Násobení matic definované procedurou EXTEND-SHORTEST-PATHS je asociativní.
- Nemusíme tedy počítat $n-1$ násobení, ale pouze $\lceil \log n - 1 \rceil$
- Počítáme sekvenci matic

$$\begin{array}{rcccc} L^{(1)} & = & W & & \\ L^{(2)} & = & W^2 & & \\ L^{(4)} & = & W^4 & = & W^2 \cdot W^2 \\ L^{(8)} & = & W^8 & = & W^4 \cdot W^4 \\ & & \vdots & & \\ L^{(2^{\lceil \log n-1 \rceil})} & = & W^{(2^{\lceil \log n-1 \rceil})} & = & W^{2^{\lceil \log n-1 \rceil}-1} \cdot W^{2^{\lceil \log n-1 \rceil}-1} \end{array}$$

  Protože $2^{\lceil \log n-1 \rceil} \geq n-1$, je $L^{(2^{\lceil \log n-1 \rceil})} = L^{(n-1)}$.

Rychlá (rychlejší) metoda

- Složitost $\Theta(n^3 \log n)$.

Floyd-Warshallův algoritmus

- Připouštíme záporné hrany.

- Avšak předpokládáme, že nemáme záporné cykly.

```
FAST-ALL-SHORTEST-PATHS(W)
1  n ← rows[W]
2  L^(1) ← W
3  m ← 1
4  while m < n − 1
5        do L^(2m) ← EXTEND-SHORTEST-PATHS(L^(m), L^(m))
6           m ← 2m
7  return L^(m)
```

Struktura nejkratších cest

- Vnitřní uzel nejkratší cesty $p = \langle v_1, v_2, \ldots, v_k \rangle$ je libovolný uzel v_i pro $1 < i < k$.
- Nechť $\{1, 2, \ldots, k\} \subseteq V = \{1, 2, \ldots, n\}$.
- Pro $i, j \in V$, uvažme všechny cesty z i do j, kde vnitřní uzly jsou z množiny $\{1, 2, \ldots, k\}$.
- Nechť p je nejkratší taková cesta.
- Floyd-Warshallův algoritmus využívá vztahu mezi cestou p a nejkratší cestou z i do j, která má vnitřní uzly z množiny $\{1, 2, \ldots, k-1\}$.
  - k není vnitřní uzel p, pak všechny vnitřní uzly p jsou z $\{1, 2, \ldots, k-1\}$. Tedy nejkratší cesta z i do j s vnitřními uzly z $\{1, 2, \ldots, k-1\}$ je rovněž nejkratší cesta z i do j s vnitřními uzly z $\{1, 2, \ldots, k\}$.
  - k je vnitřní uzel p, pak $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$. Přitom p_1 je nejkratší cesta z i do k s vnitřními uzly z $\{1, 2, \ldots, k-1\}$ a p_2 je nejkratší cesta z k do j s vnitřními uzly z $\{1, 2, \ldots, k-1\}$.

Rekurze

- Nechť $d_{ij}^{(k)}$ je ohodnocení nejkratší cesty z i do j, která má vnitřní uzly pouze z množiny $\{1, 2, \ldots, k\}$.
- $k = 0$, právě když $d_{ij}^{(0)} = w_{ij}$. Tedy

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{pro } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{pro } k \geq 1 \end{cases}$$

- Protože všechny vnitřní uzly jsou z množiny $V = \{1, 2, \ldots, n\}$, matice $D^{(n)} = (d_{ij}^{(n)})$ dává $d_{ij}^{(n)} = \delta(i, j)$ pro $i, j \in V$.

Výpočet

- Složitost $\Theta(n^3)$.

```
FLOYD-WARSHALL(W)
1  n ← rows[W]
2  D^(0) ← W
3  for k ← 1 to n
4      do for i ← 1 to n
5             do for j ← 1 to n
6                    do d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
7  return D^(n)
```

Konstrukce nejkratší cesty

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{pro } i = j \text{ nebo } w_{ij} = \infty \\ i & \text{pro } i \neq j \text{ a } w_{ij} < \infty \end{cases}$$

Pro $k \geq 1$,

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{pro } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{pro } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Tranzitivní uzávěr grafu

- Dán orientovaný graf $G = (V, E)$, $V = \{1, 2, \ldots, n\}$.
- Tranzitivní uzávěr grafu G je graf $G^* = (V, E^*)$, kde

$$E^* = \{(i, j) : \text{existuje cesta z } i \text{ do } j \text{ v } G\}.$$

- Ohodnoťme hrany hodnotou 1 a spusťme Floyd-Warshallův algoritmus ($\Theta(n^3)$).
  - Pokud existuje cesta z i do j, pak $d_{ij} < n$.
  - Jinak je $d_{ij} = \infty$.
- Lze trochu vylepšit....

Tranzitivní uzávěr grafu II

- Použijme logické spojky $\vee$ a $\wedge$ místo min a $+$.
- Definujme $t_{ij}^{(k)}$, $i, j, k \in \{1, 2, \ldots, n\}$, tak, že je rovno 1 pokud existuje cesta z i do j s vnitřními uzly z $\{1, 2, \ldots, k\}$, 0 jinak.
- Tedy

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{pro } i \neq j \text{ a } (i, j) \notin E \\ 1 & \text{pro } i = j \text{ nebo } (i, j) \in E \end{cases}$$

a pro $k \geq 1$,

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} \right).$$

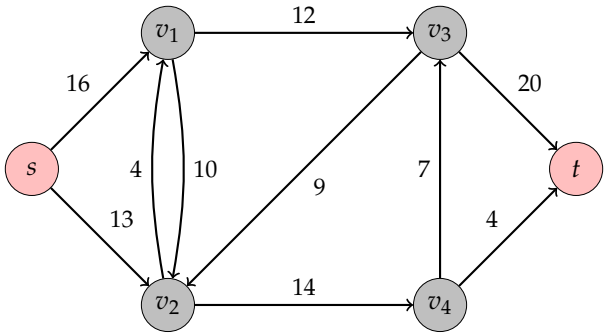
- Stejně jako ve Floyd-Warshallově algoritmu máme tři **for**-cykly, tedy složitost $\Theta(n^3)$. Proč je lepší?
- Protože logické operace na bitech jsou obvykle rychlejší než aritmetické operace na integerech. Navíc, v paměti jsou jen bity, ne bajty.

9 Toky v síti

Síť

- síť $G = (V, E)$ je orientovaný graf,
- kde každá hrana $(u, v) \in E$ má nezápornou kapacitu $c(u, v) \geq 0$.
- Nechť $c(u, v) = 0$, pokud $(u, v) \notin E$.
- Jsou specifikovány dva uzly: zdroj s a spotřebič t
- Každý uzel leží na cestě z s do t, tj. $s \rightsquigarrow v \rightsquigarrow t$ pro každý $v \in V$.
- Síť je tedy souvislý graf a $m \geq n - 1$.

Síť – příklad



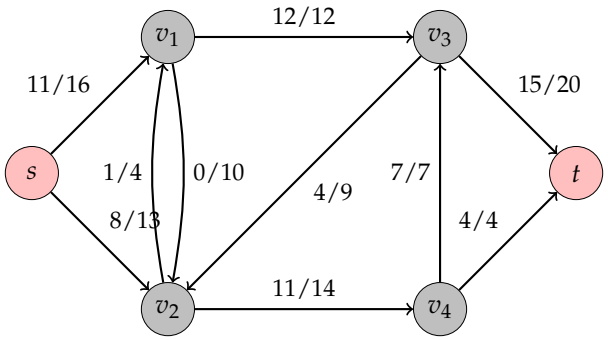
Tok v síti

- Tok v síti G je reálná funkce $f : V \times V \to \mathbb{R}$ splňující:

  1. Pro každé $u, v \in V$, $f(u, v) \leq c(u, v)$.
  2. Pro každé $u, v \in V$, $f(u, v) = -f(v, u)$.
  3. Pro každé $u \in V - \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$.

- Poslední podmínka říká, že to, co vtéká do uzlu u z něj také vytéká.
- $f(u, v)$ se nazývá tok z uzlu u do uzlu v.
- Velikost toku je definována jako

$$|f| = \sum_{v \in V} f(s, v)\,.$$



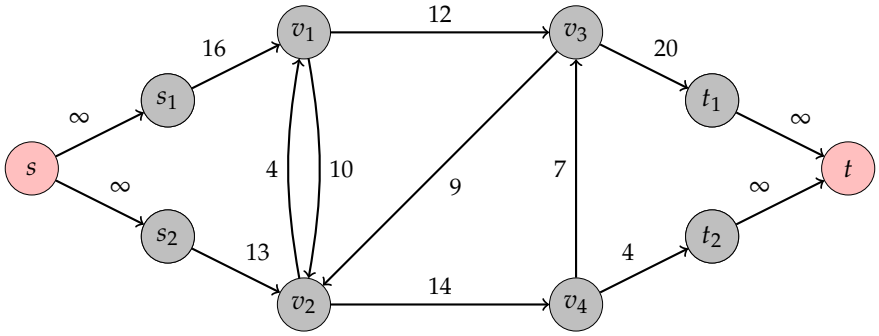
Tok v síti – příklad

- Na hranách jsou hodnoty $f(u,v)/c(u,v)$.
- Ověřte, že jde o tok v síti.
- $|f| =$???
- $|f| = 19$.

Maximální tok

- Máme danou síť G se zdrojem s a spotřebičem t.
- Hledáme tok maximální velikosti.

Více zdrojů a spotřebičů



- Co s tím?
- Vytvoříme nový zdroj a spotřebič a nastavíme novým hranám kapacitu c na ∞.

Práce s toky

- Pro $X, Y \subseteq V$, definujme $f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$.
- Pak platí, že $|f| = f(s,V)$.
- Pro $X \subseteq V$, $f(X,X) = 0$ — s každým $f(u,v)$ máme i $f(v,u)$.

- Pro $X, Y \subseteq V$, $f(X, Y) = -f(Y, X)$.
- Pro $X, Y, Z \subseteq V$, $X \cap Y = \emptyset$,
$$f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$$
a
$$f(Z, X \cup Y) = f(Z, X) + f(Z, Y)\,.$$

Práce s toky – příklad

Dokaže, že $|f| = f(V, t)$.

Důkaz.
- $|f| = f(s, V)$
- Víme $f(V, V) = f(s, V) + f(V - s, V)$ – výše.
- Tedy $f(s, V) = f(V, V) - f(V - s, V)$.
- Víme $f(V, V) = 0$ – výše.
- Tedy $f(s, V) = -f(V - s, V)$.
- Víme $f(V, V - s) = f(V, t) + f(V, V - s - t)$ – výše.
- Z přechozího a vlastnosti toku víme, že $f(V, V - s - t) = -f(V - s - t, V) = -\sum_{u \in V - \{s,t\}} \sum_{v \in V} f(u, v) =$ $-\sum_{u \in V - \{s,t\}} 0 = 0$.
- Tedy $|f| = f(V, t)$.

□

Ford-Fulkersonova metoda

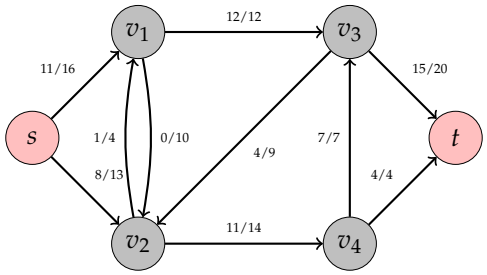
- Pro nalezení maximálního toku v síti.
- Ne algoritmus, protože existuje několik implementací s odlišnou složitostí.

```
FORD-FULKERSON-METHOD(G, s, t)
1  inicializuj f(u, v) = 0 pro u, v ∈ V
2  while existuje zlepšující cesta p
3        do zlepši tok f podle p
4  return f
```

- Zlepšující cesta je cesta z s do t, kde můžeme zvětšit tok.

Reziduální síť

- Reziduální kapacita hrany (u, v) je
$$c_f(u, v) = c(u, v) - f(u, v)\,.$$
- Např. $c_f(s, v_1) = 16 - 11 = 5$.
- Tok $f(u, v)$ tedy může být zlepšen až o 5 jednotek.



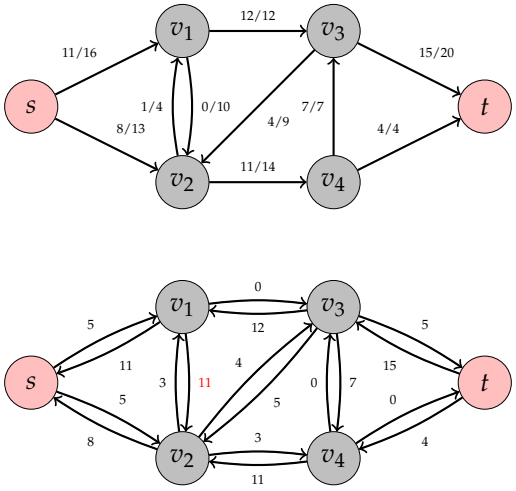
Reziduální síť

- Nechť $G = (V, E)$ je síť a f je tok v síti G.
- Reziduální síť sítě G indukovaná tokem f je síť $G_f = (V, E_f)$, kde

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}.$$

- Platí, že $|E_f| \leq 2|E|$ – rozmyslete.

Síť a její reziduální síť



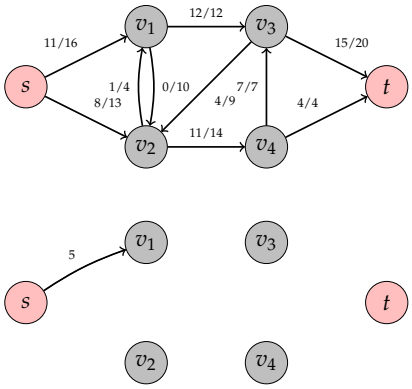
- Pozor! $f(v_1, v_2) = 0 + (-1)$, proto $c_f(v_1, v_2) = 10 - (-1) = 11$.

Reziduální síť

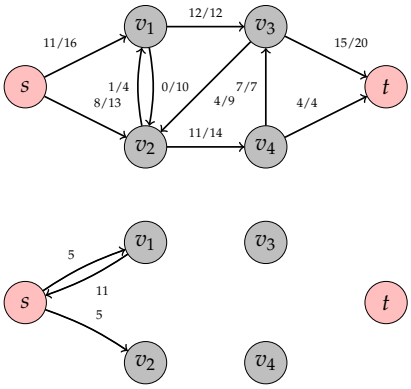
Lemma 24. *Nechť $G = (V, E)$ je síť a f je tok v G. Nechť G_f je reziduální síť G indukovaná f a nechť f' je tok v G_f. Pak $f + f'$ je tok v G s hodnotou $|f + f'| = |f| + |f'|$.*

Důkaz. • Spočívá v ověření podmínek z definice toku. □

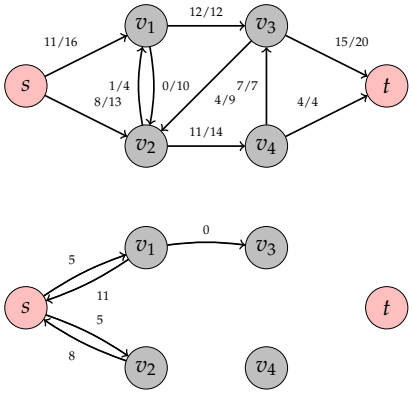
Síť a její reziduální síť



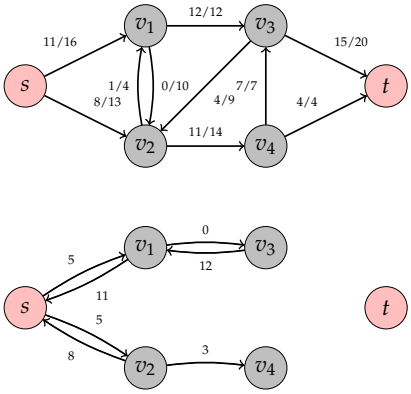
Síť a její reziduální síť



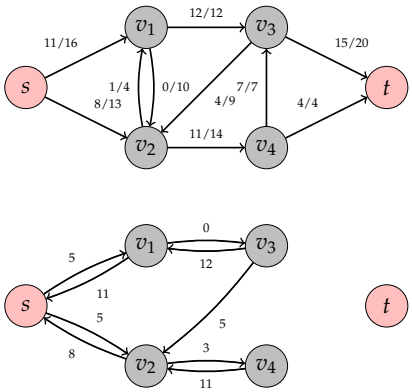
Síť a její reziduální síť



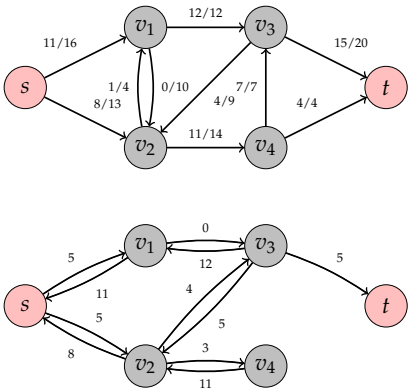
Síť a její reziduální síť



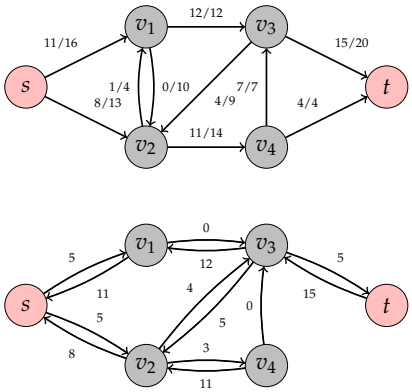
Síť a její reziduální síť



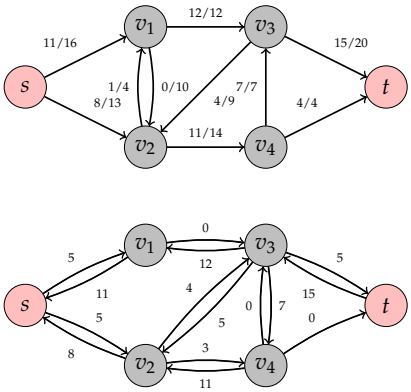
Síť a její reziduální síť



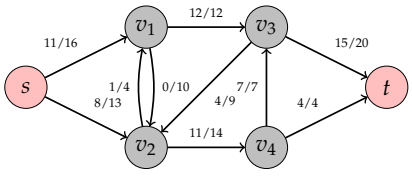
Síť a její reziduální síť

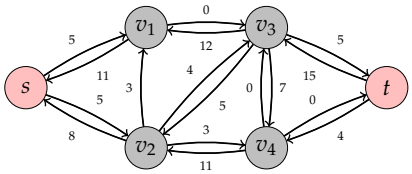


Síť a její reziduální síť

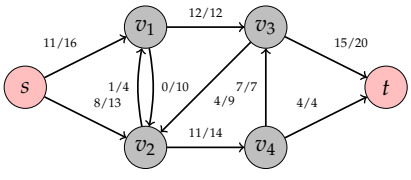


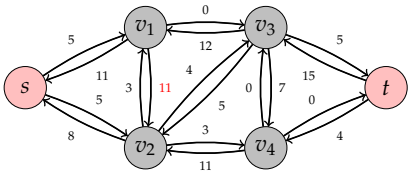
Síť a její reziduální síť





Síť a její reziduální síť





- Pozor! $f(v_1, v_2) = 0 + (-1)$, proto $c_f(v_1, v_2) = 10 - (-1) = 11$.

Podmínka 1
Chceme ukázat $(f + f')(u, v) \leq c(u, v)$.

Důkaz.
- $f'(u, v) \leq c_f(u, v)$.
- $$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v) \\ &\leq f(u, v) + (c(u, v) - f(u, v)) \\ &= c(u, v).\end{aligned}$$

□

Podmínka 2
Chceme ukázat $(f + f')(u, v) = -(f + f')(v, u)$.

Důkaz.
- $$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v) \\ &= -f(v, u) - f'(v, u) \\ &= -(f(v, u) + f'(v, u)) \\ &= -(f + f')(v, u).\end{aligned}$$

□

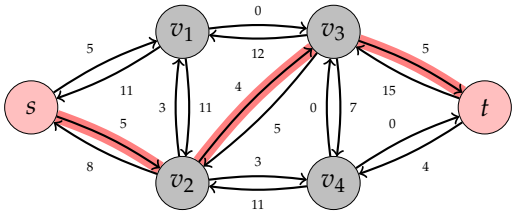
Podmínka 3
Chceme ukázat, že pro $u \in V - \{s, t\}$, $\sum_{v \in V} (f + f')(u, v) = 0$.

Důkaz.
- $$\begin{aligned}\sum_{v \in V} (f + f')(u, v) &= \sum_{v \in V} (f(u, v) + f'(u, v)) \\ &= \sum_{v \in V} f(u, v) + \sum_{v \in V} f'(u, v) \\ &= 0 + 0 = 0.\end{aligned}$$

□

Velikost výsledného toku

- $$\begin{aligned}|f + f'| &= \sum_{v \in V} (f + f')(s, v) \\ &= \sum_{v \in V} (f(s, v) + f'(s, v)) \\ &= \sum_{v \in V} f(s, v) + \sum_{v \in V} f'(s, v) \\ &= |f| + |f'|.\end{aligned}$$



Zlepšující cesta – příklad

- Nechť $G = (V, E)$ je síť a f tok.
- Zlepšující cesta p je jednoduchá cesta z s do t.

- Pomocí této cesty můžeme zlepšit tok o 4 jednotky.
- Reziduální kapacita zlepšující cesty p je

$$c_f(p) = \min\{c_f(u, v) : (u, v) \text{ je na cestě } p\}.$$

Lemma 25. *Nechť $G = (V, E)$ je síť, f její tok a p zlepšující cesta v G_f. Definujme funkci*

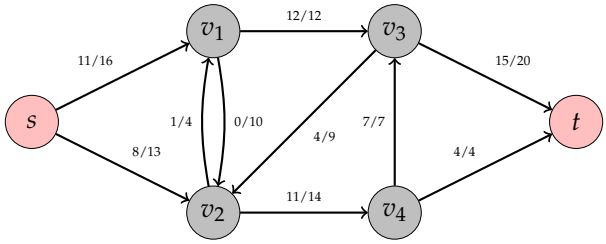
$$f_p(u, v) = \begin{cases} c_f(p) & \text{pro } (u, v) \text{ na } p \\ -c_f(p) & \text{pro } (v, u) \text{ na } p \\ 0 & \text{jinak} \end{cases}$$

Pak f_p je tok v G_f velikosti $|f_p| = c_f(p) > 0$.

Důkaz. DÚ. □

Corollary 26. *Nechť $f' = f + f_p$. Pak f' je tok v G velikosti $|f'| = |f| + |f_p| > |f|$.*

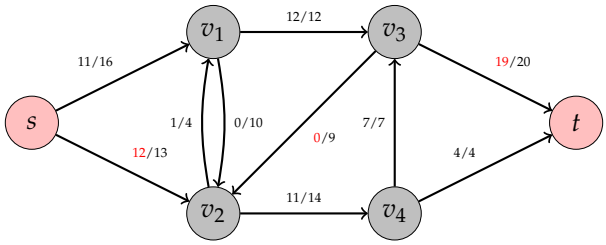
Reziduální síť a její zlepšení o 4 na $s \rightsquigarrow v_2 \rightsquigarrow v_3 \rightsquigarrow t$



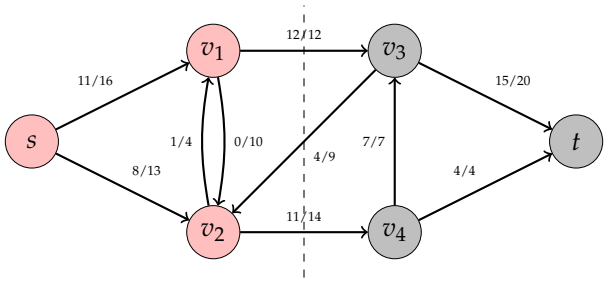
10 Řez v síti

Řez v síti

- Řez v síti $G = (V, E)$ je rozklad množiny V na S a $T = V - S$ takový, že $s \in S$ a $t \in T$.
- Tok řezem je definován jako $f(S, T)$.
- Kapacita řezu (S, T) je $c(S, T)$.
- Minimální řez je řez s minimální kapacitou.



Řez v síti – příklad



- Tok řezem $f(\{s, v_1, v_2\}, \{v_3, v_4, t\}) = f(v_1, v_3) + f(v_2, v_3) + f(v_2, v_4) = 12 + (-4) + 11 = 19$.
- Kapacita řezu $c(\{s, v_1, v_2\}, \{v_3, v_4, t\}) = c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26$.

Vlastnosti

Lemma 27. *Nechť f je tok v G se zdrojem s a spotřebičem t a nechť (S, T) je řez G. Pak $|f| = f(S, T)$.*

Důkaz.

- $f(S, T) = f(S, V) - f(S, S)$
  $= f(S, V)$
  $= f(s, V) + f(S - \{s\}, V)$
  $= f(s, V)$
  $= |f|$

□

Vlastnosti

Corollary 28. *Velikost libovolného toku f v G je shora omezena kapacitou libovolného řezu v G.*

Důkaz.

- $|f| = f(S, T)$
  $= \sum_{u \in S} \sum_{v \in T} f(u, v)$
  $\leq \sum_{u \in S} \sum_{v \in T} c(u, v)$
  $= c(S, T)$

□

Velikost maximálního toku je vždy nejvýše kapacita minimálního řezu.

Hlavní věta
Nechť f je tok v G se zdrojem s a spotřebičem t. Pak následující je ekvivalentní.

1. f je maximální tok.
2. Reziduální síť G_f nemá zlepšující cestu.
3. $|f| = c(S,T)$ pro nějaký řez (S,T) v G.

Důkaz.

- (1) $\Rightarrow$ (2):
  - Nechť f je maximální a p je zlepšující cesta v G.
  - Pak ale $f + f_p$ je tok v G a $|f + f_p| > |f|$. Spor.

□

Hlavní věta
Nechť f je tok v G se zdrojem s a spotřebičem t. Pak následující je ekvivalentní.

1. f je maximální tok.
2. Reziduální síť G_f nemá zlepšující cestu.
3. $|f| = c(S,T)$ pro nějaký řez (S,T) v G.

Důkaz.

- (2) $\Rightarrow$ (3):
  - Nechť G_f nemá zlepšující cestu, tj. v G_f neexistuje cesta z s do t.
  - Nechť
$$S = \{v \in V : \text{existuje cesta z } s \text{ do } v \text{ v } G_f\}$$
  - a nechť $T = V - S$.
  - Protože $s \in S$ a $t \in T$ je (S,T) řez v G.
  - Pro $u \in S$ a $v \in T$ máme $f(u,v) = c(u,v)$, jinak $(u,v) \in E_f$ a to dává $v \in S$.
  - $|f| = f(S,T) = c(S,T)$.

□

Hlavní věta
Nechť f je tok v G se zdrojem s a spotřebičem t. Pak následující je ekvivalentní.

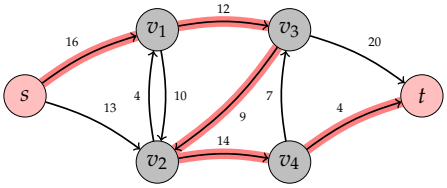
1. f je maximální tok.
2. Reziduální síť G_f nemá zlepšující cestu.
3. $|f| = c(S,T)$ pro nějaký řez (S,T) v G.

Důkaz.

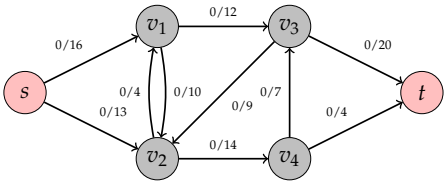
- (3) $\Rightarrow$ (1):
  - $|f| \leq c(S,T)$ pro libovolný řez (S,T).
  - Z $|f| = c(S,T)$ plyne maximalita f.

□

Základní Ford-Fulkersonův algoritmus – příklad

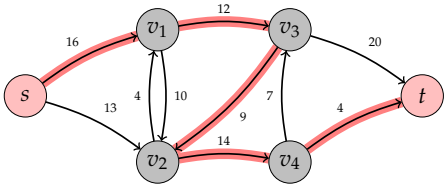


Obrázek: Reziduální síť a cesta v ní z s do t.

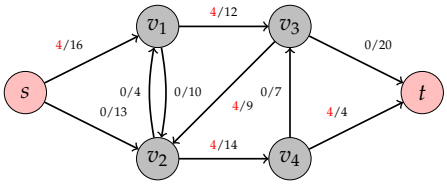


Obrázek: Vylepšený tok podle výše uvedené cesty.

Základní Ford-Fulkersonův algoritmus – příklad

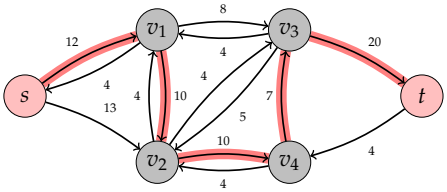


Obrázek: Reziduální síť a cesta v ní z s do t.

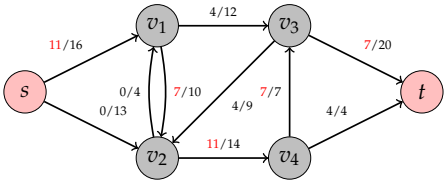


Obrázek: Vylepšený tok podle výše uvedené cesty.

Základní Ford-Fulkersonův algoritmus – příklad

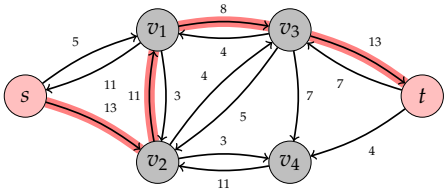


Obrázek: Reziduální síť a cesta v ní z s do t.

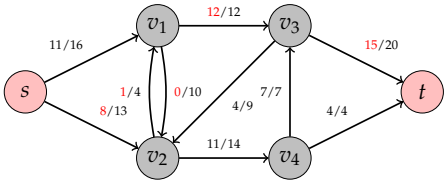


Obrázek: Vylepšený tok podle výše uvedené cesty.

Základní Ford-Fulkersonův algoritmus – příklad

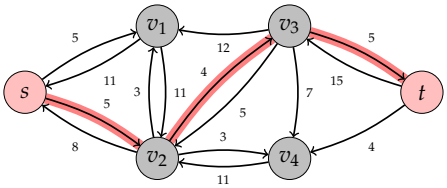


Obrázek: Reziduální síť a cesta v ní z s do t.

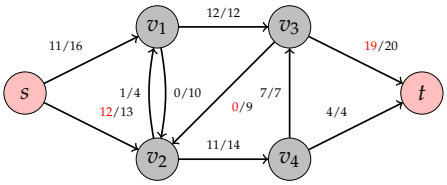


Obrázek: Vylepšený tok podle výše uvedené cesty.

Základní Ford-Fulkersonův algoritmus – příklad

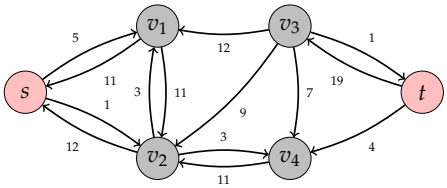


Obrázek: Reziduální síť a cesta v ní z s do t.

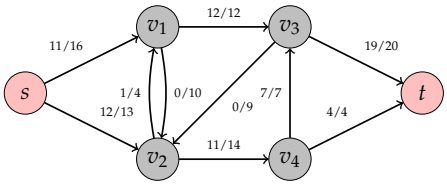


Obrázek: Vylepšený tok podle výše uvedené cesty.

Základní Ford-Fulkersonův algoritmus – příklad

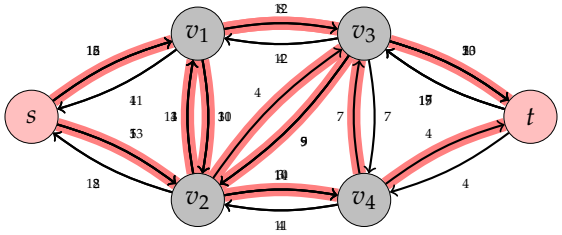


Obrázek: Reziduální síť a cesta v ní z s do t.



Obrázek: Vylepšený tok podle výše uvedené cesty.

```
FORD-FULKERSON(G, s, t)
1  for každou hranu (u, v) ∈ E
2      do f[u, v] ← 0
3         f[v, u] ← 0
4  while existuje cesta p z s do t v reziduální síti G_f
5        do c_f(p) ← min{c_f(u, v) : (u, v) na p}
6           for každou hranu (u, v) na p
7               do f[u, v] ← f[u, v] + c_f(p)
8                  f[v, u] ← −f[u, v]
```



Obrázek 17: Reziduální síť a cesta v ní z s do t.

Základní Ford-Fulkersonův algoritmus

- Složitost závisí na řádku 4.
- Prohledávání do šířky dává složitost $O(nm^2)$ – tzv. Edmonds-Karpův algoritmus.

Základní Ford-Fulkersonův algoritmus – příklad

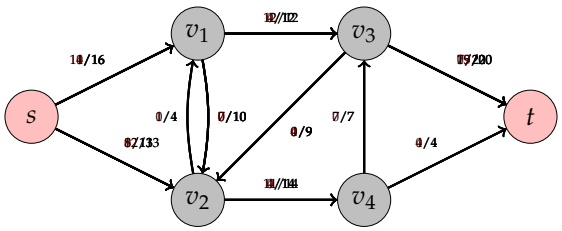
11 Maximální párování v bipartitním grafu

Maximální párování v bipartitním grafu

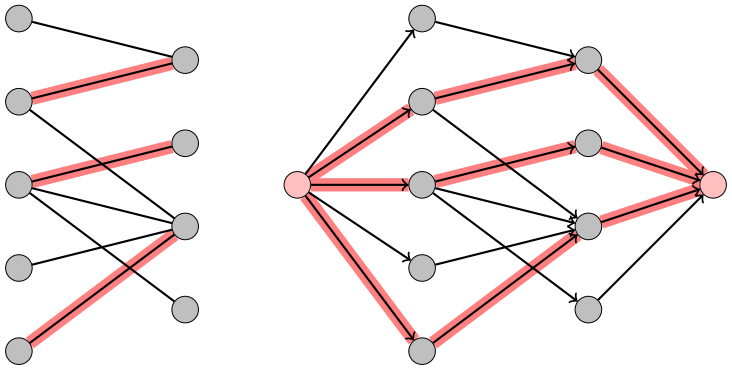
- Nechť $G = (V, E)$ je neorientovaný graf.
- Párování v G je podmožina hran $M \subseteq E$ taková, že pro každý uzel $v \in V$, v je incidentní s nejvýše *jednou* hranou z M.
- Uzel je popárován, pokud je incidentní s nějakou hranou z M.
- Maximální párování je párování s maximální kardinalitou.
- Omezujeme se pouze na bipartitní grafy, tj. takové, kde V se dá rozložit na $V = L \cup R$, $R \cap L = \emptyset$ a $E \subseteq L \times R$.
- Použijeme Ford-Fulkersonovu metodu.

Transformace na problém nalezení maximálního toku

- Složitost $O(nm)$.



Obrázek 18: Vylepšený tok podle výše uvedené cesty.



Obrázek 19: Bipartitní graf a odpovídající síť. Vyznačeno maximální párování a maximální tok (kapacita hran 1)

12 Barvení grafů

Notace

- Nechť $G = (V, E)$ je graf.
- Cíl: obarvit hrany/uzly tak, aby žádné dvě incidentní hrany (dva incidentní uzly) neměly stejnou barvu.
- Formálně, obarvení je funkce
$$f : E \to B$$
($f : V \to B$), kde B je nějaká množina barev a $f(e_1) \neq f(e_2)$ pro $e_1 \cap e_2 \neq \emptyset$ ($f(u) \neq f(v)$, pokud $\{u, v\}$ je hrana).
- $\psi_e(G)$ značí minimální počet barev potřebný k hranovému obarvení G.
- $\psi_v(G)$ značí minimální počet barev potřebný k (vrcholovému) obarvení G.
- Δ značí maximální stupeň grafu G.

12.1 Hranové barvení grafů

Hranové barvení grafů

- Jednoduché pozorování
- $\Delta \leq \psi_e(G)$.

Hranové barvení grafů

Theorem 29. *Pokud je G bipartitní, pak $\psi_e(G) = \Delta$.*

Důkaz

- Indukcí k mohutnosti množiny hran.
- $|E| = 1$ – snadné.
- Nechť všechny až na jednu hranu jsou obarveny nejvýše Δ barvami.
- Nechť (u, v) je neobarvená hrana.
- Protože máme k dispozici Δ barev, alespoň jedna barva není incidentní s u a alespoň jedna není incidentní s v.
- Pokud jsou tyto dvě barvy stejné, tak máme hotovo.
- Nechť jsou tedy různé, označené postupně C_1 a C_2.

Hranové barvení grafů

- Nechť jsou tedy různé, označené postupně C_1 a C_2.
- Nechť $H_u(C_1, C_2)$ je podgraf obsahující u a všechny hrany dosažitelné z u obarvené pouze barvami C_1 a C_2.
- Protože (u, v) je hrana, patří u a v do různých komponent bipartitního grafu.
- Pak ale každá cesta z u do v v $H_u(C_1, C_2)$ musí mít poslední hranu obarvenou C_2.
- Hrana obarvená C_2 však není incidentní s v, proto v není v $H_u(C_1, C_2)$.
- Záměnou barev C_1 za C_2 a naopak v $H_u(C_1, C_2)$ dostaneme, že C_2 není incidentní s u.
- (u, v) tedy může být obarvena C_2. □

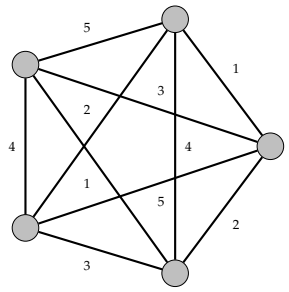
Hranové barvení grafů

Theorem 30. *Pokud je G úplný graf s n uzly, pak* $\psi_e(G) = \begin{cases} \Delta & n \text{ sudé} \\ \Delta + 1 & n \text{ liché} \end{cases}$

Důkaz

- Pokud je n liché, nakresleme graf jako pravidelný polygon (viz dále).
- Obarvíme hraniční hrany barvami 1 až $n = \Delta + 1$.
- Každá vnitřní hrana je obarvena stejně, jako s ní paralelní hrana.

Hranové barvení grafů



Hranové barvení grafů

- Nejde obarvit $\Delta = n - 1$ barvami:
- Pokud by šlo, pak protože G má $\frac{1}{2}n(n-1)$ hran, alespoň $\frac{1}{2}n$ hran by mělo stejnou barvu.
- Nechť $M \subseteq E$ taková, že žádné dvě hrany z M nejsou incidentní.
- Pak $|M| \leq \frac{1}{2}(n-1)$ – (dokažte).

Hranové barvení grafů

- Konečně, nechť n je sudé.
- Dívejme se na G jako na úplný graf G' na $n-1$ uzlech s jedním uzlem navíc spojeným se všemi ostatními uzly.
- Použijme předchozí postup na G'.
- V každém uzlu schází jedna barva.
- Tyto barvy jsou navzájem různé, proto můžeme ony nové hrany jimi obarvit.
- Použito pouze $\Delta = n - 1$ barev. □

Hranové barvení grafů

Theorem 31. *Nechť G je prostý graf. Pak $\Delta \leq \psi_e(G) \leq \Delta + 1$.*

Důkaz

- Nutno ukázat, že $\psi_e(G) \leq \Delta + 1$.
- Indukcí k počtu hran.
- Pro jednu hranu platí.
- Nechť tedy všechny hrany kromě hrany (v_0, v_1) jsou obarveny nejvýše $\Delta + 1$ barvami.
- Alespoň jedna barva není ve v_0 a alespoň jedna není ve v_1.
- Pokud jde o tutéž barvu, hotovo.

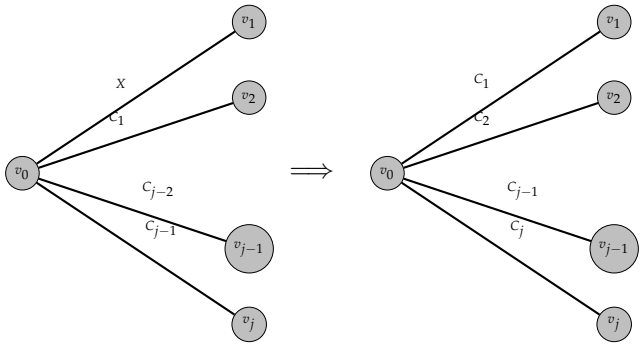
Hranové barvení grafů

- Nechť tedy C_0 je barva, která není ve v_0 a C_1 není ve v_1.
- Konstruujme posloupnost hran $(v_0, v_1), (v_0, v_2), (v_0, v_3), \ldots$ tak, že
  - C_i není ve v_i a
  - (v_0, v_{i+1}) má barvu C_i.
- Mějme tedy posloupnost $(v_0, v_1), (v_0, v_2), (v_0, v_3), \ldots, (v_0, v_i)$ a $C_1, C_2, C_3, \ldots, C_i$, pro nějaké $i \geq 0$.
- Všimněme si, že máme nejvýše jednu hranu, (v_0, v), s barvou C_i.
  - Pokud takové v existuje a $v \notin \{v_1, v_2, \ldots, v_i\}$, pak přidáme hranu (v_0, v_{i+1}), kde $v_{i+1} = v$, a C_{i+1} je barva, která není ve v_{i+1}.
  - Jinak ukončíme budování posloupnosti.
- Každá posloupnost končí s nejvýše Δ prvky.

Hranové barvení grafů

- Nechť výsledná posloupnost je $(v_0, v_1), (v_0, v_2), (v_0, v_3), \ldots, (v_0, v_j)$ a $C_1, C_2, C_3, \ldots, C_j$, pro nějaké $j \geq 0$.

  i) Neexistuje hrana (v_0, v) s barvou C_j, pak uděláme následující přebarvení ($X \neq C_j$):



Hranové barvení grafů

- Nechť výsledná posloupnost je $(v_0, v_1), (v_0, v_2), (v_0, v_3), \ldots, (v_0, v_j)$ a $C_1, C_2, C_3, \ldots, C_j$, pro nějaké $j \geq 0$.

  ii) Nechť existuje $k < j$ takové, že (v_0, v_k) má barvu C_j.
  - Pak pro $i < k$ přebarvíme hrany jako výše, tj. (v_0, v_i) dostane barvu C_i.
  - (v_0, v_k) zůstane neobarvená.

- Každá komponenta $H(C_0, C_j)$ – podgraf obsahující všechny hrany barvy C_0 a C_j – je buď cesta, nebo cyklus, protože každý uzel má nejvýše jednu hranu barvy C_0 a jednu barvy C_j.
- Alespoň jedna barva z C_0 a C_j není v každém z uzlů v_0, v_k, v_j.
- Proto ne všechny v jedné komponentě $H(C_0, C_j)$: $v_0 \overset{C_j}{\to} x \overset{X}{\to} y \ldots \overset{C_0}{\to} v_k$ a už se nedostaneme do v_j.

Hranové barvení grafů

a) $v_0 \notin H_{v_k}(C_0, C_j)$ – komponenta $H(C_0, C_j)$ obsahující v_k – pak $C_0 \leftrightarrow C_j$ v $H_{v_k}(C_0, C_j)$, což dává, že C_0 není ve v_k.

- C_0 není ani ve v_0, proto obarvíme (v_0, v_k) barvou C_0.

b) $v_0 \notin H_{v_j}(C_0, C_j)$, pak přebarvíme

  - (v_0, v_i) barvou C_i, $k \leq i < j$,
  - (v_0, v_j) zůstane neobarvená.

- Při přebarvení se nepoužila ani C_0, ani C_j, proto $H(C_0, C_j)$ nezměněno.
- Opět $C_0 \leftrightarrow C_j$ v $H_{v_j}(C_0, C_j)$ a máme, že C_0 není ve v_j.
- Obarvíme (v_0, v_j) barvou C_0. □

Hranové barvení grafů

- Předchozí důkaz dává polynomiální algoritmus.
- Vidíte ho? :-)
- Jaká je složitost?

- Problém, zda $\psi_e(G) = \Delta$ je NP-úplný.

12.2 (Vrcholové) barvení grafů

Barvení grafů

- Je graf obarvitelný nejvýše k barvami? je NP-úplný problém.

Barvení grafů

Theorem 32. *Libovolný (prostý) graf G lze obarvit $\Delta + 1$ barvami.*

Důkaz.
- Indukcí k n.
- $n = 1$, hotovo.
- Pokud přidáme uzel u, pak bude spojen nejvýše s Δ jinými uzly.
- Barev máme $\Delta + 1$, proto můžeme u obarvit nějakou barvou.

□

Barvení grafů

- Ve většině případů ale platí, že $\psi_v(G) < \Delta + 1$.
- Příklad:
- G planární, pak $\psi_v(G) \leq 4$, přitom $\Delta = k$, pro libovolné $k > 0$.

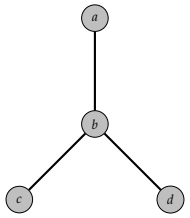
- Rychlosoutěž: Jaký algoritmus na obarvení uzlů vás napadne?

12.3 Chromatický polynom

Chromatický polynom

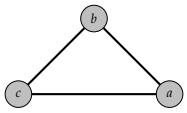
- Nechť $P_k(G)$ označuje počet způsobů obarvení grafu G k barvami.
- $P_k(G)$ je polynom.

Chromatický polynom



Obrázek 20: Graf G_1.

- b ... dostane libovolnou z k barev.
- a, c, d ... libovolnou z $k - 1$ zbývajících barev.
- $P_k(G_1) = k(k-1)^3$
- Obecně, nechť T_n je strom na n uzlech. Pak $P_k(T_n) = k(k-1)^{n-1}$.

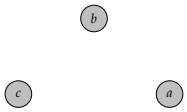


Obrázek 21: Graf G_2.

Chromatický polynom

- a ... dostane libovolnou z k barev.
- b ... libovolnou z $k-1$ zbývajících barev.
- c ... libovolnou z $k-2$ zbývajících barev.
- $P_k(G_2) = k(k-1)(k-2)$
- Obecně, nechť K_n je úplný graf na n uzlech.
- Pak $P_k(K_n) = \frac{k!}{(k-n)!}$

Chromatický polynom



Obrázek 22: Graf G_2'.

- a ... dostane libovolnou z k barev.
- b ... libovolnou z k barev.
- c ... libovolnou z k barev.
- $P_k(G_2') = k^3$
- Obecně, nechť Φ_n je graf na n uzlech bez hran.
- Pak $P_k(\Phi_n) = k^n$

Chromatický polynom

- Platí: Pokud $k < \psi_v(G)$, pak $P_k(G) = 0$.

- Nechť G je graf.
- Jak sestrojit $P_k(G)$?

- Značení:
- $G - (u, v)$ je graf vzniklý z G odebráním hrany (u, v)
- $G \circ (u, v)$ je graf vzniklý z G sloučením uzlů u a v.

Chromatický polynom

Theorem 33. *Nechť (u, v) je hrana v G, pak*

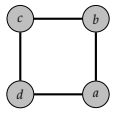
$$P_k(G) = P_k(G - (u, v)) - P_k(G \circ (u, v)) .$$

Důkaz.

- $P_k(G)$ je počet obarvení, kde u a v mají různou barvu.
- Všechna tato obarvení jsou zahrnuta i v $P_k(G - (u, v))$.
- $P_k(G - (u, v))$ však navíc obsahuje i ta obarvení, kde u a v mají stejnou barvu.
- Odečteme je tedy v $P_k(G \circ (u, v))$.

□

Chromatický polynom



Obrázek 23: Graf G_3.

- $P_k(G_3) = P_k(\Phi_4) - 4P_k(\Phi_3) + 6P_k(\Phi_2) - 3P_k(\Phi_1)$
  $= k(k-1)(k^2 - 3k + 3)$

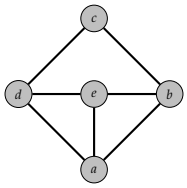
Chromatický polynom

Theorem 34. *Nechť (u, v) je hrana v G, pak*

$$P_k(G) = P_k(G - (u, v)) - P_k(G \circ (u, v)) .$$

- Pokud má graf hodně hran, je lepší použít přeformulovanou variantu:
- $P_k(G) = P_k(G + (u, v)) + P_k((G + (u, v)) \circ (u, v))$
- tj., doplňujeme na úplný graf.

Chromatický polynom



Obrázek 24: Graf G_4.

- $P_k(G_4) = P_k(K_5) + 3P_k(K_4) + 2P_k(K_3)$
  $= k(k-1)(k-2)(k^2 - 4k + 5)$

Chromatický polynom

- Nyní máme, že $\psi_v(G)$ je minimální k takové, že $P_k(G) > 0$.

- $\psi_v(G_3) = 2$
- $\psi_v(G_4) =$?

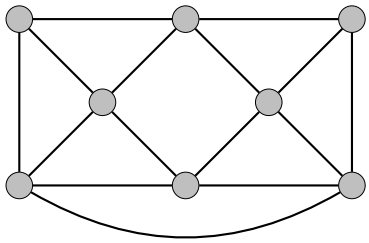
13 Eulerovské tahy

L. Euler a W. R. Hamilton

- Leonard Euler (1707 – 1783, švýcarský matematik)
  - sedm mostů města Královce
  - průchod grafem přes všechny hrany právě jednou.
- William Rowan Hamilton (1805 – 1865, anglický matematik)
  - zeměpisná hra průchodu 20 měst na vrcholech pravidelného dvanáctistěnu
  - průchod grafem přes všechny vrcholy právě jednou.
- Otázka: Jak se vyznačují eulerovské/hamiltonovské grafy?
- Alternativní definice: Cyklus = uzavřený tah

Eulerovský graf

- Eulerovský graf je graf, který obsahuje eulerovský tah, tj. tah procházející všemi hranami právě jednou.



Věta o existenci eulerovského tahu

Theorem 35. *Neorientovaný graf G má eulerovský tah, právě když je G souvislý a počet vrcholů s lichým stupněm je 0 nebo 2.*

Důkaz

- Podmínka nutná je zřejmá: Pokud existuje eulerovský tah v G, pak je G souvislý a buď je tah uzavřený, nebo oba okrajové vrcholy mají lichý stupeň.
- Podmínka postačující: Indukcí k mohutnosti množiny hran.
- Předpokládejme, že $G = (V_G, E_G)$ s $|E_G| > 2$ splňuje tuto větu.
- Obsahuje-li G dva uzly s lichým stupněm, označme je v_1 a v_2.
- Uvažujme průchod po uzavřené (příp. otevřené) kostře $T = (V_G, E_T)$ z uzlu v_i (příp. v_1), dokud nenarazím na uzel v_j, u kterého jsem prošel všechny incidentní hrany.
  (a) žádný vrchol lichého stupně $\Rightarrow v_i = v_j$.
  (b) jinak nutně $\Rightarrow v_j = v_2$.

Věta o existenci eulerovského tahu

Důkaz (pokračování)

- Nechť $G' = G - T = (V_{G'} = \{u, v | (u, v) \in E_G - E_T\}, E_G - E_T) \Rightarrow$ potencionálně nesouvislý, jen sudé stupně.
- Z IP má G' eulerovský tah pro každou komponentu.
- Ze spojitosti G plyne, že $V_T \cap V_{G'} \neq \emptyset$.
- Do T vložíme eulerovské tahy z G' přes libovolné společné uzly. □

Orientovaný eulerovský tah

Corollary 36. *Orientovaný graf G má eulerovský tah, právě když je G souvislý a počet vrcholů s lichým stupněm je 0 nebo 2.*

Orientovaná kostra grafu $G = (V, E)$ je orientovaný strom $T = (V, E')$ s kořenem $u \in V$, kde $E' \subseteq E$ a $d_+(u) = 0$ a $d_+(v) = 1$ pro všechna $v \in V - \{u\}$.

Vyvážený graf $G = (V, E)$ je orientovaný graf s $d_+(u) = d_-(u)$ pro všechny $u \in V$.

Věta o kostře orientovaného eulerovského grafu

Theorem 37. *Mějme orientovaný eulerovský graf $G = (V, E)$ a jeho podgraf T, který vznikne průchodem eulerovského tahu z libovolného uzlu u tak, že do T přidáváme pro každý uzel $v \neq u$ první hranu vedoucí do tohoto uzlu. Pak T je orientovaná kostra grafu G s kořenem v uzlu u.*

Důkaz

- Z konstrukce T platí platí pro T, že $d_+(u) = 0$ a $d_+(v) = 1$ pro všechna $u \neq v$, $u, v \in V$.
- Tedy T má $n - 1$ hran. Nyní ukažme acykličnost (sporem):
- Předpokládejme, že T obsahuje cyklus dokončený hranou (v_i, v_j).
- $v_j \neq u$, protože $d_+(u) = 0$.
- Protože (v_i, v_j) uzavírá cyklus, bylo v_j již zpracováno, což odporuje konstrukci T. Spor! □

Věta o orientovaném eulerovském tahu

Theorem 38. *Je-li G souvislý a vyvážený orientovaný graf s orientovanou kostrou T s kořenem u, pak eulerovský cyklus v reverzním směru lze zjistit následovně:*

(a) Začneme libovolnou hranou incidentní k u.

(b) Další hrany vybíráme jako incidentní k aktuálnímu uzlu takové, že:

(i) hrana ještě nebyla vybrána,

(ii) žádná hrana z T se nevybere, dokud lze vybrat jinou.

(c) Hledání končí, když aktuální uzel nemá incidentní žádné nepoužité hrany.

Důkaz

- Vyváženost zaručuje konec v kořenu u.
- Předpokládejme, že cyklus neobsahuje hranu (v_i, v_j).

Věta o orientovaném eulerovském tahu

Důkaz

- Předpokládejme, že cyklus neobsahuje hranu (v_i, v_j).
- Kvůli vyváženosti musí být v_i koncový pro další nevyužitou hranu (v_k, v_i).
- Mějme (v_k, v_i) hranu z T, takže nebude použita kvůli kroku (b(ii)).
- Nyní následujme sekvenci hran zpětně do u.
- Protože G je vyvážený, najdeme nevyužitou hranu incidentní s u, což je spor s krokem (c). □

Algoritmus hledání orientovaného eulerovského tahu

```
EULER-CIRCUIT(G)
 1  Najdi orientovanou kostru T = (V, E_T) grafu G = (V, E)
 2  for každý uzel v ∈ V
 3      do A[v] ← ∅
 4         I[v] ← 0
 5  for každou hranu (v_i, v_j) ∈ E
        do if (v_i, v_j) ∈ E_T
           then přidej v_i na konec seznamu A[v_j]
           else přidej v_i na začátek seznamu A[v_j]
 6  EC ← ∅
 7  CV ← u
 8  while I[CV] ≤ d_+(CV)
 9      do přidej CV na začátek seznamu EC
10         I[CV] ← I[CV] + 1
11         CV ← A[CV][I[CV]]
12  Vypiš EC
```

Algoritmus hledání orientovaného eulerovského tahu

Analýza časové složitosti

- Eulerův graf má vždy $m \geq n$.
- Řádek 1: DFS, zjistím nejvyšší f a pak DFS z uzlu s nejvyšším $f \Rightarrow O(m)$.
- V cyklu *while* vždy inkrementujeme $I[CV]$, takže $\sum_{v \in V} d_+(v) = \Theta(m)$.
- Celkově tedy časová složitost $O(m)$.

Aplikace eulerovského tahu

- de Bruijnova posloupnost (žádné dva podřetězce délky k nejsou stejné)
- Úloha čínského pošťáka, který má projít všechny ulice města a vrátit se do výchozího místa.
  - Je dán orientovaný kladně ohodnocený souvislý graf.
  - Hledáme nejkraktší uzavřený sled obsahující všechny hrany.
  - Optimální řešení pro graf, který není eulerovský: $O(m + n^3)$

14 Hamiltonovské cesty a kružnice

Hamiltonovské cesty a kružnice

- Hamiltonovský graf je graf, který obsahuje hamiltonovskou kružnici (cyklus), tj. uzavřenou cestu procházející všemi vrcholy právě jednou.
- Typy hamiltonovských úloh
  - zjištění existence (příp. nalezení) hamiltonovské cesty
  - optimalizační úlohy v ohodnocených grafech
- Všechny NP-úplné

Podmínky postačující pro neobecné grafy

Theorem 39. *Každý úplný graf je hamiltonovský.*

Důkaz

- Vezměme libovolnou permutaci vrcholů. (Ostatní důkazy za DÚ.)

Theorem 40. *Každý orientovaný graf, jehož symetrizace je úplný graf, obsahuje hamiltonovskou cestu.*

Theorem 41. *Každý silně souvislý orientovaný graf, jehož symetrizace je úplný graf, je hamiltonovský graf.*

Theorem 42. *Je-li graf $G = (V, E)$ takový, že $|V| > 3$ a $\min_{v \in V}(d(v)) > n/2$, pak je G hamiltonovský graf.*

Chvátalova věta (1972)

Theorem 43. *Nechť G je prostý graf s $n \geq 3$ vrcholy. Jestliže $d(v_1) \leq d(v_2) \leq \cdots \leq d(v_n)$ je soubor stupňů jeho vrcholů a platí-li, že*

$$d(v_k) \leq k \Rightarrow d(v_{n-k}) \leq n - k \quad \text{pro } 1 \leq k \leq n/2,$$

pak G obsahuje hamiltonovskou kružnici.

Jestliže posloupnost stupňů tuto podmínku, pak existuje graf o n vrcholech, který neobsahuje hamiltonovskou kružnici (pro všechny vrcholy v_i platí $d(v_i) \geq d_i$).

- První část zaručuje existenci hamilt. kružnice pro dost velké stupně.
- Druhá část ukazuje, že jde o nejlepší možnou podmínku postačující založenou pouze na stupních vrcholů.
- Důkaz je náročný, veden sporem a nekonstrukční.

Problém obchodního cestujícího

- Obchodní cestující má navštívit nejkratší cestou n měst a vrátit se zpět.
- Formálně: Nalezení nejkratší hamiltonovské kružnice v úplném neorientovaném ohodnoceném grafu.
- Převod optimalizační úlohy na úplný graf:
  - Doplním obecný graf na úplný, hrany ohodnotíme M.
  - M je dostatečně velké (např. suma vše ohodnocení).
  - Řeším optimalizační úlohu. Výsledek obsahuje přidanou hranu, právě když neexistuje řešení v obecném grafu.
- Aplikace: dopravní úlohy (rozvoz, zásobování), plánování procesů