

Libor Polák
Grafové algoritmy

Zápisy z přednášky zpracoval:
Jan Šerák

23. května 1995



Obsah

1 Úvodní slovo autora	3
2 Intro	3
2.1 Reprezentace grafů v počítači	3
2.2 Složitost algoritmů a její klasifikace	3
3 Základní grafové algoritmy	5
3.1 Prohledávání do šířky	5
3.2 Prohledávání do hloubky	8
3.2.1 Algoritmus prohledávání do hloubky	8
3.2.2 Klasifikace hran grafu po proběhnutí DFS	10
3.2.3 Topologické třídění	11
3.3 Silně souvislé komponenty grafu	11
4 Užitečné datové struktury	15
4.1 Binární halda	15
4.2 Prioritní fronta	18
4.3 Prioritní fronta implementovaná binární haldou	19
4.4 Binomiální halda	20
4.5 Prioritní fronta implementovaná binomiální haldou	21
4.6 Analýza složitosti	22
4.7 Fibonacciho haldy	22
4.8 Zásobník	23
4.9 Datové struktury pro disjunktní množiny	24
4.9.1 První implementace	24
4.9.2 Druhá implementace	25

5	Problém minimální kostry	26
5.1	Obecný algoritmus	26
5.2	Kruskalův algoritmus	27
5.3	Prinův algoritmus	28
5.4	Složitost algoritmů	29
5.4.1	Složitost Kruskalova algoritmu	29
5.4.2	Složitost Primova algoritmu	29
6	Nejkratší cesty	30
6.1	Nejkratší cesty z daného vrcholu	30
6.2	Relaxace	31
6.3	Stromy nejkratších cest	33
6.4	Dijkstrův algoritmus	34
6.5	Bellman – Fordův algoritmus	35
6.6	Nejkratší cesty z jediného vrcholu v acyklických grafech	37
6.7	Nejkratší cesty all pairs	39
6.7.1	Floyd – Warshallův algoritmus	41
6.8	Tranzitivní uzávěr orientovaného grafu	41
7	Maximální toky v sítích	42

Seznam obrázků

1	Reprezentace grafu pomocí seznamů sousedů	4
2	Příklad matice sousednosti	4
3	Příklad na klasifikaci hran grafu po proběhnutí DFS	10
4	Příklad grafu topologicky neseříděného	12
5	Struktura binární haldy	15
6	Příklad konstrukce binomiálního stromu B_k	20
7	Příklad binomiální haldy	20
8	Datové struktury pro reprezentaci binomiální haldy	21
9	Topologická rovnost binomiálních stromů u Fibonacciho haldy	22
10	Příklad Fibonacciho haldy	23
11	Příklad množiny $\{a, b, c\}$ v reprezentaci linked list	24
12	Příklad záporného cyklu	30
13	Příklad kritické hrany	47

Seznam tabulek

1	Operace prioritní fronty a složitost jejich implementací	19
2	Seznam algoritmů, realizujících operace nad binomiální haldou	21
3	Chování funkce \log^*	25
4	Porovnání složitosti algoritmů při použití různých datových struktur	26
5	Složitosti triviálních řešení	39

1 Úvodní slovo autora

2 Intro

2.1 Reprezentace grafů v počítači

K tomu, aby se mohly grafové algoritmy realizovat v počítačovém zpracování, je třeba určit datové struktury, které budou daný graf reprezentovat. Nejčastější užívané implementace grafových algoritmů užívají těchto dvou reprezentací:

- seznamy sousedů
- maticí sousednosti

Pro ilustraci uveďme příklad těchto reprezentací. Mějme graf G z obrázku 1(a). Pak jeho reprezentace seznamy sousedů je na obrázku 1(b), a reprezentace maticí sousednosti na obrázku 2. Prvky této matice jsou určeny takto:

$$a_{ij} = \begin{cases} 1, & (i, j) \in G[E] \\ 0, & (i, j) \notin G[E] \end{cases}$$

V případě orientovaného grafu $G = (V, E)$ se reprezentace seznamy sousedů nezmění. Matice sousednosti $M = (a_{ij})$ se změní v tom smyslu, že

$$a_{ij} = \begin{cases} 1, & (i, j) \in G[E] \\ -1, & a_{ji} = 1 \\ 0, & jinak \end{cases}$$

Někdy je však matice sousednosti $M = (a_{ij})$ tvořena takto:

$$a_{ij} = \begin{cases} 1, & i \in V, j \in E, i \in j \\ 0, & jinak \end{cases}$$

2.2 Složitost algoritmů a její klasifikace

V této sekci si nadefinujeme třídy funkcí $f : \mathbf{N} \rightarrow \mathbf{R}$ podle jejich složitosti.

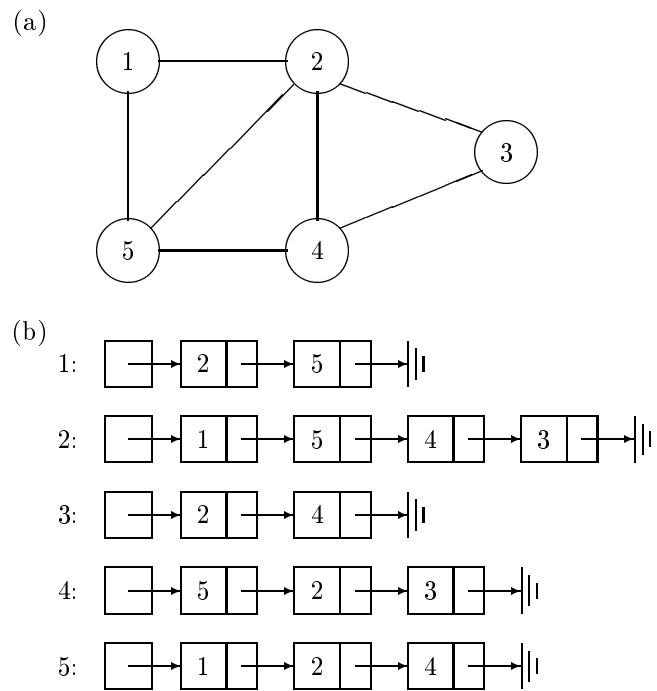
Definice 2.1

Položme $\mathbf{N} = \{0, 1, 2, \dots\}$. Nechť $g : \mathbf{N} \rightarrow \mathbf{R}$, pak definujeme $O(g)$, $\Omega(g)$, $\Theta(g)$ a $o(g)$ takto:

- $O(g) = \{f : \mathbf{N} \rightarrow \mathbf{R} \mid (\exists c, n_0 > 0) (\forall n \in \mathbf{N}) (n \geq n_0 \implies 0 \leq f(n) \leq cg(n))\}$
- $\Omega(g) = \{f : \mathbf{N} \rightarrow \mathbf{R} \mid (\exists c, n_0 > 0) (\forall n \in \mathbf{N}) (n \geq n_0 \implies 0 \leq cg(n) \leq f(n))\}$
- $\Theta(g) = \{f : \mathbf{N} \rightarrow \mathbf{R} \mid (\exists n_0, c_1, c_2 > 0) (\forall n \in \mathbf{N}) (n \geq n_0 \implies 0 \leq c_1g(n) \leq f(n) \leq c_2g(n))\}$
- $o(g) = \{f : \mathbf{N} \rightarrow \mathbf{R} \mid (\forall c > 0) (\exists n_0 > 0) (\forall n \geq n_0) (0 \leq f(n) < cg(n))\}$

Poznámka 2.2

- Místo obvyklého $f \in \Theta(g)$, píšeme $f = \Theta(g)$.
- Zřejmě platí $f = \Omega(g) \iff g = O(f)$.
- Zřejmě platí $\Theta(g) = O(g) \cap \Omega(g)$.



Obrázek 1: Reprezentace grafu pomocí seznamů sousedů

	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

Obrázek 2: Příklad matice sousednosti

3 Základní grafové algoritmy

V této kapitole budou uvedeny základní grafové algoritmy, tzn.

- Prohledávání do šířky,
- Prohledávání do hloubky,
- Topologické třídění,
- Silně souvislé komponenty.

3.1 Prohledávání do šířky

Algoritmus 3.1

BREADTH_FIRST_SEARCH(G, s)

1. for each vertex $u \in V[G] - \{s\}$ do
2. $\text{color}[u] \leftarrow \text{WHITE}$
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow \text{NIL}$
5. $\text{color}[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \{s\}$
9. while $Q \neq \emptyset$ do
10. $u \leftarrow \text{head}[Q]$
11. for each $v \in \text{Adj}[u]$ do
12. if $\text{color}[v] = \text{WHITE}$ then
13. $\text{color}[v] \leftarrow \text{GRAY}$
14. $d[v] \leftarrow d[u] + 1$
15. $\pi[v] \leftarrow u$
16. ENQUEUE(Q, v)
17. DEQUEUE(Q)
18. $\text{color}[u] \leftarrow \text{BLACK}$

$V[G]$ zde znamená množinu vrcholů daného grafu G , $\text{color}[u]$ je barva vrcholu u , $d[u]$ je „vzdálenost“¹ vrcholu u od počátku prohledávání s , $\pi[u]$ je předek vrcholu u . Q je fronta vrcholů, $\text{head}[Q]$ je vrchol v čele fronty Q , ENQUEUE(Q, v) je operace, která na konec fronty Q vloží vrchol v , DEQUEUE(Q) je operace, která odstraní vrchol z čela fronty. Množina $\text{Adj}[u]$ je množina všech vrcholů, které „sousedí“ s vrcholem u .

V řádcích 1.–8. algoritmu 3.1 se provádí inicializace: všechny vrcholy jsou bílé (s je šedý), jejich aktuální vzdálenost od s je ∞ ($d[s]$ je samozřejmě nulová), žádný vrchol nemá svého předka. V řádku 8. se inicializuje fronta Q tak, že v ní je pouze vrchol s . Ve zbývajících řádcích se provádí vlastní algoritmus – dokud není prázdná fronta, vezme se vrchol u z jejího čela a všichni jeho bílí sousedi v zešednou, „naváží“ se na u a zařadí do fronty. Až je to provedeno pro všechny sousedy v , vrchol u zčerná a odstraní se z čela fronty. Tím vznikne strom s kořenem s , definovaný relací π .

¹Vzdáleností se zde myslí počet hran na $s - u$ cestě.

Složitost BFS: Po inicializaci (1.–4.) se již žádný vrchol nepřebarvuje na bílý. Každý vrchol se do fronty zařadí maximálně jednou a tedy je maximálně jednou odstraněn. Tyto operace vyžadují čas $O(1)$, celkem tedy $O(V)^2$. Seznam sousedů daného vrcholu se zkoumá jen když je tento vrchol odstraňován z fronty, proto maximálně jednou, celkem $O(E)$ neboť součet délek těchto seznamů je $O(E)$. Inicializace vezme $O(V)$. Celkem tedy:

$$|V| \cdot O(1) + O(E) + O(V) = O(V) + O(E) + O(V) = O(V + E)$$

Definice 3.1

$G = (V, E)$ graf orientovaný nebo neorientovaný, $s \in V$. Pro $v \in V$ je $\delta(s, v)$ vzdálenost v od s , která je rovna minimálnímu počtu hran na cestě z s do v ($s - v$ cestě). Spec. $\delta(s, s) = 0$; pokud $\neg \exists s - v$ cesta: $\delta(s, v) = \infty$.

Lemma 3.2

Pro $\forall u, v \in V$:

$$(u, v) \in E \implies \delta(s, v) \leq \delta(s, u) + 1$$

Důkaz:

1. $\delta(s, u) < \infty$, pak i $\delta(s, v) < \infty$, a platí $\delta(s, v) \leq \delta(s, u) + 1$
2. $\delta(s, u) = \infty$

□

Lemma 3.3

Po skončení BREADTH_FIRST_SEARCH (alg. 3.1) je $d[v] \geq \delta(s, v)$.

Důkaz: indukci vzhledem k počtu zařazení vrcholu do fronty Q :

- $n=1$: na řádku 8: $Q \leftarrow \{s\}$
 $d[s] = 0$ (řádek 6), $\delta(s, s) = 0$
 $v \neq s$, pak $d[v] = \infty \geq \delta(s, v)$
- Ind. krok: Řádek 16: ENQUEUE(Q, v), prohledávalo se z vrcholu u . $d[u] \geq \delta(s, u)$ (ind. předpoklad). dále:
 $d[v] \leftarrow d[u] + 1 \geq \underbrace{\delta(s, u) + 1}_{\text{Lemma 3.2}} \geq \delta(s, v)$. Hodnota $d[v]$ se v budoucnu již nezmění.
- Pokud w nebylo zařazeno do fronty Q , pak se nemodifikovalo $d[w]$ a je tedy $d[w] = \infty$ a nerovnost $d[w] \geq \delta(s, w)$ též platí.

□

Lemma 3.4

Předpokládáme, že v průběhu výpočtu je $Q = (v_1, \dots, v_r)$, pak $d[v_r] \leq d[v_1] + 1$, $d[v_i] \leq d[v_{i+1}]$ pro $i = 1, \dots, r-1$.

Důkaz: Indukci vzhledem k počtu operací provedených s Q :

- $n=1$: $Q = (s)$, $r = 1$, $d[s] \leq d[s] + 1$ triv. Dále není co ověřovat.
- Nechť v_1 je odstraňováno: $r = 1$ zřejmě $r \geq 2$ (řádek 17): $Q' = (v_2, v_3, \dots, v_r)$. Pak:

$$d[v_r] \leq \underbrace{d[v_1]}_{\text{I.P.}} + \underbrace{1}_{\text{I.P.}} \leq d[v_2] + 1$$

Zbývající nerovnosti zůstávají v platnosti.

²Správnější by byl zápis: $O(n)$, kde $n = |V|$

- Nechť $v = v_{r+1}$ je přidáváno do Q : $Q' = (v_1, v_2, \dots, v_r, v_{r+1})$, $v_1 = u$, $v_{r+1} = v$

$$d[v_{r+1}] = \underbrace{d[v] = d[u] + 1}_{\text{Ř. 14}} = d[v_1] + 1$$

Tedy:

$$\underbrace{d[v_r] \leq d[v_1] + 1}_{\text{I.P.}} = d[u] + 1 = d[v] = d[v_{r+1}]$$

a ostatní nerovnosti pro $1, \dots, r-1$ zůstaly.

□

Věta 3.5 KOREKTNOST BFS

Nechť $G = (V, E)$ je orientovaný nebo neorientovaný graf, nechť algoritmus 3.1 běží ze zadaného $s \in V$. Pak algoritmus 3.1 objeví v průběhu výpočtu každý vrchol v dosažitelný z s a po zastavení platí $d[v] = \delta(s, v)$. Navíc $\forall v \in V$, $v \neq s$ je nejkratší cesta z s do $\pi(v)$ následovaná hranou $(\pi(v), v)$ nejkratší cestou z s do v .

Důkaz:

1. v je nedosažitelný z s : Z lemmatu 3.3: $d[v] \geq \delta(s, v) = \infty$ vrcholy zařazené do Q mají $d[v] < \infty$, proto pro v se nesmí provést řádek 14 a v nebude objeven³.
2. $V_k := \{v \in V : \delta(s, v) = k\}$ Indukcí vzhledem ke k : $\forall v \in V_k$ se právě na jednom místě v průběhu výpočtu stane:

- (a) v je obarven na šedou (GRAY),
- (b) $d[v] \leftarrow k$,
- (c) pro $v \neq s$, $\pi(v)$ nabývá hodnoty u pro něj $u \in V_{k-1}$,
- (d) v je zařazen na konec fronty.

Zřejmě k (a) – (d) dochází maximálně jednou. Indukce:

- $k = 0$: $V_k = \{s\}$, provede se v krocích 5. - 8.
- Ind. krok: $Q = \emptyset$ jen po zastavení s výjimkou inicializace. Pokud v je zařazeno do Q , $d[v]$ a $\pi[v]$ se již nikdy nemění. Jsou-li vrcholy zařazeny do Q v pořadí (v_1, \dots, v_r) ⁴, platí, že $d[v_i] \leq d[v_{i+1}]$ pro $i = 1, \dots, r-1$ (Lemma 3.4: monotonie).

Nechť $v \in V_k$, $k \geq 1$. Tento vrchol je objeven až po zařazení všech vrcholů z V_{k-1} do Q , neboť $\underbrace{d[v] \geq k}_{\text{Lemma 3.3, I.P.}}$

Poněvadž $\delta(s, v) = k$, \exists cesta s, \dots, u, v , $u \in V_{k-1}$, $(u, v) \in E$. Vezmeme to u , které zešedlo nejdříve. V jistém okamžiku se u musí objevit v čele fronty Q . Vrchol v je objeven při prohlídce sousedů vrcholu u . Byl-li objeven, pak jedou řádky 13, 14, 15, 16.

Konečně pro $v \in V_k$, je $\pi[v] \in V_{k-1}$; tedy tvrzení o nejkratší cestě. □

Definice 3.6

Nechť $G = (V, E)$, $s \in V$, $\pi : V \rightarrow V \cup \{\text{NIL}\}$. Pak mohu definovat: $G_\pi = (V_\pi, E_\pi)$, kde:

- $V_\pi := \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$,
- $E_\pi := \{(\pi[v], v) : v \in V_\pi - \{s\}\}$.

Tento graf se nazývá *BF-strom*, pokud V_π je množina všech vrcholů G dosažitelných z s a jestliže $\forall v \in V_\pi \exists!$ $s-v$ cesta v G_π a ta je jedna z nejkratších $s-v$ cest v G .

³t.j. přebarven na GRAY.

⁴toto není aktuální stav fronty!

Lemma 3.7

Algoritmus BFS (3.1) aplikován na $G = (V, E)$, $s \in V$ sestrojí π takové, že G_π je BF-strom.

Důkaz: π s hodnotou různou od NIL se určí pouze na řádce 15: zde $\pi[v] \leftarrow u$ pro $(u, v) \in E$, $\delta(s, v) < \infty$. Tedy V_π je množina všech vrcholů dosažitelných z s . Tedy G_π je strom, neboť:

- G_π je souvislý (Věta 3.5)
- $|E_\pi| = |V_\pi| - 1$

a proto pro každý vrchol v existuje právě jedna $s - v$ cesta a ta je nejkratší cestou v G (Věta 3.5). □

Algoritmus PRINT_PATH(G, s, v) (alg. 3.2) vytiskne cestu z kořene s do vrcholu v na základě relace π , kterou produkuje algoritmus BREADTH_FIRST_SEARCH(G, s) (alg. 3.1). Je lineární vzhledem k počtu vrcholů na $s - v$ cestě.

Algoritmus 3.2

PRINT_PATH(G, s, v)

1. if $v = s$
2. then PRINT s
3. else if $\pi[v] = \text{NIL}$
4. then PRINT "No path from s to v exists"
5. else PRINT_PATH($G, s, \pi[v]$)
6. PRINT v

3.2 Prohledávání do hloubky**3.2.1 Algoritmus prohledávání do hloubky**

Prohledávání do hloubky je další ze základních problémů, kterými se budeme zabývat v přednášce Grafové algoritmy.

Algoritmus 3.3

DEPTH_FIRST_SEARCH(G)

1. for each vertex $u \in V[G]$ do
2. color[u] \leftarrow WHITE
3. $\pi[u] \leftarrow \text{NIL}$
4. time $\leftarrow 0$
5. for each vertex $u \in V[G]$ do
6. if color[u] = WHITE then
7. DFS_VISIT(u)

DFS_VISIT(u)

1. color[u] \leftarrow GRAY
2. $d[u] \leftarrow \text{time} \leftarrow \text{time} + 1$
3. for each vertex $v \in \text{Adj}[u]$ do
4. if color[v] = WHITE then
5. $\pi[v] \leftarrow u$
6. DFS_VISIT(v)
7. color[u] \leftarrow BLACK
8. $f[u] \leftarrow \text{time} \leftarrow \text{time} + 1$

Vysvětlivky: $V[G]$ je množina vrcholů grafu G , $\text{Adj}[u]$ je množina všech sousedů vrcholu u , $\text{color}[u]$ je barva vrcholu u , $\pi[u]$ je předek vrcholu u , proměnná time „krokuje“ barevné změny vrcholů, $d[u]$ je čas, kdy vrchol u zešednul a $f[u]$ je čas, kdy vrchol u zčernal.

Pár slov k algoritmu 3.3 DEPTH_FIRST_SEARCH (dále DFS). Řádky 1. – 4. v algoritmu DFS provedou inicializaci a v řádcích 5. – 7. se provede pro každý vrchol, pokud je bílý, algoritmus DFS_VISIT. Algoritmus DFS_VISIT rekurzivně vybere nějakou „větev“ v grafu G .

Složitost algoritmu DFS:

- **Sekce DFS:**

- ★ Řádky: 1. – 3. vezmou čas $\Theta(V)$,

- ★ Řádek: 4. vezme čas $\Theta(1)$,

- ★ Řádky: 5. – 7. vezmou čas $\Theta(V)$ (v kroku 7. se bere pouze čas, který se spotřebuje vyvoláním sekce DFS_VISIT.)

- **Sekce DFS_VISIT:** Je vyvolána právě jednou pro každý vrchol $v \in V$.

- ★ Řádky: 1. – 2. vezmou čas $\Theta(1)$,

- ★ Řádky: 3. – 6. se provedou $|\text{Adj}[u]|$ -krát (v řádku 6. opět jen vyvolání sekce DFS_VISIT), celkem tedy:

$$\sum_{u \in V} |\text{Adj}[u]| = \Theta(E)$$

- ★ Řádky: 7. – 8. jsou provedeny v konstantním čase — $\Theta(E)$.

- **Celkem:**

$$\Theta(V) + \Theta(1) + |V| \cdot (\Theta(1) + \Theta(1)) + \Theta(E) = \Theta(V + E)$$

Definice 3.8

Grafem předchůdců nazveme $G_\pi = (V, E_\pi)$, kde

$$E_\pi := \{(\pi[v], v) \mid v \in V, \pi[v] \neq \text{NIL}\}$$

Graf předchůdců je orientovaný. Platí, že $u = \pi[v] \iff \text{DFS_VISIT}(v)$ bylo vyvoláno v průběhu průzkumu seznamu sousedů vrcholu u . G_π odpovídá rekurzivnímu volání DFS_VISIT \implies je to les.

Věta 3.9 ZÁVORKOVÁ VĚTA (PARENTHESIS THEOREM)

Při DFS grafu G pro libovolné vrcholy $u, v \in V$, $u \neq v$ nastává právě jedna z možností:

1. intervaly $[d[u], f[u]]$ a $[d[v], f[v]]$ jsou disjunktní,
2. $d[v] < d[u] < f[u] < f[v]$ a u „je potomek“⁵ vrcholu v v grafu G_π ,
3. $d[u] < d[v] < f[v] < f[u]$ a v „je potomek“ vrcholu u v grafu G_π .

Důkaz: Bez újmy na obecnosti⁶ předpokládejme, že $d[u] < d[v]$.

- Dále předpokládejme, že $d[v] < f[u]$, tedy když v šedne, tak u je již šedé. Všechny hrany z v jsou prozkoumány dříve (prohledávání z v je skončeno dříve) než hrany z u ; tedy také $f[v] < f[u]$ a platí možnost číslo 3 z věty.
- Předpokládejme ještě, že $f[u] < d[v]$, tedy prohledávání z vrcholu u bylo ukončeno dříve, než se začalo s prohledáváním z vrcholu $v \implies$ možnost číslo 1 z věty.

□

⁵ \equiv existuje orientovaná cesta

⁶Známý německý matematik Bruno BÚNO první formuloval matematickou frází „Bez Újmy Na Obecnosti“

Poznámka 3.10

Proč se této větě 3.9 říká „závorková“? Označíme-li zešednutí vrcholu x zápisem $(x$ a zčernání téhož vrcholu $x)$ a budeme-li zaznamenávat veškeré změny barev v průběhu DFS, dostaneme zápis typu:

$$\left(s \left(z \left(y \left(x \ x \right) y \right) \left(w \ w \right) z \right) s \right) \left(t \left(v \ v \right) \left(u \ u \right) t \right)$$

Důsledek 3.11

v je potomek $u \iff d[u] < d[v] < f[v] < f[u]$.

Věta 3.12 VĚTA O BÍLÉ CESTĚ (WHITE PATH THEOREM)

V grafu G_π ⁷ je vrchol v potomkem vrcholu $u \iff$ v okamžiku $d[u]$ existuje v G $u - v$ cesta ze samých bílých vrcholů (s výjimkou vrcholu u).

Důkaz:

- \Rightarrow : Nechť v je potomek u , w ($w \neq u$) vrchol na $u - v$ cestě v grafu G_π . Podle důsledku 3.11 je $d[u] < d[w]$. Proto w je v okamžiku $d[u]$ bílý (WHITE).
- \Leftarrow : Nechť v grafu G existuje $u - v$ cesta z bílých vrcholů (mimo u) v okamžiku $d[u]$, ale v není potomek u v grafu G_π . Lze předpokládat, že ostatní vrcholy na této cestě jsou potomky u . Naše cesta je (u, \dots, w, v) (může být $u = w$). Důsledek závorkové věty dá $f[w] \leq f[u]$. Dále $d[u] < d[v] < f[w]$, protože jsme v okamžiku $d[u]$ a vrchol v je bílý. Závorková věta dá $d[u] < d[v] < f[v] < f[u]$ a podle jejího důsledku (důsl. 3.11) je vrchol v potomkem vrcholu u , což je spor.

□

3.2.2 Klasifikace hran grafu po proběhnutí DFS

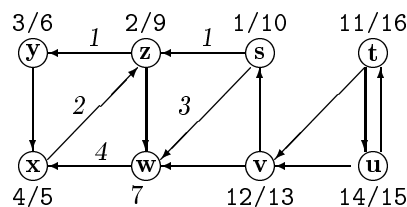
Nechť graf G je orientovaný graf po provedení DFS. Pak hrany tohoto grafu dělíme do těchto skupin:

- *stromové hrany*: hrany z grafu G_π ,
- *zpětné hrany*: hrany $(u, v) \in E$, které splňují podmínku, že vrchol u je potomkem vrcholu v , a smyčky $(u, u) \in E$,
- *přímé hrany*: hrany $(u, v) \in E$, kde vrchol v je potomek u , ale hrana $(u, v) \notin E_\pi$,
- „*cross*“ hrany: hrany $(u, v) \in E$ takové, které nejsou ani stromové ani zpětné ani přímé.

Cvičení: Modifikujte algoritmus DFS tak, aby označil hrany grafu G takto: stromové hrany bílou barvou, zpětné hrany šedou, přímé a „cross“ hrany černou barvou.

Řešení: v řádku 3 sekce DFS_VISIT algoritmu DFS přidáme obarvení hrany podle aktuální barvy koncového vrcholu:

$$\text{edgecolor}[u, v] \leftarrow \text{color}[v]$$



Obrázek 3: Příklad na klasifikaci hran grafu po proběhnutí DFS

⁷Tento graf nazýváme DF-les

Cvičení: Na obrázku 3 je příklad orientovaného grafu, po proběhnutí algoritmu DFS. Uvnitř kružnic jsou identifikátory uzlů (**boldfaced**) a nad nimi resp. pod nimi jsou uvedeny vždy dvojice čísel (číslo1/číslo2), které udávají pro uzel u časy $d[u]/f[u]$. U některých hran jsou čísla (*slanted*), která určují klasifikaci těchto hran:

- 1 jsou hrany stromové
- 2 jsou hrany zpětné
- 3 jsou hrany přímé
- 4 jsou cross hrany

Ostatní hrany (tzn. bez čísla) nechtě si laskavý čtenář ohodnotí příslušným číslem jako cvičení.

Věta 3.13

V neorientovaném grafu G je libovolná hrana stromová nebo zpětná.

Důkaz: Nechtě $(u, v) \in E$, BÚNO mějme $d[u] < d[v]$. Pak $d[v] < f[v] < f[u]$ a hrana (u, v) byla objevena jedním z následujících způsobů:

1. $v \in \text{Adj}[u]$, pak (u, v) je stromová.
2. $u \in \text{Adj}[v]$, pak je v tomto okamžiku vrchol u šedý a tedy hrana (u, v) je zpětná.

□

3.2.3 Topologické třídění

DFS je vhodný k topologickému uspořádání orientovaného grafu.

Definice 3.14

Topologické uspořádání orientovaného grafu je lineární uspořádání $<$ na V , které splňuje:

$$(u, v) \in E \implies u < v$$

Takové uspořádání lze sestavit právě u acyklických grafů⁸ následujícím algoritmem.

Algoritmus 3.4

TOPOLOGICAL_SORT(G)

1. Call DEPTH_FIRST_SEARCH(G) to compute finishing times $f[v]$ for each vertex v
2. As each vertex is finished, insert it onto the front of a linked list
3. Return the linked list of vertices

Ilustrativně lze říci, že můžeme mít graf oblečení, jak je nakreslen na obr. 4, který nám udává, mezi kterými částmi oblečení musíme při oblékání zachovat správné pořadí (např. musíme obléci ponožky před botami). Algoritmus topologického třídění nám však všechny části oblečení uspořádá do takového pořadí, že požadované budou zachovány a přitom postup při oblékání bude jednoznačný.

3.3 Silně souvislé komponenty grafu

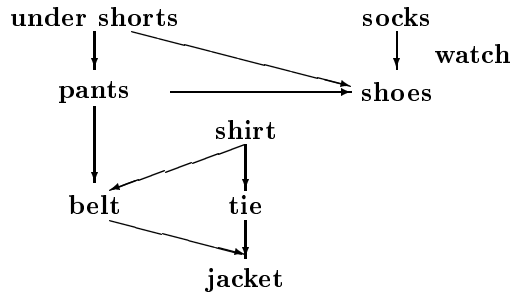
Definice 3.15

Nechtě $G = (V, E)$ je orientovaný graf. *SC-komponenta* (*silně souvislá komponenta*) grafu G je maximální $U \subseteq V$ taková, že $\forall u, v \in U$ existuje $u - v$ cesta i $v - u$ cesta.

Definice 3.16

Cesta v orientovaném grafu: (v_1, \dots, v_k) , $k \geq 1$, přičemž $(v_i, v_{i+1}) \in E$, pro $\forall i = 1, \dots, k - 1$. $u - u$ cesta je délky 0, i když $(u, u) \notin E$.

⁸acyklický graf = graf bez kružnic



Obrázek 4: Příklad grafu topologicky nesetříděného

Definice 3.17

V orientovaném grafu definujme relaci $\sim: u \sim v \iff \exists u - v$ cesta i $v - u$ cesta v grafu G . Relace \sim je ekvivalence, její třídy jsou právě SC-komponenty.

Algoritmus SCC (alg. 3.5), který konstruuje SC-komponenty, vyvolá dvakrát algoritmus DFS (alg. 3.3).

Definice 3.18

Nechť G je orientovaný graf. Definujme graf $G^T := (V, E^T)$, kde $E^T := \{(v, u) \mid (u, v) \in E\}$.

Poznámka 3.19

Je-li G dáno seznamy sousedů, lze seznam sousedů pro G^T vytvořit v čase $o(V + E)$.

Algoritmus 3.5

SC_COMPONENTS(G)

1. Call DEPTH_FIRST_SEARCH(G) to compute finishing times $f[u]$ for each vertex $u \in V$.
2. Compute G^T .
3. Call DEPTH_FIRST_SEARCH(G^T), but in the main loop of DEPTH_FIRST_SEARCH consider the vertices in order of decreasing $f[u]$ (as computed in line 1)
4. Output the vertices of each tree in DF-forest of step 3 as a separate SC-components

Lemma 3.20

Nechť platí $u \sim v$, $u \rightarrow \dots \rightarrow w \rightarrow \dots \rightarrow v$ v grafu G . Pak platí také $w \sim u$.

Důkaz: Máme $u \rightarrow \dots \rightarrow w \rightarrow \dots \rightarrow v$ a $u \sim v$. Tedy existuje ještě orientovaná cesta $v \rightarrow \dots \rightarrow u$. Pak ovšem existují orientované cesty $u \rightarrow \dots \rightarrow w$ a $w \rightarrow \dots \rightarrow v \rightarrow \dots \rightarrow u$ a tedy $w \sim u$. \square

Lemma 3.21

DFS aplikovaný na G umístí vrcholy téže SC-komponenty do téhož DF-stromu (tj. rozklad na SCC je jemější nebo rovný rozkladu na DF-stromy).

Důkaz: Nechť U je SC-komponenta, r je první objevený vrchol v U . V okamžiku $d[r]$ je libovolný vrchol $s \in U$ bílý. Podle Věty O bílé cestě (věta 3.12) se s stane potomkem $r \implies$ tedy patří do téhož DF-stromu. \square

Definice 3.22

Položíme $\Phi(u) = w$, existuje-li $u - w$ cesta v grafu G a $f[w]$ je maximální. Vrchol w nazveme *praotec* vrcholu u .

Lemma 3.23

Platí:

1. $f[u] \leq f[\Phi(u)]$,

2. $u \rightarrow \dots \rightarrow v \implies f[\Phi(v)] \leq f[\Phi(u)]$,
3. $\Phi(\Phi(u)) = \Phi(u)$.

Důkaz:

1. Kdyby platila opačná relace ($f[u] > f[\Phi(u)]$), bylo by to ve sporu s definicí Φ .
2. Nechť $u \rightarrow \dots \rightarrow v$, pak $\{w \mid u \rightarrow \dots \rightarrow w\} \supseteq \{w \mid v \rightarrow \dots \rightarrow w\}$
3. $f[\Phi(\Phi(u))] \geq f[\Phi(u)]$ plyne z bodu 1. Víme, že existuje cesta $\Phi(u) \rightarrow \dots \rightarrow u$ a z bodu 2 dostaneme nerovnost \leq . Tedy bod 3 platí.

□

Lemma 3.24

Užitím DFS pro G dostáváme, že pro $\forall u \in V$ je $\Phi(u)$ předek vrcholu u .

Důkaz:

- Příklad $\Phi(u) = u$ je triviální.
- Nechť tedy $\Phi(u) \neq u$. Uvažujme barvy vrcholů v čase $d[u]$:
 1. $\Phi(u)$ je černý. Pak $f[\Phi(u)] < f[u] \implies$ spor s bodem 1. Lemmatu 3.23. Protože $u \rightarrow \dots \rightarrow \Phi(u)$ a u zčerná až po tom, co zčerná $\Phi(u)$.
 2. $\Phi(u)$ je šedý. Pak u je potomek $\Phi(u)$.
 3. $\Phi(u)$ je bílý:
 - (a) $\underbrace{u \rightarrow \dots \rightarrow \Phi(u)}_{\text{WHITE}}$
Podle Věty o bílé cestě (3.12) je $\Phi(u)$ potomek u , proto $f[\Phi(u)] < f[u]$ (potomci umřou dříve než předci), což je spor s bodem 1 Lemmatu 3.23.
 - (b) $u \rightarrow \dots \rightarrow \underbrace{t \rightarrow \dots \rightarrow \Phi(u)}_{\text{WHITE}}$, vrchol t je poslední vrchol na $u - \Phi(u)$ cestě, který není bílý:
 - ★ t černý: nemůže existovat cesta do bílého $\Phi(u) \implies$ spor.
 - ★ t šedý: pak máme cestu $t \rightarrow \dots \rightarrow \underbrace{\Phi(u)}_{\text{WHITE}}$ a použijeme Větu o bílé cestě (3.12): $\Phi(u)$ je potomek t , neboli $f[\Phi(u)] < f[t]$ (potomci umřou dříve), což je spor s definicí $\Phi(u)$.

□

Důsledek 3.25

Užitím DFS pro G je $(\forall u \in V) (u \sim \Phi(u))$.

Důkaz: $u \rightarrow \dots \rightarrow \Phi(u)$ z definice $\Phi(u)$.

$\Phi(u) \rightarrow \dots \rightarrow u$ z Lemmatu 3.24.

□

Lemma 3.26

Užitím DFS pro G dostáváme, že pro $\forall u, v \in V$ platí:

$$(u \sim v \iff \Phi(u) = \Phi(v))$$

Důkaz:

- \implies : $u \sim v$ dá $\{w \mid u \rightarrow \dots \rightarrow w\} = \{w \mid v \rightarrow \dots \rightarrow w\}$ a tedy $\Phi(u) = \Phi(v)$.
- \impliedby : z důsledku 3.25.

□

Věta 3.27

$\text{SCC}(G)$ korektně spočítá SC-komponenty grafu G .

Důkaz: Indukcí vzhledem k počtu k DF–stromů vytvořených algoritmem DFS na G^T . Ukážeme, že tyto stromy jsou SC–komponenty grafu G .

- Indukční předpoklad: $k = 0$ je zřejmý.
- Indukční krok: $k \geq 1$, je spočítáno $k - 1$ stromů, které jsou SC–komponentami grafu G . Nechť k -tý strom T má kořen r . Označíme

$$C(r) := \{v \in V \mid \Phi(v) = r\}$$

Podle Lemmatu 3.26 je to SC–komponenta (je-li neprázdná). Ukážeme, že

$$u \in T \iff u \in C(r)$$

★ \supseteq : prvky $C(r)$ patří do téhož stromu: viz Lemma 3.26 a Lemma 3.21. Zbývá ukázat, že $r \in C(r)$ neboli $\Phi(r) = r$, což plyne z bodu 3 Lemmatu 3.23. Tedy $T \supseteq C(r)$.

★ \subseteq : ukážeme, že pro $\forall w \in V$ platí:

$$(f[\Phi(w)] \neq f[r] \implies w \notin T)$$

Dále: různé časy zčernání \implies různé vrcholy. Tedy:

$$\begin{aligned} \Phi(w) \neq r &\implies w \notin T \\ w \notin C(r) &\implies w \notin T \\ w \in T &\implies w \in C(r) \end{aligned}$$

- Nechť $f[\Phi(w)] < f[r]$. Kdyby bylo $w \in T$, bylo by $w \rightarrow \dots \rightarrow r$ v G ($w \leftarrow \dots \leftarrow r$ v G^T), proto $f[r] = f[\Phi(r)] \leq f[\Phi(w)]$ (z bodu 2 Lemmatu 3.23), tj. spor.
- Nechť $f[\Phi(w)] > f[r]$. Když r je vybráno, w je již umístěno ve stromě s kořenem $\Phi(w)$. Zde je to sestupné pořadí.

□

4 Užitečné datové struktury

4.1 Binární halda

Definice 4.1

Binární halda je pole, které může být znázorněno jako úplný binární strom. Všechna patra takového stromu – mimo nejnižšího – jsou plná, nejnižší patro je zaplněno zleva. Binární halda splňuje též vlastnost

$$A[\text{PARENT}(i)] \geq A[i]$$

kde $\text{PARENT}(i)$ je rodič uzlu i . Tuto vlastnost budeme nazývat *haldová vlastnost*.

Strukturu desetiprvkové binární haldy ukazuje obrázek 5. Čísla v uzlech této haldy označují index prvku pole A , který má být v uzlu uložen.

Na binární haldě rovněž můžeme definovat jednoduché operace:

- $\text{PARENT}(i)$, která vrací rodiče uzlu i . Lze ji nadefinovat jediným příkazem:

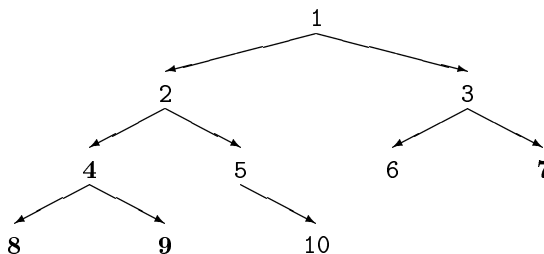
$$\text{return}(\lfloor \frac{i}{2} \rfloor)$$

- $\text{LEFT}(i)$, která vrací rodiče uzlu i . Lze ji nadefinovat:

$$\text{return}(2 \cdot i)$$

- $\text{RIGHT}(i)$, která vrací rodiče uzlu i . Lze ji nadefinovat:

$$\text{return}(2 \cdot i + 1)$$



Obrázek 5: Struktura binární haldy

Algoritmus 4.1

HEAPIFY(A, i)

přel. jako „haldování“

1. $l \leftarrow \text{LEFT}(i)$
2. $r \leftarrow \text{RIGHT}(i)$
3. if $l \leq \text{heap_size}[A]$ and $A[l] > A[i]$
4. then $\text{largest} \leftarrow l$
5. else $\text{largest} \leftarrow i$
6. if $r \leq \text{heap_size}[A]$ and $A[r] > A[\text{largest}]$
7. then $\text{largest} \leftarrow r$
8. if $\text{largest} \neq i$ then
9. exchange $A[i] \leftrightarrow A[\text{largest}]$

10. HEAPIFY(A , largest)

Atribut `heap_size[A]` udává počet prvků v haldě A . V dalším si tento počet označíme n .

Předpokládejme, že binární stromy s kořeny $\text{LEFT}(i)$ a $\text{RIGHT}(i)$ splňují haldovou vlastnost (t.j. $A[i]$ může být menší než $A[\text{LEFT}(i)]$ resp. $A[\text{RIGHT}(i)]$). Cílem algoritmu HEAPIFY (alg. 4.1) je binární strom s kořenem i , který má haldovou vlastnost.

Složitost algoritmu 4.1: Zjištění vztahů mezi $A[i]$, $A[\text{LEFT}(i)]$, $A[\text{RIGHT}(i)]$ je složitosti $\Theta(1)$. K tomu musíme přičíst složitost HEAPIFY na některém synovi, což je kořen stromu o $\leq \frac{2n}{3}$ vrcholech. Nechť je počet potomků roven m . Pak:

$$\begin{aligned} \frac{m^2}{n} &= \frac{1 + 2 + \dots + 2^{k-2} + 2^{k-1}}{1 + 2 + \dots + 2^{k-1} + 1 + 1 + 2 + \dots + 2^{k-2}} \\ &= \frac{2^k - 1}{2^k - 1 + 1 + 2^{k-1} - 1} \\ &\stackrel{?}{\leq} \frac{2}{3} \end{aligned}$$

Nerovnost označená otazníkem („?“) platí \iff platí $\forall k$:

$$\begin{aligned} 3 \cdot 2^k - 3 &\leq 2 \cdot 2^k + 2^k - 2 \\ -1 &\leq 0 \end{aligned}$$

Tedy pro dobu výpočtu $T(n)$ platí:

$$T(n) \leq T\left(\left\lceil \frac{2}{3} \cdot n \right\rceil\right) + \Theta(1)$$

Věta 4.2 MASTER THEOREM

Nechť jsou $a \geq 1$, $b > 1$ konstanty, $f: \mathbf{N}_0 \rightarrow \mathbf{R}$ funkce a nechť je $T: \mathbf{N}_0 \rightarrow \mathbf{R}$ definováno vztahem:

$$T(n) = a \cdot T\left(\left\lceil \frac{n}{b} \right\rceil\right) + f(n)$$

Pak:

1. Pro $f(n) = O(n^{\log_b a - \epsilon})$ pro konstantu $\epsilon > 0$ je

$$T(n) = \Theta(n^{\log_b a})$$

2. Pro $f(n) = \Theta(n^{\log_b a})$ je

$$T(n) = \Theta(n^{\log_b a} \cdot \log n)$$

3. Pro $f(n) = \Omega(n^{\log_b a + \epsilon})$ pro konstantu $\epsilon > 0$ a $f\left(\left\lceil \frac{n}{b} \right\rceil\right) \leq c \cdot f(n)$ pro konstantu $c < 1$ a dostatečně velká n , je

$$T(n) = \Theta(f(n))$$

Tedy – pokračujeme-li v úvahách o složitosti: $a = 1$, $b = \frac{3}{2}$, $\log_b a = 0$. Podle případu 2 tedy

$$T(n) = \Theta(\log n)$$

Složitost algoritmu 4.1 je tedy:

$$T(n) = O(\log n)$$

protože uvažujeme horní odhad složitosti.

Jiný odhad složitosti alg. 4.1: Pro uzel i hloubky h je složitost $O(h)$, protože vezme konstantní čas a vyvolá se pro vrchol s hloubkou o 1 menší.

Algoritmus 4.2

```

BUILD_HEAP(A)                                     # vytvoření haldy
1. heap_size[A] ← length[A]
2. for i ← ⌊length[A]/2⌋ downto 1
3.     do HEAPIFY(A,i)

```

Pro tento algoritmus je dáno pole $A[1, 2, \dots, n]$ a algoritmus uspořádá toto pole tak, že splňuje haldovou vlastnost.

Korektnost: prvky $A[\lceil \frac{n}{2} + 1 \rceil, \dots, n]$ jsou listy a jsou tedy kořeny hald s haldovou vlastností. Algoritmus HEAPIFY zachovává haldovou vlastnost.

Složitost:

- provedení každého HEAPIFY ... $O(\log n)$
- HEAPIFY voláme celkem ... $O(n)$ -krát
- Celkem tedy složitost algoritmu 4.2 ... $O(n \log n)$.

Je však znám lepší odhad $O(n)$, ale k jeho odvození je třeba si více pohlédnout.

Lemma 4.3

n -prvková binární úplná halda má $\lceil \frac{n}{2^{h+1}} \rceil$ vrcholů hloubky h .

Důkaz: Důkaz necháváme jako cvičení. □

Složitost (pokračování):

$$\begin{aligned}
 &\leq \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \cdot O(h) = O\left(n \cdot \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^{h+1}}\right) \\
 &= O\left(n \cdot \underbrace{\sum_{h=0}^{\infty} \frac{h}{2^{h+1}}}_2\right) \\
 &= O(n)
 \end{aligned}$$

Trik, který jsme použili při sčítání řad, je následující:

$$\begin{aligned}
 \sum_{k=0}^{\infty} x^k &= \frac{1}{1-x}, \quad |x| < 1 \quad / \text{derivace} \\
 \sum_{k=0}^{\infty} k \cdot x^{k-1} &= \frac{1}{(1-x)^2} \quad / \cdot x \\
 \sum_{k=0}^{\infty} k \cdot x^k &= \frac{x}{(1-x)^2} \quad / x \rightarrow 2 \\
 \sum_{k=0}^{\infty} \frac{k}{2^k} &= \frac{\frac{1}{2}}{\left(\frac{1}{2}\right)^2} = 2
 \end{aligned}$$

Algoritmus 4.3HEAPSORT(A)

1. BUILD_HEAP(A)
2. for $i \leftarrow \text{length}(A)$ do
3. exchange $A[1] \leftrightarrow A[i]$
4. heap_size[A] \leftarrow heap_size[A] $- 1$
5. HEAPIFY($A, 1$)

Algoritmus HEAPSORT (alg. 4.3) uspořádá binární haldu neklesajícím způsobem.

Složitost:

- BUILD_HEAP ... $O(n)$
- HEAPIFY ... $O(\log n)$
- cyklus se provede ... $O(n)$ -krát
- tělo cyklu stojí ... $\Theta(1)$ ⁹
- Celkem je tedy složitost algoritmu 4.3 HEAPSORT ... $O(n \cdot \log n)$

4.2 Prioritní fronta**Definice 4.4**

Prioritní fronta je datová struktura S , v níž je každý její prvek ohodnocen *klíčem*. Prioritní fronta definuje tyto operace:

- INSERT(S, x) ... provede zařazení prvku x do prioritní fronty S .
- MAXIMUM(S) ... vrátí prvek z prioritní fronty S s největším klíčem.
- EXTRACT_MAX(S) ... odstraní prvek s největším klíčem z fronty S .

Prioritní fronty můžeme implementovat jako:

- binární haldy
- binomiální haldy
- Fibonacciho haldy

Složitost jednotlivých operací implementovaných těmito způsoby ukazuje tabulka 1.

Poznámky k tabulce 1: U binárních a binomiálních hald je uvedena složitost nejhoršího případu. U Fibonacciho hald je to tzv. *amortizovaná složitost*.

ad 1. Operace MAKE_HEAP vytvoří haldu, která neobsahuje žádný prvek.

ad 5. Operace UNION(H_1, H_2)¹⁰ vytvoří sjednocení dvou hald H_1 a H_2 .

ad 6. Operace DECREASE_KEY(H, x, k) sníží klíč vrcholu x v haldě H na hodnotu k

ad 7. Operace DELETE(H, x) odstraní z haldy H vrchol x .

⁹Do složitosti těla cyklu uvažujeme jen vyvolání algoritmu HEAPIFY, nikoli však jeho běh

¹⁰Při implementaci prioritní fronty jako binární haldy lze tuto operaci nadefinovat například pomocí BUILD__HEAP. Takto má však tato operace velkou složitost, což je argument PRO používání binomiální resp. Fibonacciho haldy

Operace		binární h.	binomická h.	Fibonacciho h.
1.	MAKE_HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
2.	INSERT	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)$
3.	MINIMUM	$\Theta(1)$	$O(\log n)$	$\Theta(1)$
4.	EXTRACT_MIN	$\Theta(\log n)$	$O(\log n)$	$O(\log n)$
5.	UNION	$O(n)$	$O(\log n)$	$\Theta(1)$
6.	DECREASE_KEY	$O(\log n)$	$O(\log n)$	$\Theta(1)$
7.	DELETE	$O(\log n)$	$O(\log n)$	$O(\log n)$

Tabulka 1: Operace prioritní fronty a složitost jejich implementací

4.3 Prioritní fronta implementovaná binární haldou

Algoritmy uvedené v této kapitole jsou implementací základních operací nad prioritní frontou pomocí binární haldy.

Algoritmus 4.4

HEAP_MAXIMUM(A)

1. return $A[1]$

Složitost algoritmu 4.4: je zřejmě $\Theta(1)$.

Algoritmus 4.5

HEAP_EXTRACT_MAX(A)

1. if $\text{heap_size}[A] < 1$
2. then error "heap_underflow"
3. $\text{max} \leftarrow A[1]$
4. $A[1] \leftarrow A[\text{heap_size}[A]]$
5. $\text{heap_size}[A] \leftarrow \text{heap_size}[A] - 1$
6. HEAPIFY($A, 1$)
7. return max

Složitost algoritmu 4.5: Kroky 1. – 5. a krok 7. mají složitost $\Theta(1)$, algoritmus HEAPIFY (krok 6.) má složitost $O(\log n)$.

Celkem tedy má algoritmus 4.5 HEAP_EXTRACT_MAX složitost $O(\log n)$.

Algoritmus 4.6

HEAP_INSERT(A, key)

1. $\text{heap_size}[A] \leftarrow \text{heap_size}[A] + 1$
2. $i \leftarrow \text{heap_size}[A]$
3. while $i > 1$ and $A[\text{PARENT}(i)] < key$ do
4. $A[i] \leftarrow A[\text{PARENT}(i)]$
5. $i \leftarrow \text{PARENT}(i)$
6. $A[i] \leftarrow key$

Složitost algoritmu 4.6: Kroky 1. – 3. a krok 6. mají složitost $\Theta(1)$, kroky 4. a 5. mají také složitost $\Theta(1)$, ale provedou se celkem $O(\log n)$ -krát.

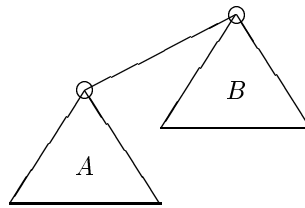
Celková složitost algoritmu 4.6 HEAP_INSERT je tedy $O(\log n)$.

4.4 Binomiální halda

Definice 4.5

Binomiální strom B_k definujeme induktivně takto:

- B_0 je strom o jednom vrcholu
- Máme-li dva binomiální stromy B_{k-1} označené např. A, B , pak binomiální strom B_k vznikne z těchto dvou např. tak, že kořen stromu B přijme za syna kořen stromu A . Názorně viz obr. 6.



Obrázek 6: Příklad konstrukce binomiálního stromu B_k

Upozorníme na fakt, že pořadí vrcholů v dané úrovni binomiálního stromu je podstatné. Binomiální strom B_k má 2^k vrcholů.

Definice 4.6

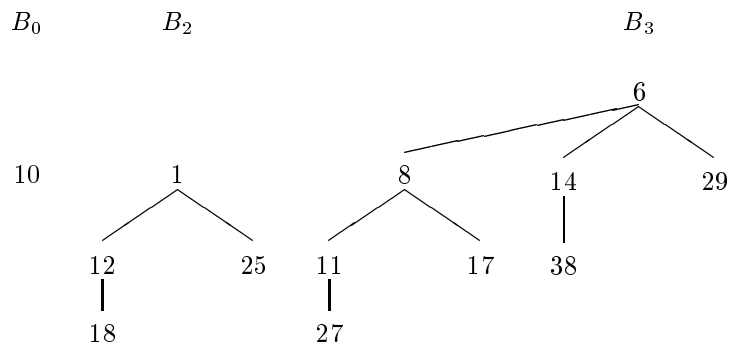
Binomiální halda je systém binomiálních stromů B_k (závisí na pořadí) takový, že stromy splňují haldovou vlastnost, přičemž k roste zleva doprava.

Do binomiální haldy lze umístit libovoný počet n prvků, protože existuje jednoznačný binomiální rozklad čísla n :

$$n = |B_{k_1}| + |B_{k_2}| + \dots + |B_{k_p}|$$

přičemž platí $k_1 < k_2 < \dots < k_p$.

Příklad binomiální haldy ukazuje obrázek 7.



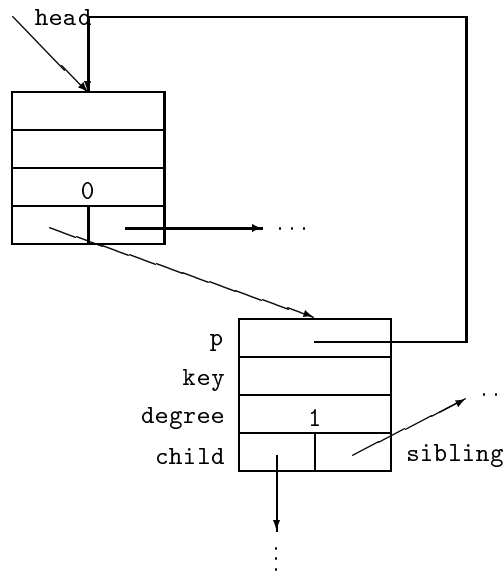
Obrázek 7: Příklad binomiální haldy

4.5 Prioritní fronta implementovaná binomiální haldou

Při reprezentaci binomiálních hald budeme používat těchto datových struktur:

- $p[x]$... ukazatel na otce vrcholu x .
- $child[x]$... ukazatel na nejlevějšího potomka.
- $sibling[x]$... ukazatel na sourozence bezprostředně vpravo.
- $degree[x]$... stupeň¹¹ vrcholu x .
- $head[H]$... ukazatel na první vrchol haldy H .

Demonstrace těchto datových struktur je na obr. 8.



Obrázek 8: Datové struktury pro reprezentaci binomiální haldy

V této přednášce si nebudeme uvádět přesná znění algoritmů realizujících operace na prioritní frontě implementované binomiální haldou. Uvedeme si zde pouze jejich seznam (tab. 2) s tím, že zájemci je mohou najít pod těmito názvy v literatuře [3].

	Název algoritmu	Počet řádků
1.	BINOMIAL_HEAP_MINIMUM(H)	9
2.	BINOMIAL_LINK(y, z)	4
3.	BINOMIAL_HEAP_UNION(H_1, H_2)	22
4.	BINOMIAL_HEAP_MERGE(H_1, H_2)	–
5.	BINOMIAL_HEAP_INSERT(H, x)	7
6.	BINOMIAL_HEAP_EXTRACT_MIN(H)	5
7.	BINOMIAL_HEAP_DECREASE_KEY(H, x, k)	10
8.	BINOMIAL_HEAP_DELETE(H, x)	2

Tabulka 2: Seznam algoritmů, realizujících operace nad binomiální haldou

Poznamenejme ještě, že BINOMIAL_LINK provede operaci, která je popsána na obrázku 6, kde y a z jsou kořeny spojovaných stromů. BINOMIAL_HEAP_MERGE je procedura v algoritmu BINOMIAL_HEAP_UNION, která spojí seznamy kořenů H_1 a H_2 do seznamu, kde atribut degree neklesá.

¹¹Stupněm vrcholu zde rozumíme úroveň tohoto vrcholu v daném binomiálním stromu

4.6 Analýza složitosti

Složitost můžeme porovnávat podle:

- nejhoršího případu
- průměrného případu, t.j. zadá se pravděpodobnostní rozdělení na jednotlivých možných datech a spočítá se střední hodnota potřebného času
- *amortizovaná* složitost

O amortizované složitosti se nyní rozhovoříme podrobněji.

Dejme tomu, že máme provést posloupnost operací s datovou strukturou:

$$D_0 \xrightarrow{c_1} D_1 \xrightarrow{c_2} D_2 \xrightarrow{c_3} \dots \xrightarrow{c_n} D_n$$

Nechť $\Phi : D \mapsto \Phi(D) \in \mathbf{R}$ je funkce potenciálu.

Amortizovaný čas pro i -tou operaci je

$$\hat{c}_i := c_i + \Phi(D_i) - \Phi(D_{i-1})$$

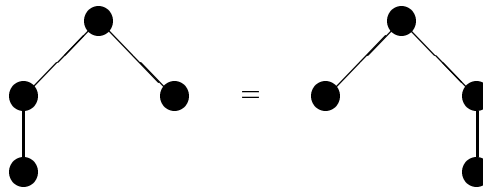
tedy součet aktuálního času a vzrůstu potenciálu.

$$\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n c_i + \Phi(D_n) - \Phi(D_0)$$

Je-li tedy $\Phi(D_n) \geq \Phi(D_0)$, pak $\sum \hat{c}_i$ je horním odhadem pro $\sum c_i$. Proto můžeme čas v průběhu operace omezit na $\frac{1}{n} \cdot \sum \hat{c}_i$.

4.7 Fibonacciho haldy

Nechť U_k je neorientovaný k -tý binomiální strom (přičemž nezáleží na pořadí slučování binomiálních stromů, tedy rovnost na obr. 9 narozdíl od binomiální haldy platí).



Obrázek 9: Topologická rovnost binomiálních stromů u Fibonacciho hald

Definice 4.7

Fibonacciho halda je systém neuspořádaných neorientovaných binomiálních stromů.

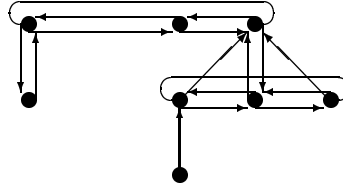
Datová struktura pro reprezentaci Fibonacciho hald:

- $p[x]$... ukazatel na otce.
- $child[x]$... ukazatel na některého syna (libovolného).
- $left[x]$... ukazatel na levého souseda vrcholu x , který má téhož otce jako x .
- $right[x]$... pravá obdoba ukazatele $left[x]$.

- $\text{degree}[x]$... stupeň.
- $\text{mark}[x]$... nabývá hodnot $\{\text{true}, \text{false}\}$, přičemž význam hodnoty true je:

$$\text{mark}[x] = \text{true} \iff \text{vrchol } x \text{ ztratil nějakého potomka od doby, kdy se sám naposledy stal potomkem jiného vrcholu.}$$

Příklad Fibonacciho haldy je zobrazen na obr. 10.



Obrázek 10: Příklad Fibonacciho haldy

Pro výpočet příslušné amortizované složitosti operací nad Fibonacciho haldou použijeme tuto potenciálovou funkci:

$$\Phi(H) = t(H) + 2m(H)$$

kde $t(H)$ je počet stromů v haldě H a $m(H)$ je počet označených vrcholů (t.j. vrcholů x , kde $\text{mark}[x] = \text{true}$) v haldě H . Použití této potenciálové funkce je korektní, protože platí: $\Phi(D_0) = 0$ a $\forall k \quad \Phi(D_k) \geq 0$

Fibonacciho haldy mohou být libovolně široké, proto lze dělat operaci UNION nad Fibonacciho haldami v konstantním čase.

4.8 Zásobník

Na zásobníku definujeme operace:

- $\text{POP}(S)$ vyzvedne prvek ze zásobníku.
- $\text{PUSH}(S)$ uloží prvek do zásobníku.
- $\text{MULTIPOP}(S, k)$ vyzvedne k prvků ze zásobníku (viz alg. 4.7).

Algoritmus 4.7

$\text{MULTIPOP}(S, k)$

1. while not $\text{STACK_EMPTY}(S)$ and $k \neq 0$ do
2. $\text{POP}(S)$
3. $k \leftarrow k - 1$

Amortizovaná složitost: Definujeme potenciálovou funkci zásobníku: $\Phi(S)$ jako počet prvků v zásobníku S . Začínáme od prázdného zásobníku D_0 , tedy $\Phi(D_0) = 0$. Pro $\forall i$ tedy platí $\Phi(D_i) \geq \Phi(D_0)$.

Odtud plyne, že celková amortizovaná cena je horním odhadem celkové aktuální ceny:

- PUSH zvětší potenciál Φ zásobníku o jedničku:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 1 = 2$$

- POP sníží potenciál Φ zásobníku o jedničku:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 - 1 = 0$$

- MULTIPOP sníží potenciál Φ o $k' = \min\{k, s\}$, $s = |S|$. Necht $\Phi(D_i) - \Phi(D_{i-1}) = -k'$, pak

$$\hat{c}_i = c_i - k' = k' - k' = 0$$

4.9 Datové struktury pro disjunktní množiny

Datové struktury pro disjunktní množiny uchovávají systém $\mathcal{S} = \{S_1, \dots, S_k\}$ po dvou disjunktních dynamických množin; v každé z těchto množin je určen reprezentant této množiny.

Operace:

- $\text{MAKE_SET}(x)$... vytvoří systém $\{\{x\}\}$.
- $\text{UNION}(x, y)$... sjednotí množiny obsahující x resp. y .
- $\text{FIND_SET}(x)$... vrátí reprezentanta množiny, v níž leží x .

Algoritmus 4.8

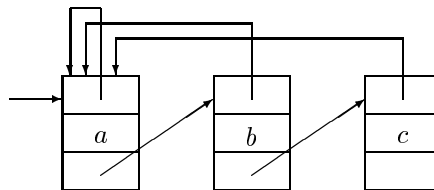
$\text{CONNECTED_COMPONENTS}(G)$

algoritmus vyžaduje na vstupu obyčejný (neorientovaný) graf G .

1. for each vertex $v \in V[G]$
2. do $\text{MAKE_SET}(v)$
3. for each edge $(u, v) \in E[G]$
4. do if $\text{FIND_SET}(u) \neq \text{FIND_SET}(v)$
5. then $\text{UNION}(u, v)$

4.9.1 První implementace

Zde pohovoříme o složitosti algoritmu 4.8 $\text{CONNECTED_COMPONENTS}$, implementovaného pomocí *linked list* reprezentace. Podle této reprezentace je každá množina reprezentována dynamickým (řetězeným) seznamem, v němž navíc každý prvek obsahuje ukazatel na reprezentanta množiny. Příklad najdete na obr. 11.



Obrázek 11: Příklad množiny $\{a, b, c\}$ v reprezentaci linked list

Složitost:

- n -krát se provádí MAKE_SET (kroky 1. – 2. algoritmu 4.8).
- m -krát jsou prováděny všechny tři operace dohromady
- platí: $m \geq n$, počet operací $\text{UNION} \leq n - 1$.

Fakt: Složitost algoritmu 4.8 je tedy při implementaci linked list reprezentací je $\Theta(m^2)$.

Fakt: Při použití heuristiky „váženého sjednocování“ (tzn. připojuji kratší seznam za delší, nikoli naopak) je složitost algoritmu 4.8 $O(m + n \log n)$.

4.9.2 Druhá implementace

Zde pohovoříme o složitosti algoritmu 4.8 CONNECTED_COMPONENTS, implementovaného pomocí *lesů disjunkt-ních množin*. Každou množinu reprezentuje jeden strom, kde relace otec \rightarrow syn má „opačnou orientaci“, tj. otec \leftarrow syn. Příklad si inteligentní čtenář vymyslí sám.

Fakt: Složitost při této implementaci je $O(m \log^* n)$, použijeme-li heuristiku „union by rank path compression“. Přitom:

$$\log^{(i)} n = \begin{cases} n, & i = 0 \\ \log(\log^{(i-1)} n), & i > 0, \ln^{(i-0)} > 0 \\ \text{nedefinováno,} & \text{jinak} \end{cases}$$

Dále:

$$\log^* n = \min \{ i \geq 0 \mid \log^{(i)} n \leq 1 \}$$

Příklad, jak vypadá funkce \log^* uvádí tabulka 3.

n	$\log^* n$
2	1
4	2
16	3
65536	4
2^{65536}	5

Tabulka 3: Chování funkce \log^*

Hodnota $\log^* n \geq 5$ se v našem případě nevyskytne. Tedy složitost této implementace je „v podstatě lineární“.

Lepší odhad: $O(m \cdot \alpha(m, n))$, kde α je tzv. *Ackermannova funkce* a hodnota $\alpha(m, n)$ pro v našem případě myslitelné situace nepřevyší hodnotu 4.

5 Problém minimální kostry

5.1 Obecný algoritmus

Definice 5.1

Nechť je $G = (V, E)$ souvislý neorientovaný graf a $w : E \rightarrow \mathbf{R}$ libovolné. Pak *problém minimální kostry* je problém nalezení $T \subseteq E$ tak, aby platilo:

1. T je acyklický
2. (V, T) je souvislý
3. $w(T)$ je minimální, kde

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

Poprvé byl tento problém vyřešen v roce 1926 panem Borůvkou, který zkonstruoval první algoritmus tohoto druhu na zakázku pro MEZ. V současnosti známe dva nejlepší algoritmy pánů

- Kruskala
- Prina

které mají složitost $O(E \log V)$ pomocí binárních hald; Prinův algoritmus implementovaný pomocí Fibonacciho hald má složitost $O(E + V \log V)$.

Následující tabulka (tab. 4) nám přesněji ukazuje, jak velký podíl na celkové složitosti algoritmu má použitá datová struktura. V tabulce použité symboly jsou definovány: $n = |V|$, $m = |E|$ ¹².

Graf	Poměr hran	$m \log n$	$m + n \log n$	Porovnání
řídký graf	$m \sim n$	$n \log n$	$n \log n$	=
hustý graf	$m \sim n^2$	$n^2 \log n$	n^2	≠

Tabulka 4: Porovnání složitosti algoritmů při použití různých datových struktur

Algoritmus 5.1

GENERIC_MINIMAL_SPANNING_TREE(G, w)

1. $A \leftarrow \emptyset$
2. while A does not form a spanning tree do
3. find an edge (u, v) that is safe for A
4. $A \leftarrow A \cup \{(u, v)\}$
5. return A

Algoritmus 5.1 nám ukazuje jakousi „kostru“ algoritmu na vyhledávání minimální kostry grafu. A je množina hran, do které přidáváme „bezpečné“ hrany (takové hrany, které můžeme přidat bez obav, že by nám množinu A porušily tak, že bychom se nikdy k minimální kostře nedostali, blíže následující definice 5.2), dokud se nedobereme minimální kostry. Množina A bude mít vždy vlastnost, že existuje minimální kostra T obsahující prvky A .

Definice 5.2

Hrana $(u, v) \in E$ se nazývá *bezpečná pro A* , platí-li $(u, v) \notin A$ a zároveň sjednocení $A \cup \{(u, v)\}$ je obsaženo v některé minimální kostře grafu $G = (V, E)$.

Korektnost algoritmu 5.1 je zřejmá. Zatím však neznáme způsob, jak se vybírá nová hrana (u, v) .

¹²Pro m tedy platí: $n - 1 \leq m \leq \binom{n}{2}$.

Definice 5.3

Rozklad $(S, V - S)$ množiny V se nazývá řez grafu G , přičemž $\emptyset \neq S \neq V$.

Definice 5.4

Hrana $(u, v) \in E$ protíná řez $(S, V - S)$, je-li $u \in S$ a $v \in V - S$ nebo naopak.

Definice 5.5

Řez $(S, V - S)$ respektuje množinu $A \subseteq E$, jestliže žádná hrana z A neprotíná tento řez.

Definice 5.6

Lehká hrana řezu $(S, V - S)$ je hrana protínající tento řez a mezi takovými hranami má minimální ohodnocení.

Věta 5.7

Nechť je $G = (V, E)$ souvislý neorientovaný graf, $w : E \rightarrow \mathbf{R}$ libovolné ohodnocení hran, $A \subseteq E$ je obsažena v nějaké minimální kostře grafu G , $(S, V - S)$ je řez respektující A a (u, v) je lehká hrana pro tento řez. Pak (u, v) je bezpečná pro A .

Důkaz: Nechť A je obsažena v minimální kostře T . Pokud $(u, v) \in T$, jsme hotovi.

Nechť $(u, v) \notin T$. Existuje u - v cesta v T . u a v jsou v různých částech řezu a proto na výše uvedené cestě existuje hrana (x, y) taková, že $x \in S$ a $y \in V - S$. Ovšem $(x, y) \notin A$, protože řez respektuje A . Odstraněním (x, y) z T se T rozpadne do dvou souvislých komponent. Přidáním (u, v) máme opět kostru, kterou označíme $T' = (T - \{(x, y)\}) \cup \{(u, v)\}$. (u, v) i (x, y) protínají řez $(S, V - S)$. (u, v) je lehká $\implies w(u, v) \leq w(x, y)$ a proto $w(T') \leq w(T)$. T byla minimální kostra $\implies T'$ je minimální kostra. (u, v) je bezpečná pro A , neboť $(u, v) \notin A$ a $A \cup \{(u, v)\} \subseteq T'$. \square

Důsledek 5.8

$G = (V, E)$ je souvislý neorientovaný graf, $w : E \rightarrow \mathbf{R}$, $A \subseteq E$ je obsažena v nějaké minimální kostře, C je souvislá komponenta v lese $G_A = (V, A)$, (u, v) minimálně ohodnocená mezi hranami spojujícími C s jinou komponentou. Pak (u, v) je bezpečná pro A .

Důkaz: Uvažujme řez $(C, V - C)$ a můžeme použít týchž úvah jako v předešlém důkaze. \square

5.2 Kruskalův algoritmus

A je stále les, přidává se hrana (u, v) těchto vlastností:

1. $(u, v) \notin A$ a $A \cup \{(u, v)\}$ je les;
2. $w(u, v)$ je minimální mezi všemi hranami, které splňují 1.

Algoritmus 5.2

MST_KRUSKAL(G, w)

1. $A \leftarrow \emptyset$
2. for each vertex $v \in V[G]$ do
3. MAKE-SET(v)
4. sort the edges of G by nondecreasing weight w
5. for each edge $(u, v) \in E[G]$ in order of nondecreasing weight w do
6. if FIND_SET(u) \neq FIND_SET(v) then
7. $A \leftarrow A \cup \{(u, v)\}$
8. UNION(u, v)
9. return A

K implementaci algoritmu 5.2 potřebujeme efektivní datovou strukturu pro rozklady množiny vrcholů V a operace na ní:

- FIND_SET(u): tato operace vrátí reprezentanta třídy obsahující prvek u .
- UNION(u, v): sjednotí třídy obsahující prvky u a v .
- MAKE_SET(u): vytvoří singleton¹³ $\{u\}$.

Korektnost algoritmu 5.2:

- 1. ř. – 3. ř.: inicializace
 $A = \emptyset$, rozklad $\{\{v\} \mid v \in V\}$.
- 5. ř. – 8. ř.: zaručují plnění předpokladů důsledku 5.8.

Složitost závisí na implementaci rozkladu množin:

- 1. ř. – 3. ř.: $O(V)$
- 4. ř.: $O(E \log E)$
- 9. ř.: $O(V)$
- 5. ř. – 8. ř.: $O(E)$ operací s rozkladu lze implementovat tak, že každá vezme $O(\log E)$, celkem tedy $O(E \log E)$, přesně $O(E) \cdot (O(1) + O(\log E))$. Celkem tedy:

$$O(V) + O(E \log E) = O(E \log E)$$

Pravá strana rovnosti platí za předpokladu, že graf je souvislý, čili $|E| \geq |V| - 1$.

5.3 Prinův algoritmus

A je stále strom, přidává se hrana (u, v) s těmito vlastnostmi:

1. $(u, v) \notin A$ a $A \cup \{(u, v)\}$ je strom;
2. mezi všemi hranami, které splňují 1, je $w(u, v)$ minimální.

Algoritmus 5.3

MST_PRIN(G, w, r)

1. $Q \leftarrow V[G]$
2. for each vertex $u \in Q$ do
3. $\text{key}[u] \leftarrow \infty$
4. $\text{key}[r] \leftarrow 0$
5. $\pi[r] \leftarrow \text{NIL}$ /* π je otec vrcholu */
6. while $Q \neq \emptyset$ do
7. $u \leftarrow \text{EXTRACT_MIN}(Q)$
8. for each $v \in \text{Adj}[u]$ do
9. if $v \in Q$ and $w(u, v) < \text{key}[v]$ then
10. $\pi[v] \leftarrow u$
11. $\text{key}[v] \leftarrow w(u, v)$

Vysvětlení symbolů z algoritmu 5.3:

- Q ... prioritní fronta vrcholů.

¹³singletonem $\{a\}$ budeme rozumět jednoprvkovou množinu $\{a\}$

- $A = \{(v, \pi(v)) \mid v \in V - (\{r\} \cup Q)\}$
- $\text{key}[u] \dots$ je ohodnocení hrany, která má nejmenší ohodnocení mezi hranami spojujícími A s vrcholem u . $\text{key}[u] \in \mathbf{R} \cup \{\infty\}$.
- $\pi[u] \in V \cup \{\text{NIL}\} \dots$ otec vrcholu u .
- $r \dots$ kořen stromu.
- operace $\text{EXTRACT_MIN}(Q) \dots$ vyřadí z Q takový vrchol u , jehož $\text{key}[u]$ je minimální.

Korektnost algoritmu 5.3:

- 1.ř. – 5.ř. inicializace
- Kdykoli v průběhu algoritmu: vrcholy aktuálního stromu A jsou přesně ty, které patří do rozdílu $V - Q$.
- na řádce 7 s výjimkou prvního průchodu (což je odstranění kořene r z Q) se do A přidává lehká hrana pro řez $(V - Q, Q)$.

5.4 Složitost algoritmů

5.4.1 Složitost Kruskalova algoritmu

- 1. ř. $\dots O(1)$.
- 2. – 3. ř. $\dots O(V)$.
- 4. ř. $\dots O(E \log E)$.
- cyklus 5. – 8. ř. \dots se provádí $|E|$ -krát. Provádím tedy $O(E)$ operací na datové struktuře pro disjunktní množiny S . Lze implementovat tak, aby to vzalo $O(E \cdot \alpha(E, V))$. Jsou-li $|V| = n$ a $|E| = m$, pak $n + 2m + m \doteq O(E)^{14}$.
- 9. ř. $\dots O(1)$.

$$O(\alpha(E, V)) = O(\log E) \implies O(E \log E) = O(E \log V)$$

5.4.2 Složitost Primova algoritmu

Nechť Q je implementována pomocí binární haldy.

- 1. ř. \dots BUILD_HEAP a řádky 2. – 5. celkem $O(V)$.
- cyklus 7. – 11. se provádí $|V|$ -krát, každé EXTRACT_MIN vezme $O(\log V)$. Celkem 7. ř. vezme $O(V \log V)$
- cyklus 8. – 11. se provádí celkem $O(E)$ -krát.
 - ★ 9. ř. \dots rezervuje bit u každého vrcholu (dáme 1 v řádce 1; dáme 0 v řádce 7), tedy $O(V)$.
 - ★ 11. ř. \dots DECREASE_KEY vezme $O(\log V)$
 - ★ 9. – 11. ř. $\dots O(1) + O(\log V) = O(\log V)$
 - ★ cyklus 8. – 11. tedy celkem: $O(E) \cdot (O(1) + O(\log V)) = O(E \log V)$. $O(1)$ je na organizaci cyklu.
- Celkem je $O(E \log V)$.

Složitost algoritmu implementovaného pomocí Fibonacciho hald je $O(E + V \log V)$.

¹⁴Je to po řadě za: MAKE_SET, FIND_SET a UNION.

6 Nejkratší cesty

6.1 Nejkratší cesty z daného vrcholu

Nechť je $G = (V, E)$ orientovaný graf, $w : E \rightarrow \mathbf{R}$ hranové ohodnocení.

Definice 6.1

Váha cesty $w(p)$, kde $p = (v_1, \dots, v_k)$ se definuje jako $w(v_1, v_2) + \dots + w(v_{k-1}, v_k)$.
Jsou-li $u, v \in V$, pak

$$\delta(u, v) = \begin{cases} \min \{w(p) \mid p \text{ je } u - v \text{ cesta}\} & , \text{ pokud existuje } u - v \text{ cesta} \\ \infty & , \text{ jinak} \end{cases}$$

je váha nejkratší cesty mezi vrcholy u a v .

Nejkratší cesta je $u - v$ cesta p , pro niž platí, že $w(p) = \delta(u, v)$.

Základní problém: Nechť je $G, w, s \in V$ dáno. Úkolem je najít pro každý vrchol $v \in V$ příslušnou hodnotu $\delta(s, v)$.

Modifikace základního problému:

1. Dáno $G, w, s \in V$. Za úkol máme najít pro $\forall v \in V$ příslušnou hodnotu $\delta(v, s)$.

Řešení jsou dvě:

- přeorientovat hrany a aplikovat algoritmy pro základní problém
- dualizovat algoritmy pro základní problém

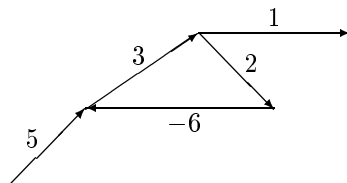
2. Je dáno G, w, s a t . Úkolem je najít $\delta(s, t)$. Tento problém lze řešit algoritmy základního problému¹⁵. Nutno však na tomto místě poznamenat, že žádný lepší algoritmus není znám.

3. Je dáno G, w . Úkolem je najít $\delta(u, v)$ pro $\forall u, v \in V$. Řešení:

- Triviálně lze tento problém řešit pomocí algoritmů pro základní problém. Postupně budeme brát všechny vrcholy $v \in V$ za s a aplikujeme tyto algoritmy.
- Nutno podotknout, že existují rafinovanější a lepší metody. O nich budeme hovořit v odstavci 6.7.

Poznamenejme ještě, že některé algoritmy (např. Dijkstra) připouštějí pouze nezáporná ohodnocení hran.

Algoritmy, které připouštějí libovolné ohodnocení hran, musí počítat s tím, že nejkratší cesta nemusí existovat kvůli existenci záporného cyklu¹⁶. Příklad záporného cyklu uvádí obr. 12. Jsou však algoritmy například Bellman – Fordův, které záporný cyklus odhalí.



Obrázek 12: Příklad záporného cyklu

¹⁵Tzn. nalézt δ pro všechny vrcholy v grafu a na výstup dát jen $\delta(s, t)$.

¹⁶Na takovém cyklu je možné natočením příslušného počtu obrátek snížit váhu cesty na libovolně malou hodnotu

Reprezentace nejkratších cest: Necht' máme libovolné $\pi : V \rightarrow V \cup \{\text{NIL}\}$. Zobrazení π nám určuje graf $G_\pi = (V_\pi, E_\pi)$, kde

$$\begin{aligned} V_\pi &= \{v \in V \mid \pi[v] \neq \text{NIL}\} \cup \{s\} \\ E_\pi &= \{(\pi[v], v) \in E \mid v \in V_\pi - \{s\}\} \end{aligned}$$

Definice 6.2

Strom nejkratších cest $G' = (V', E')$ grafu G je graf, kde $V' \subseteq V$, $E' \subseteq E$ a V' je množina vrcholů grafu G dosažitelných z vrcholu s (\equiv existuje $s - v$ cesta v grafu G). G' je strom s kořenem s . Pro $\forall v \in V$ existuje jediná $s - v$ cesta v grafu G' a je nejkratší $s - v$ cestou v grafu G .

Lemma 6.3

Necht' (v_1, \dots, v_k) je nejkratší $v_1 - v_k$ cesta v grafu G a necht' $1 \leq i \leq j \leq k$. Pak (v_i, \dots, v_j) je nejkratší $v_i - v_j$ cesta v tomto grafu.

Důkaz: Zřejmý. □

Lemma 6.4

Necht' $p = (s, \dots, u, v)$ je nejkratší $s - v$ cesta. Pak platí:

$$\delta(s, v) = \delta(s, u) + w(u, v)$$

Důkaz: Z lemmatu 6.3 plyne, že $p' = (s, \dots, u)$ je nejkratší $s - u$ cesta, proto

$$\begin{aligned} \delta(s, v) = w(p) &= w(p') + w(u, v) = \\ &= \delta(s, u) + w(u, v) \end{aligned}$$

□

Lemma 6.5

Pro $\forall (u, v) \in E$ platí nerovnost:

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

Důkaz: Je zřejmý, stačí si uvědomit, že levá strana nerovnosti je délka nejkratší $s - v$ cesty, kdežto pravá strana je délka nějaké $s - v$ cesty. □

6.2 Relaxace

Algoritmus 6.1

INITIALIZE_SINGLE_SOURCE(G, s)

1. for each vertex $v \in V[G]$ do
2. $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow \text{NIL}$
4. $d[s] \leftarrow 0$

$d[v] \in \mathbf{R} \cup \{\infty, -\infty\}$ je horní odhad délky nejkratší $s - v$ cesty.

Po proběhnutí algoritmu jsou nastaveny potřebné datové struktury takto:

- $\pi[v] = \text{NIL}$ pro $\forall v \in V$
- $d[s] = 0$ a $d[v] = \infty$ pro $\forall v \in V - \{s\}$

Algoritmus 6.2

RELAX(u, v, w)

1. if $d[v] > d[u] + w(u, v)$ then

2. $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

Algoritmus 6.2 RELAX provede tzv. *relaxaci*. Relaxace se tomuto jevu říká proto, že podmínka $d[v] \leq d[u] + w(u, v)$, která musí být splněna pro $d[v] = \delta(s, v)$ a $d[u] = \delta(s, u)$ nás přestává „tížit“.

Všechny algoritmy na vyhledávání nejkratších cest pracují tak, že se provede:

- inicializace (algoritmus 6.1)
- posloupnost relaxací (algoritmus 6.2)

Jednotlivé algoritmy se od sebe liší pouze tím, jakým způsobem se relaxace vybírají k uskutečnění.

Lemma 6.6

Po provedení algoritmu RELAX(u, v, w) platí:

$$d[v] \leq d[u] + w(u, v)$$

Důkaz: zřejmý. □

Lemma 6.7

Nechť jsme provedli algoritmus 6.1 INITIALIZE_SINGLE_SOURCE(G, s) a posloupnost operací RELAX(u_i, v_i, w) (alg. 6.2). Pak $d[v] \geq \delta(s, v)$ pro $\forall v \in V$ a pokud $d[v]$ někdy v průběhu nabylo hodnoty $\delta(s, v)$, pak se již v dalším průběhu hodnota $d[v]$ nezmění.

Důkaz:

- Po inicializaci platí: $d[s] = 0 \geq \delta(s, s)$, protože $\delta(s, s) \in \{0, -\infty\}$, a pro $\forall v \in V - \{s\}$ je $d[v] = \infty \geq \delta(s, v)$.
- Nechť podmínka $d[v] \geq \delta(s, v)$ je poprvé porušena provedením RELAX(u, v, w). Pak platí:

$$\begin{aligned} d[u] + w(u, v) = d[v] &< \delta(s, v) \leq \\ &\stackrel{\text{Lemma 6.5}}{\leq} \delta(s, u) + w(u, v) \end{aligned}$$

tedy $d[u] < \delta(s, u)$, což je spor, protože poprvé byla tato podmínka splněna až pro vrchol v .

- Jakmile $d[v] = \delta(s, v)$, stále platí $d[v] \geq \delta(s, v)$ a při tom $d[v] > \delta(s, v)$ již nemůže nastat, protože RELAX nezvětšuje hodnoty $d[_]$. □

Důsledek 6.8

Nechť neexistuje $s - v$ cesta, nechť $v \in V$. Pak po inicializaci $d[v] = \delta(s, v)$ a tato rovnost zůstává v platnosti po provedení libovolné posloupnosti relaxací.

Důkaz: Podle lematu 6.7 stále platí $d[v] \geq \delta(s, v) = \infty$. Po inicializaci je $d[v] = \infty$ a tedy stále platí $d[v] = \delta(s, v) = \infty$. □

Lemma 6.9

Nechť (s, \dots, u, v) je nejkratší $s - v$ cesta. Nechť je provedena inicializace a provádíme posloupnost relaxací. Pak bylo-li $d[u] = \delta(s, u)$ někdy před vyvoláním relaxace RELAX(u, v, w), pak je $d[v] = \delta(s, v)$ kdykoliv po provedení této relaxace.

Důkaz: Podle lematu 6.7 je $d[u] = \delta(s, u)$ stále od okamžiku, kdy tato rovnost nastala. Zejména to platí po provedení RELAX(u, v, w).

$$\begin{aligned} d[v] &\stackrel{\text{Lemma 6.6}}{\leq} d[u] + w(u, v) \\ &= \delta(s, u) + w(u, v) \\ &\stackrel{\text{Lemma 6.4}}{=} \delta(s, v) \end{aligned}$$

Lemma 6.6 spolu s lemmatem 6.7 dává ve výše uvedeném odvození rovnost a její zachování v budoucnosti. □

6.3 Stromy nejkratších cest

Lemma 6.10

Nechť graf G s hranovým ohodnocením w neobsahuje žádné záporné cykly dosažitelné z vrcholu s . Pak po provedení inicializace a libovolné posloupnosti relaxací je G_π strom s kořenem s .

Důkaz:

- Po inicializaci: $V_\pi = \{s\}$ a $E_\pi = \emptyset$, což je v pořádku.
- Nechť jsme provedli $q > 1$ relaxací. Ukážeme sporem, že G_π je acyklický:

Připusťme, že $c = (v_0, \dots, v_k)$ přičemž $v_0 = v_k$ je obsažen v grafu G_π , pak $\pi[v_i] = v_{i-1}$ pro $i = 1, \dots, k$ a lze tedy předpokládat, že poslední relaxace byla $\text{RELAX}(v_{k-1}, v_k, w)$.

Pro každý vrchol v cyklu c platí $\pi[v] \neq \text{NIL}$ a byla mu přiřazena konečná hodnota $d[v]$. Podle lematu 6.7 je $\delta(s, v)$ konečná (v je dosažitelný z s). Ukážeme, že cyklus c je záporný cyklus, což bude spor.

Vzhledem k tomu, že $\pi[v_i] = v_{i-1}$ pro $i = 1, \dots, k$, byla poslední změna $d[v_i]$: $d[v_i] \leftarrow d[v_{i-1}] + w(v_{i-1}, v_i)$. Od tohoto okamžiku se eventuálně mohlo $d[v_{i-1}]$ zmenšit. Tedy:

$$d[v_i] \geq d[v_{i-1}] + w(v_{i-1}, v_i)$$

pro $i = 1, \dots, k$. Při vyvolání $\text{RELAX}(v_{k-1}, v_k, w)$, které vytvořilo cyklus c máme

$$d[v_k] > d[v_{k-1}] + w(v_{k-1}, v_k)$$

Sečtením:

$$\begin{aligned} \sum_{i=1}^k d[v_i] &> \sum_{i=1}^k (d[v_{i-1}] + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^k d[v_i] + \sum_{i=1}^k w(v_{i-1}, v_i) \end{aligned}$$

Z výše uvedeného plyne, že:

$$\sum_{i=1}^k w(v_{i-1}, v_i) < 0$$

a tedy cyklus c je záporný cyklus, čímž jsme došli ke kýženému sporu.

Graf G_π je tedy acyklický.

Zbývá tedy dokázat, že $\forall v \in V_\pi \exists!$ $s - v$ cesta v grafu G_π .

- ★ Existence cesty: po inicializaci je existence zřejmá. Dále indukcí: po $\text{RELAX}(u, v, w)$ mohl pouze v přibýt do V_π . Pak $\pi[v] \leftarrow u$ a π se pro ostatní vrcholy nemění.
- ★ Jednoznačnost cesty: Předpokládejme, že $s - v$ cesta není v G_π jednoznačná. Pak vypadá např. takto:

$$s \rightarrow \dots \rightarrow \left\langle \begin{array}{ccc} \rightarrow \dots & \rightarrow x & \rightarrow \\ \rightarrow \dots & \rightarrow y & \rightarrow \end{array} \right\rangle \rightarrow z \rightarrow \dots \rightarrow v$$

Přičemž $x \neq y$. Pak by však muselo platit zároveň $\pi[z] = x$ a $\pi[z] = y$, což samozřejmě nelze a tudíž máme spor.

□

Lemma 6.11

Nechť neexistují v grafu G záporné cykly dosažitelné z vrcholu s . Nechť je provedena inicializace a posloupnost relaxací. Nechť $\forall v \in V$ je $d[v] = \delta(s, v)$. Pak G_π je strom nejkratších cest.

Důkaz: V_π je právě množina všech vrcholů dosažitelných z vrcholu s : vrchol $v \neq s$ je dosažitelný $\iff d[v] < \infty \iff \pi[v] \neq \text{NIL} \iff v \in V_\pi$.

Podle lemmatu 6.10 je G_π strom s kořenem s . Necht' $v \in V_\pi$. Chceme, aby (jediná) $s - v$ cesta v G_π byla nejkratší $s - v$ cesta $p = (v_0 = s, \dots, v_k = v)$ v grafu G .

Pro $i = 1, \dots, k$ platí $d[v_i] = \delta(s, v_i)$ podle předpokladů. Z důkazu lemmatu 6.10 máme $d[v_i] \geq d[v_{i-1}] + w(v_{i-1}, v_i)$, což je po úpravě:

$$w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1})$$

Sečtením pak dostaneme:

$$\begin{aligned} w(p) = \sum_{i=1}^k w(v_{i-1}, v_i) &\leq \sum_{i=1}^k (\delta(s, v_i) - \delta(s, v_{i-1})) \\ &= \delta(s, v_k) - \delta(s, v_0) \\ &= \delta(s, v_k) - \underbrace{\delta(s, s)}_0 \\ &= \delta(s, v_k) \end{aligned}$$

Z definice δ máme $w(p) \geq \delta(s, v)$. Tedy celkem:

$$w(p) = \delta(s, v)$$

a p je nejkratší $s - v$ cesta v grafu G . □

6.4 Dijkstrův algoritmus

Jak jsme si již dříve uvedli, Dijkstrův algoritmus předpokládá, že pro $\forall(u, v) \in E$ platí $w(u, v) \geq 0$.

Algoritmus 6.3

DIJKSTRA(G, w, s)

1. INITIALIZE_SINGLE_SOURCE(G, s)
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V[G]$
4. while $Q \neq \emptyset$ do
5. $u \leftarrow \text{EXTRACT_MIN}(Q)$
6. $S \leftarrow S \cup \{u\}$
7. for each vertex $v \in \text{Adj}[u]$ do
8. RELAX(u, v, w)

S je množina všech vrcholů, pro něž bylo d určeno definitivně. Q je prioritní fronta z vrcholů $V - S$ s klíči danými hodnotami d .

Algoritmus opakovaně vybírá vrchol z $V - S$ s nejmenším klíčem, zařadí jej do S a relaxuje všechny hrany, které z něj vycházejí.

Věta 6.12

Algoritmus 6.3 DIJKSTRA pracuje korektně.

Důkaz: Ukážeme, že $\forall v \in V$ je $d[v] = \delta(s, v)$ v okamžiku přidání vrcholu v do množiny S . Fakt, že tato rovnost zůstává zachována, dokazuje 2. část důkazu lemmatu 6.7.

Sporem: Necht' u je první z vrcholů vložených do S , s vlastností $d[u] \neq \delta(s, u)$ v okamžiku tohoto vložení. Protože vrchol s je zařazen jako první a $d[s] = \delta(s, s) = 0$, nebudeme vrchol s uvažovat; tedy necht' $u \neq s$.

Uvažujme okamžik zahájení řádků 5. – 8. pro vrchol u . V té době je $S \neq \emptyset$. Existuje $s - u$ cesta (kdyby ne, je $\delta(s, u) = \infty$, avšak $d[u] \neq \infty$ a existuje nejkratší $s - u$ cesta). Tedy existuje nejkratší $s - u$ cesta:

$$s \in S \rightarrow \dots \rightarrow x \rightarrow y \rightarrow \dots \rightarrow u \in V - S$$

Nechť y je na této cestě první, který patří do $V - S$. V okamžiku vložení vrcholu x do S je $d[x] = \delta(s, x)$ a po $\text{RELAX}(x, y, w)$ je také $d[y] = \delta(s, y)$ (plyne to z lemmatu 6.9).

Dále z nezápornosti w a z lemmatu 6.3 plyne $\delta(s, y) \leq \delta(s, u)$. Pak tedy:

$$d[y] = \delta(s, y) \leq \delta(s, u) \underbrace{\leq}_{\text{Lemma 6.7}} d[u]$$

Volba vrcholu u nám dává $d[u] \leq d[y]$, z čehož nám plyne:

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

Což je kýžený spor, protože jsme u zvolili tak, aby $d[u] \neq \delta(s, u)$.

Konečně z lemmatu 6.11 plyne, že G_π je strom nejkratších cest. □

Složitost: Nechť $n = |V|$ a $m = |E|$.

1.
 - Je-li Q implementováno jako seznam, pak EXTRACT_MIN vezme $O(n)$. To se provádí n -krát, tedy řádky 4. – 6. vezmou $O(n^2)$.
 - Řádky 7. a 8. vezmou $O(m)$ (bráno zvlášť, tedy „mimo“ cyklu while).
 - Řádky 1. a 3. vezmou každý $O(n)$.
 - Řádek 2. je konstantní přiřazení, tedy $O(1)$.

Podtrženo a sečteno je složitost algoritmu $O(n^2)$, protože velikost m je shora omezena n^2 .

2.
 - Je-li Q implementováno jako binární halda, pak EXTRACT_MIN vezme $O(\log n)$ a řádky 4. – 6. tudíž $O(n \log n)$.
 - Řádek 3. reprezentuje vytvoření binární haldy, tedy složitost je $O(n)$.
 - V řádce 8. musí po $\text{RELAX}(u, v, w)$ následovat $\text{DECREASE_KEY}(Q, v, d[u] + w(u, v))$, což vezme samo o sobě $O(\log n)$, celkem tedy 7. a 8. řádek vezmou $O(m \log n)$.
 - Řádky 1. a 2. dohromady vezmou $O(n)$.

Celková složitost algoritmu je tedy $O((m+n) \log n)$, což pro grafy, v nichž je každý vrchol dosažitelný, představuje složitost $O(m \log n)$, protože pro souvislé grafy je n shora omezeno m .

3. Je-li Q implementováno jako Fibonacciho halda, je celková složitost algoritmu $O(n \log n + m)$.

6.5 Bellman – Fordův algoritmus

Existuje-li záporný cyklus dosažitelný z vrcholu s , Bellman – Fordův algoritmus tuto skutečnost odhalí. Jinak korektně vrátí strom nejkratších cest.

Algoritmus 6.4

$\text{BELLMAN_FORD}(G, w, s)$

1. $\text{INITIALIZE_SINGLE_SOURCE}(G, s)$
2. for $i \leftarrow 1$ to $|V[G]| - 1$ do
3. for each edge $(u, v) \in E[G]$ do
4. $\text{RELAX}(u, v, w)$
5. for each edge $(u, v) \in E[G]$ do
6. if $d[v] > d[u] + w(u, v)$ then
7. return FALSE
8. return TRUE

Složitost:

- 1. řádek ... $O(n)$
- 2. – 4. řádek ... $O(mn)$
- 5. – 8. řádek ... $O(m)$

Celková složitost algoritmu 6.4 je tedy $O(mn)$.

Lemma 6.13

Nechť G vzhledem k w neobsahuje žádný záporný cyklus dosažitelný z s , pak po skončení algoritmu 6.4 platí pro $\forall v \in V$ $d[v] = \delta(s, v)$.

Důkaz:

1. Nechť v je dosažitelný z s . Nechť $p = (v_0 = s, \dots, v_k = v)$ je nějaká nejkratší $s - v$ cesta (s nejmenším ohodnocením). Je možné předpokládat, že v p se neopakují vrcholy. Jinak p obsahuje cyklus:
 - kladný, pak jej mohu z cesty „vyhodit“, cesta p není nejkratší
 - nulový, pak jej mohu z cesty „vyhodit“ a nic se nestane
 - záporný, pak nastává spor s předpokladem

Tedy $k \leq n - 1$.

Indukcí vzhledem k $i = 0, \dots, k$ dokážeme, že rovnost $d[v_i] = \delta(s, v_i)$ po i -tém provedení cyklu 2. – 4. řádek zůstává dále zachována.

$i=0$: Po inicializaci je $d[v_0] = d[s] = \delta(s, s) = \delta(s, v_0) = 0$. Podle lemmatu 6.7 je tato rovnost nadále zachována.

Indukční krok: Nechť $i \geq 1$, $d[v_{i-1}] = \delta(s, v_{i-1})$ po $i - 1$ provedeních cyklu 2. – 4. řádek. V i -tém cyklu se relaxuje hrana (v_{i-1}, v_i) . Podle lemmatu 6.9 je po provedení této relaxace $d[v_i] = \delta(s, v_i)$ a kdykoliv potom.

2. Nechť je vrchol v nedosažitelný. Pak podle důsledku 6.8 platí $d[v] = \delta(s, v) = \infty$.

□

Lemma 6.14

Pro $v \in V$ existuje $s - v$ cesta \iff algoritmus 6.4 končí s hodnotou $d[v] < \infty$.

Důkaz: Pokud v grafu G neexistuje vzhledem k w záporný cyklus dosažitelný z vrcholu s , pak to plyne přímo z lemmatu 6.13 a nemáme co dokazovat.

Indukcí dokážeme, že existuje $s - v$ cesta z méně než i nebo z i hran \iff po i -té iteraci algoritmu 6.4 je $d[v] < \infty$:

$i=0$: existuje $s - v$ cesta z 0 hran $\iff v = s \iff$ po inicializaci se provede $d[v] \leftarrow 0$.

Indukční krok: \implies : 1. Existuje $s - v$ cesta z $\leq i - 1$ hran. Podle indukčního předpokladu je $d[v] < \infty$ a $d[v]$ se relaxacemi nezvyšuje.

2. Neexistuje $s - v$ cesta z $\leq i - 1$ hran, existuje $s - v$ cesta (s, \dots, u, v) z i hran. Po i -té operaci je podle indukčního předpokladu $d[u] < \infty$. Relaxace hrany (u, v) v i -tém cyklu dá $d[v] < \infty$.

\impliedby : 1. $d[v] < \infty$ již po $(i - 1)$ -té iteraci, tvrzení plyne přímo z indukčního předpokladu.

2. $d[v] = \infty$ po $(i - 1)$ -té iteraci a $d[v] < \infty$ po i -té iteraci. Nechť poslední změna $d[v]$ je po relaxaci hrany (u, v) a použitím indukčního předpokladu máme tvrzení.

□

Věta 6.15

Algoritmus 6.4 je korektní. Jinými slovy platí ekvivalence: Algoritmus 6.4 vrátí TRUE \iff v grafu G není vzhledem k w dosažitelný záporný cyklus \iff pro $\forall v \in V$ platí $d[v] = \delta(s, v) \iff G_\pi$ je graf nejkratších cest.

Důkaz: Nechť graf G neobsahuje záporný cyklus dosažitelný z s . Pak:

- $\forall v \in V: d[v] = \delta(s, v)$ plyne z lemmatu 6.13.
- G_π je strom nejkratších cest plyne z lemmatu 6.11.

Zbývá tedy dokázat ekvivalence: Algoritmus 6.4 vrátí TRUE \iff neexistuje záporný cyklus dosažitelný z s .

\Leftarrow : Po skončení cyklu 2. – 4. řádek platí:

$$d[v] = \delta(s, v) \stackrel{\text{Lemma 6.5}}{\leq} \delta(s, u) + w(u, v) \\ = d[u] + w(u, v)$$

z čehož plyne, že testy na řádku 6. jsou všechny negativní.

\Rightarrow : Nechť $c = (v_0, \dots, v_k)$, $v_k = v_0$ je záporný cyklus dosažitelný z s . Pak

$$\sum_{i=1}^k w(v_{i-1}, v_i) < 0$$

Připusťme, že algoritmus 6.4 vrátil TRUE. Pak

$$d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$$

pro $i = 1, \dots, k$. Sečtením máme:

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

Z lemmatu 6.14 plyne, že pro $\forall i = 1, \dots, k$ platí $d[v_i] < \infty$.

Po úpravě dostáváme, že $0 <$ záporné číslo, což je samozřejmě spor.

□

6.6 Nejkratší cesty z jediného vrcholu v acyklických grafech

Algoritmus 6.5

DAG_SHORTEST_PATH(G, w, s)

1. topologically sort the vertices of G
2. INITIALIZE_SINGLE_SOURCE(G, s)
3. for each vertex u taken in topological sorted order do
4. for each vertex $v \in \text{Adj}[u]$ do
5. RELAX(u, v, w)

Složitost:

- Řádek 1. nám zabezpečí, že „hrany vedou jen jedním směrem“. Složitost je $O(m + n)$
- Řádky 2. a 3. vezmou každý $O(n)$.
- Řádky 4. a 5. vezmou dohromady $O(m)$.

Celková složitost algoritmu 6.5 je $O(m + n)$.

Věta 6.16

Je-li G orientovaný acyklický graf, pak algoritmus 6.5 pracuje korektně.

Důkaz: Je-li v nedosažitelný, pak z důsledku 6.8 plyne $d[v] = \delta(s, v) = \infty$.

Nechť $v \in V$ je dosažitelný vrchol z s . Nechť $p = (v_0 = s, \dots, v_k = v)$ je nejkratší $s - v$ cesta a nechť jsou hrany této cesty relaxovány v pořadí: $(v_0, v_1), \dots, (v_{k-1}, v_k)$.

Indukcí vzhledem k $i = 0, \dots, k$ z těchto relaxací ukážeme, že po provedení i -té relaxace je $d[v_i] = \delta(s, v_i)$.

i=0: po inicializaci $d[v_0] = d[s] = \delta(s, s) = \delta(s, v_0) = 0$.

Indukční krok: Nechť $i \geq 1$ a nechť $d[v_{i-1}] = \delta(s, v_{i-1})$ po provedení $(i - 1)$ -té relaxace. Po provedení i -té relaxace je podle lemmatu 6.9 $d[v_i] = \delta(s, v_i)$ a tato rovnost nadále zůstává.

Z lemmatu 6.11 plyne, že G_π je strom nejkratších cest. □

Příklad užití algoritmu 6.5

V lineárním programování se hledá případné řešení diferenčních podmínek, které jsou tvaru: $x_j - x_i \leq b_k$.

Nechť $Ax \leq b$ je systém diferenčních podmínek. Pak jej mohu jednoduše převést na graf tímto způsobem:

- $V := \{v_0, \dots, v_n\}$
- $E := \{(v_i, v_j) \mid x_j - x_i \leq b_k\} \cup \{(v_0, v_1), \dots, (v_0, v_n)\}$

Přitom všechny hrany (v_i, v_j) (kde $i \neq 0$) jsou ohodnoceny příslušným číslem b_k a všechny hrany (v_0, v_i) jsou ohodnoceny nulou.

Věta 6.17

Nechť G neobsahuje záporný cyklus. Pak

$$x = (\delta(v_0, v_1), \dots, \delta(v_0, v_n))$$

je řešením systému diferenčních podmínek $Ax \leq b$. Je-li v grafu G záporný cyklus, nemá $Ax \leq b$ řešení.

Důkaz:

1. Lemma 6.5 dá

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

neboli

$$x_j \leq x_i + b_k$$

2. Nechť G obsahuje záporný cyklus $c = (v_1, \dots, v_k = v_1)$. Zřejmě pro $i = 1, \dots, k$ je $v_i \neq v_0$. Nechť x splňuje $Ax \leq b$, pak jsou splněny tyto rovnice:

$$x_2 - x_1 \leq w(v_1, v_2)$$

$$x_3 - x_2 \leq w(v_2, v_3)$$

...

$$x_1 - x_k \leq w(v_k, v_1)$$

Sečtením dostaneme:

$$0 \leq w(c)$$

a přitom předpokládáme, že $w(c) < 0$ (c je záporný cyklus), což je samozřejmě spor. □

Algoritmus	Složitost
Dijkstra, lineární seznam	$O(n^3)$
Dijkstra, binomiální halda	$O(mn \log n)$
Dijkstra, Fibonacciho halda	$O(n^2 \log n + mn)$
Bellman – Ford	$O(mn^2)$

Tabulka 5: Složitosti triviálních řešení

6.7 Nejkratší cesty all pairs

Nechť je dán orientovaný graf $G = (V, E)$ a ohodnocení hran $w : E \rightarrow \mathbf{R}$. Pro $\forall u, v \in V$ máme najít $\delta(u, v)$, tzn. délku (váhu) nejkratší (vzhledem k w) $u - v$ cesty.

Můžeme ($n = |V|$)-krát použít libovolný algoritmus pro nejkratší cesty z daného vrcholu. Tabulka 5 shrnuje složitosti algoritmů, použitých tímto způsobem.

Existují však rafinovanější algoritmy, než ty, které jsme si už ukázali, aplikované tímto „hrůzným“ způsobem.

Tyto algoritmy používají reprezentaci grafy G maticí incidence.

Vstup: Matice W typu $n \times n$, $W = (w_{ij})$, kde

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

Dále pro zjednodušení nechť množina vrcholů $V = \{1, \dots, n\}$.

Výstup: Matice D typu $n \times n$, $D = (d_{ij})$, kde $d_{ij} = \delta(i, j)$. Pokud chceme i nejkratší cesty, budeme mít další výstup, což bude matice $\Pi = (\pi_{ij})$, kde

$$\pi_{ij} = \begin{cases} \text{NIL} & , \text{ pro } i = j \text{ nebo neexistuje } - \text{ li } i - j \text{ cesta} \\ k & , \text{ jinak} \end{cases}$$

kde k je předchůdce j na nějaké nejkratší $i - j$ cestě.

Pro $i \in V$ definujeme graf $G_{\pi_i} = (V_{\pi_i}, E_{\pi_i})$:

- $V_{\pi_i} := \{j \in V \mid \pi_{ij} \neq \text{NIL}\} \cup \{i\}$
- $E_{\pi_i} := \{(\pi_{ij}, j) \mid j \in V_{\pi_i}, \pi_{ij} \neq \text{NIL}\}$

Tento strom nazveme strom nejkratších cest z vrcholu i .

Algoritmus 6.6

PRINT_ALL_PAIRS_SHORTEST_PATHS(Π, i, j)

1. if $i = j$
2. then print i
3. else if $\pi_{ij} = \text{NIL}$
4. then "No path from i to j "
5. else PRINT_ALL_PAIRS_SHORTEST_PATHS(Π, i, π_{ij})
6. print j

Předpokládejme, že neexistují záporné cykly. Nechť $d_{ij}^{(m)}$ je váha nejkratší $i - j$ cesty přes $\leq m$ hran:

$$d_{ij}^{(0)} := \begin{cases} 0, & i = j \\ \infty, & i \neq j \end{cases}$$

a pro $m \geq 1$:

$$\begin{aligned} d_{ij}^{(m)} &:= \min(d_{ij}^{(m-1)}, \min_{1 \leq k \leq n} (d_{ik}^{(m-1)} + w_{kj})) \\ &\stackrel{w_{jj}=0}{=} \min_{1 \leq k \leq n} (d_{ik}^{(m-1)} + w_{ij}) \end{aligned}$$

Nejkratší $i - j$ cesta obsahuje maximálně $n - 1$ hran, tedy platí $\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = \dots$. Základem je tedy počítat posloupnost matic $W = D^{(1)}, D^{(2)}, \dots, D^{(n-1)}$.

Algoritmus 6.7

EXTEND_SHORTEST_PATH(D, W)

1. $n \leftarrow \text{rows}[D]$
2. let $D' = (d'_{ij})$ be an $n \times n$ matrix
3. for $i \leftarrow 1$ to n do
4. for $j \leftarrow 1$ to n do
5. $d'_{ij} \leftarrow \infty$
6. for $k \leftarrow 1$ to n do
7. $d'_{ij} = \min(d'_{ij}, d_{ik} + w_{kj})$
8. return D'

Řádky 6. a 7. spočítají $\min_k (d_{ik} + w_{kj})$. V řádku 1. znamená zápis $\text{rows}[D]$ přiřazení řádu matice D do proměnné n .

Složitost algoritmu 6.7 je $\Theta(n^3)$.

Algoritmus 6.8

SLOW_ALL_SHORTEST_PATHS(W)

1. $n \leftarrow \text{rows}[W]$
2. $D^{(1)} \leftarrow W$
3. for $m \leftarrow 2$ to $n - 1$ do
4. $D^{(m)} \leftarrow \text{EXTEND_SHORTEST_PATH}(D^{(m-1)}, W)$
5. return $D^{(n-1)}$

Složitost algoritmu 6.8 je $\Theta(n^4)$.

Algoritmus 6.9

FASTER_ALL_PAIRS_SHORTEST_PATHS(W)

1. $n \leftarrow \text{rows}[W]$
2. $D^{(1)} \leftarrow W$
3. $m \leftarrow 1$
4. while $n - 1 > m$ do
5. EXTEND_SHORTEST_PATH($D^{(m)}, D^{(m)}$)
6. $m \leftarrow 2m$
7. return $D^{(m)}$

Na vysvětlenou k řádkům 5. a 6.: Nejkratší $i - j$ cesta přes $\leq 2m$ hran je kompozicí nejkratší $i - k$ cesty přes $\leq m$ hran a nejkratší $k - j$ cesty přes $\leq m$ hran.

Složitost algoritmu 6.9 je $\Theta(n^3 \log n)$.

6.7.1 Floyd – Warshallův algoritmus

Nechť je $d_{ij}^{(k)}$ váha nejkratší $i - j$ cesty přes vrcholy z množiny $\{1, 2, \dots, k\}$. Tedy: $d_{ij}^{(0)} = w_{ij}$ a pro $k \geq 1$ je

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

Zřejmě platí, že $\delta(i, j) = d_{ij}^{(n)}$.

Algoritmus 6.10

FLOYD_WARSHALL(W)

1. $n \leftarrow \text{rows}[W]$
2. $D^{(0)} \leftarrow W$
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. return $D^{(n)}$

Složitost algoritmu 6.10 je $O(n^3)$. Jeho modifikací lze také počítat Π .

6.8 Tranzitivní uzávěr orientovaného grafu**Definice 6.18**

Tranzitivní uzávěr grafu $G = (V, E)$ je graf $G^* = (V, E^*)$, kde $E^* = \{(i, j) \mid \exists i - j \text{ cesta v } G\}$.

Tranzitivní uzávěr grafu G získáme ohodnocením všech hran číslem 1 a aplikací grafu na nějaký algoritmus pro nalezení nejkratších cest „all pairs”. Po skončení takového algoritmu bude platit:

$$\delta(i, j) < \infty \iff (i, j) \in E^*$$

Johnson's algoritmus na sestrojení tranzitivního uzávěru orientovaného grafu má složitost $O(n^2 \log n + mn)$.

7 Maximální toky v sítích

Definice 7.1

Síť je orientovaný graf $G = (V, E)$ spolu s funkcí $c : E \rightarrow \mathbf{R}^{+17}$. $c(u, v)$ je kapacita hrany (u, v) . Obvykle je však c definováno jako $c : V^2 \rightarrow \mathbf{R}^+$, přičemž pro $\forall(u, v) \notin E$ je $c(u, v) = 0$.

Nechť $s, t \in V$. Vrchol s nazýváme *zdroj* a t nazýváme *stok*. Přitom předpokládáme, že $\forall v \in V$ leží na nějaké $s - t$ cestě (pokud ne, lze graf zjednodušit průzkumem).

Definice 7.2

Tok v síti (G, s, t, c) je funkce $f : V \times V \rightarrow \mathbf{R}$ splňující:

1. $\forall u, v \in V$ platí $f(u, v) \leq c(u, v)$
2. $\forall u, v \in V$ platí $f(u, v) = -f(v, u)$
3. $\forall u \in V - \{s, t\}$ platí

$$\sum_{v \in V} f(u, v) = 0$$

Definice 7.3

Velikost toku f je číslo:

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

Problém: pro danou síť najít tok f s maximální velikostí $|f|$ tohoto toku.

Poznámka 7.4

$\forall u \in V$ je $f(u, u) = -f(u, u)$, z čehož plyne $f(u, u) = 0$.

Poznámka 7.5

Bod 3 definice 7.2 je ekvivalentní s tvrzením, že pro $\forall v \in V - \{s, t\}$ platí

$$\sum_{u \in V} f(u, v) = 0$$

Poznámka 7.6

- Nechť $(u, v), (v, u) \notin E$. Pak

$$\left. \begin{array}{l} f(u, v) \leq c(u, v) = 0 \\ f(v, u) \leq c(v, u) = 0 \end{array} \right\} f(u, v) = 0$$

- Nechť $(u, v) \in E$ a $(v, u) \notin E$. Pak

$$\begin{array}{l} 0 \leq f(u, v) \leq c(u, v) \\ f(v, u) \leq c(v, u) = 0 \end{array}$$

- Nechť $(u, v), (v, u) \in E$. Pak

$$-c(v, u) \leq f(u, v) \leq c(u, v)$$

Poznámka 7.7

Nechť $v \in V$. *Positivním tokem do vrcholu v* nazveme číslo:

$$\sum_{u \in V} f(u, v) > 0$$

¹⁷ \mathbf{R}^+ množina nezáporných reálných čísel.

Pozitivním tokem z vrcholu v nazveme číslo:

$$\sum_{\substack{w \in V \\ f(v, w) > 0}} f(v, w)$$

Pak pro $\forall v \in V - \{s, t\}$ můžeme psát:

$$\begin{aligned} 0 = \sum_{w \in V} f(v, w) &= \sum_{\substack{w \in V \\ f(v, w) > 0}} f(v, w) + \sum_{\substack{w \in V \\ f(v, w) < 0}} f(v, w) \\ &= \sum_{\substack{w \in V \\ f(v, w) > 0}} f(v, w) - \sum_{\substack{w \in V \\ f(w, v) > 0}} f(w, v) \end{aligned}$$

Tedy pozitivní tok do v se rovná pozitivnímu toku z v .

Definice 7.8

Nechť $X, Y \subseteq V$. Pak definujeme

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)$$

Lemma 7.9

Nechť f je tok v síti (G, s, t, c) . Pak:

1. pro $X \subseteq V$ je $f(X, X) = 0$.
2. pro $X, Y \subseteq V$ je $f(X, Y) = -f(Y, X)$.
3. pro $X, Y, Z \subseteq V$, $X \cap Y = \emptyset$ je:

$$\begin{aligned} f(X \cup Y, Z) &= f(X, Z) + f(Y, Z) \\ f(Z, X \cup Y) &= f(Z, X) + f(Z, Y) \end{aligned}$$

Důkaz:

- 1: Sčítanci $f(x, x)$ jsou všechny nulové (viz poznámka 7.4) a s každým $f(x, y)$, kde $x \neq y$ obsahuje i $f(y, x) = -f(x, y)$.
- 2 a 3 se dokáže jednoduše rozepsáním.

□

Algoritmus 7.1

FORD_FULKERSON_METHOD(G, s, t)

1. initialize flow f to 0
2. while there exists an augmenting path p
3. do augment flow f along p
4. return f

Definice 7.10

Jestliže je dána síť (G, s, t, c) s tokem f , definujeme *reziduální kapacitu* hrany (u, v) :

$$c_f(u, v) := c(u, v) - f(u, v)$$

Dále definujeme *reziduální síť* $G_f = (V, E_f)$, kde

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

Poznamenejme, že $|E_f| \leq 2|E|$.

Definice 7.11

Zvětšující cesta v síti (G, s, t, c) vzhledem k toku f je jednoduchá¹⁸ $s - t$ cesta v grafu G_f .

Lemma 7.12

Nechť f je tok v síti (G, s, t, c) , nechť f' je tok v síti (G_f, s, t, c_f) . Pak $f + f'$ je tok v původní síti (G, s, t, c) a jeho velikost $|f + f'| = |f| + |f'|$.

Důkaz: Nejprve musíme dokázat, že $f + f'$ je tok. $f + f'$ tedy musí splňovat definiční podmínky:

- 1. podmínka:

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

- 2. podmínka:

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

- 3. podmínka: nechť $u \in V - \{s, t\}$. Pak:

$$\sum_{v \in V} (f + f')(u, v) = \sum_{v \in V} f(u, v) + \sum_{v \in V} f'(u, v) = 0 + 0 = 0$$

Dále musíme ověřit, zda $|f + f'| = |f| + |f'|$. To podle definice velikosti toku však platí. □

Definice 7.13

Reziduální kapacita zvětšující cesty p je:

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

Lemma 7.14

Nechť f je tok v síti (G, s, t, c) a p je zvětšující cesta. Definujeme

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

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

Důkaz: je zřejmý. □

Definice 7.15

Řez je rozklad (S, T) množiny V takový, že $s \in S$ a $t \in T$.

Tok řezem (S, T) je $f(S, T)$.

Kapacita řezu (S, T) je $c(S, T)$.

Lemma 7.16

Nechť f je tok v síti (G, s, t, c) a nechť (S, T) je řez. Pak

$$|f| = f(S, T)$$

¹⁸tzn. taková cesta, v níž se neopakují vrcholy

Důkaz: plyne z lemmatu 7.9:

$$\begin{aligned}
 f(S, T) &= f(S, V - S) = \\
 &= f(S, V) - \underbrace{f(S, S)}_{=0} = \\
 &= f(S, V) = \\
 &= f(s, V) + \underbrace{f(S - \{s\}, V)}_{=0} = \\
 &= f(s, V) = |f|
 \end{aligned}$$

Ještě poznamenejme, že $f(S - \{s\}, V) = 0$ platí proto, že $f(u, V) = 0$ pro $\forall u \in V - \{s, t\}$. □

Důsledek 7.17

Velikost libovolného toku v síti je shora omezena kapacitou libovolného řezu.

Důkaz: Nechť f je tok a (S, T) je řez. Pak:

$$|f| \stackrel{\text{Lemma 7.16}}{=} f(S, T) \leq c(S, T)$$

□

Věta 7.18

Nechť f je tok v síti (G, s, t, c) . Je ekvivalentní:

1. f je maximální tok.
2. neexistuje zvětšující cesta v (G, s, t, c) vzhledem k f .
3. $|f| = c(S, T)$ pro nějaký řez (S, T) .

Důkaz:

- 1 \Rightarrow 2 Pripusťme, že f je maximální tok a G_f má zvětšující cestu vzhledem k f . Pak pomocí lemmatu 7.14 je f_p tok v G_f a $|f_p| > 0$ a pomocí lemmatu 7.12 je $f + f_p$ tok v G a $|f + f_p| > |f|$, což je spor.
- 2 \Rightarrow 3 Nechť neexistuje zvětšující cesta. Definujme řez:

$$\begin{aligned}
 S &:= \{v \in V \mid \text{v } G_f \text{ existuje } s - v \text{ cesta}\} \\
 T &:= V - S
 \end{aligned}$$

(S, T) je řez, protože neexistuje $s - t$ cesta v síti G_f . Pro $\forall u \in S, v \in T$ je $f(u, v) = c(u, v)$ (jinak by $(u, v) \in E_f, v \in S$), proto tedy:

$$|f| \stackrel{\text{Lemma 7.16}}{=} f(S, T) = c(S, T)$$

- 3 \Rightarrow 1 Z důsledku 7.17 plyne $|f| \leq c(S, T)$ pro \forall řez (S, T) , proto rovnost pro nějaký řez dá okamžitě maximalitu f . □

Algoritmus 7.2

FORD_FULKERSON(G, s, t)

1. for each edge $(u, v) \in E[G]$ do
2. $f[u, v] \leftarrow 0$
3. $f[v, u] \leftarrow 0$
4. while there exists a path p from s to t in G_f do

5. $c_f(p) \leftarrow \min \{c_f(u, v) \mid (u, v) \text{ in } p\}$
6. for each edge (u, v) in p do
7. $f[u, v] \leftarrow f[u, v] + c_f(p)$
8. $f[v, u] \leftarrow f[v, u] - c_f(p)$

V případě, že jsou-li kapacity celé, je algoritmus 7.2 konečný. Jsou-li kapacity racionální, pak je stačí vynásobit jejich společným jmenovatelem a máme kapacity celé. V těchto případech je počet iterací na řádcích 4. – 8. konečný.

Složitost: Nechť $c : E \rightarrow \mathbf{Z}$. Pak složitost vypadá takto:

- Řádky 1. – 3. je $O(m)$.
- Pokud se v řádku 4. vybírá libovolná cesta, pak je to zvládnutelné v čase $O(m)$.
- Počet iterací řádků 4. – 8. je maximálně $|f^*|$, kde f^* je maximální tok.
- Jedna iterace řádků 4. – 8.: Nechť $G' = (V, E')$ a $E' := \{(u, v) \mid (u, v) \in E \vee (v, u) \in E\}$. Pak je $E_f \subseteq E'$. Cesty v E_f lze najít např. pomocí BFS (algoritmus 3.1) nebo pomocí DFS (algoritmus 3.3) v čase $O(m + n) = O(m)$, neboť G je souvislý. Tedy jednu iteraci řádků 4. – 8. lze provést v čase $O(m)$.
- Celkem tedy $O(m \cdot |f^*|)$.

Algoritmus 7.3

Algoritmus EDMONDS_KARP je uveden v literatuře ([3]) a má čtyři řádky. Spočívá v tom, že ohodnotíme hrany G_f jedničkami. Pak $\delta_f(u, v)$ je délka (počet hran na ní) nejkratší $u - v$ cesty. Algoritmus tedy vyhledá nejkratší $s - t$ cestu.

Složitost: V této publikaci si uvedeme důkaz složitosti, kdy výsledná složitost algoritmu 7.3 je $O(m^2n)$.

V literatuře [3] můžeme najít důkazy lepších složitostí:

- $O(mn^2)$ v odstavci 27.4.
- $O(n^3)$ v odstavci 27.5.

Lemma 7.19

Při použití algoritmu 7.3 na síť (G, s, t, c) funkce $\delta_f(s, v)$ pro $\forall v \in V - \{s, t\}$ neklesá.

Důkaz: Připustíme, že pro $v \in V - \{s, t\}$ existuje zvětšující cesta vzhledem k f , zvětšující tento tok na f' a přitom

$$\delta_{f'}(s, v) < \delta_f(s, v) \quad (*)$$

Volbou v lze dosáhnout, že $\forall u \in V - \{s, t\}$ platí

$$(\delta_{f'}(s, u) < \delta_f(s, u) \implies \delta_{f'}(s, v) \leq \delta_{f'}(s, u))$$

Nechť $p' \equiv s \rightarrow \dots \rightarrow u \rightarrow v$ je nejkratší $s - v$ cesta v $G_{f'}$. Platí:

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) - 1 \quad (1)$$

$$\delta_f(s, u) \leq \delta_{f'}(s, u) \quad (2)$$

přičemž rovnice 1 plyne z lemmatu 6.7 a bod 2 (plyne z bodu 1) platí i pro $u = s$.

Je-li $f[u, v] < c(u, v)$, je $(u, v) \in G_f$ a

$$\begin{aligned} \delta_f(s, v) &\stackrel{\text{Lemma 6.5}}{\leq} \delta_f(s, u) + 1 \leq \\ &\stackrel{\text{bod 2}}{\leq} \delta_{f'}(s, u) + 1 = \\ &\stackrel{\text{bod 1}}{=} \delta_{f'}(s, v) \end{aligned}$$

což je spor s tím, že $f[u, v] < c(u, v)$.

Tedy $f[u, v] = c(u, v)$ a $(u, v) \notin G_f$. Necht p je $s - t$ (zvětšující) cesta v G_f dávající $G_{f'}$. (v, u) leží na p (neboť $(u, v) \notin E_f$ a $(u, v) \in E_{f'}$).

Tedy $\delta_f(s, u) = \delta_f(s, v) + 1$.

Nyní

$$\begin{aligned} \delta_f(s, v) &= \delta_f(s, u) - 1 \leq \\ &\underbrace{\leq}_{\text{bod 2}} \delta_{f'}(s, u) - 1 = \\ &\underbrace{=}_{\text{bod 1}} \delta_{f'}(s, v) - 2 < \\ &< \delta_{f'}(s, v) \end{aligned}$$

což je samozřejmě spor s (*). □

Věta 7.20

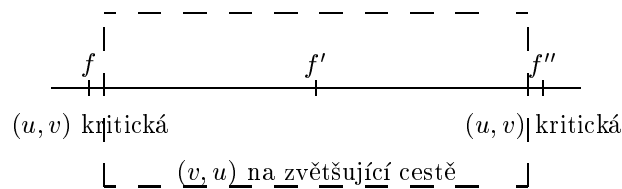
Počet iterací v algoritmu 7.3 je $O(mn)$.

Důkaz: Označení: (u, v) je *kritická* hrana v G_f vzhledem ke zvětšující cestě p , je-li $c_f(p) = c_f(u, v)$.

Kolikrát může být (u, v) kritická? Po zvětšení toku na f' pomocí f již není v $G_{f'}$. Jestliže je (u, v) kritická vzhledem k f , dostaneme

$$\delta_f(s, v) = \delta_f(s, u) + 1 \tag{3}$$

Znovu se může objevit na zvětšující cestě jen po té, co (v, u) bylo na nějaké zvětšující cestě (viz obr. 13).



Obrázek 13: Příklad kritické hrany

Tedy

$$\begin{aligned} \delta_{f'}(s, u) &= \delta_{f'}(s, v) + 1 \geq \\ &\underbrace{\geq}_{\text{Lemma 7.19}} \delta_f(s, v) + 1 = \\ &\underbrace{\geq}_{\text{vztah 3}} \delta_f(s, u) + 2 \end{aligned}$$

Tedy mezi dvěma okamžiky, kdy je hrana (u, v) kritická, vzroste $\delta_f(s, v)$ alespoň o 2. Přicházejí do úvahy hodnoty $0, 1, \dots, n - 2$. Tedy (u, v) je $O(n)$ -krát kritická.

V průběhu celého algoritmu 7.3 je $O(mn)$ kritických hran.

Každá zvětšující cesta má alespoň jednu kritickou hranu a po modifikaci toku f tato hrana přestává být kritická, tedy tento algoritmus proběhne v celkem $O(mn)$ iteracích. □

Složitost algoritmu 7.3: Z věty 7.20 plyne, že složitost algoritmu 7.3 je $O(m^2n)$, protože jedna iterace tohoto algoritmu vezme $O(m)$.

Literatura

- [1] L. Kučera: *Kombinatorické algoritmy*
- [2] Plesnik: *Grafové algoritmy*, 85 – 88
- [3] M. Syslo: *Discrete Optimization Algorithms with Pascal Programs*, Prentice Hall 83