10. Nejkratší cesty *(zapsali L. Banáková, O. Hoferek, J. Břečka)*

Definice: *Vzdálenost vrcholů* $d(u,v)$ je délka nejkratší cesty (u,v), pokud existuje, nebo $+\infty$, pokud taková cesta neexistuje.

Problém: Je dán graf G a funkce $l : E(G) \to \mathbb{R}$ přiřazující hranám jejich délky. Dále je dán startovací a cílový vrchol $s, c \in V(G)$. Chceme najít cestu $s = v_1, v_2, \ldots, v_k = c$ takovou, aby délka cesty

$$l(v_1, v_2) + l(v_2, v_3) + \ldots + l(v_{k-1}, v_k)$$

byla minimální. Takovéto cestě budeme říkat *nejkratší cesta* z s do c.

Bez újmy na obecnosti budeme předpokládat, že graf G je orientovaný, bez smyček a násobných hran.

Nejprve uvážíme případy, kdy jsou všechny hrany stejně dlouhé. Tehdy můžeme použít algoritmus prohledávání do šířky (BFS). K jeho implementaci je zapotřebí fronta Q s metodami $Put(Q,v)$ (přidání prvku v na konec fronty) a $Get(Q)$ (odebrání prvku ze začátku fronty). Algoritmus vrátí pole „značek“ z: $z(v) = 1$ právě tehdy pokus je vrchol v dostupný ze startovacího vrcholu s, jinak $z(v) = 0$.

Algoritmus: (Prohledávání do šířky (BFS))

1. $z(*) \leftarrow 0, z(s) \leftarrow 1$
2. $Q \leftarrow \{s\}$
3. while $Q \neq \emptyset$:
4. $v \leftarrow Get(Q)$
5.   for $\forall w : (v,w) \in E$
6.   if $z(w) = 0$ then
7. $z(w) \leftarrow 1$
8. $Put(Q,w)$

Lemma Č (časové): BFS doběhne v čase $\mathcal{O}(m+n)$.

Důkaz: Každý vrchol se do fronty dostane maximálně jednou a každou hranu zpracujeme maximálně jednou. ♡

Lemma D (dosažitelnost): BFS označí právě vrcholy dostupné ze s.

Důkaz:

„$\Rightarrow$“: Indukcí podle běhu algoritmu.

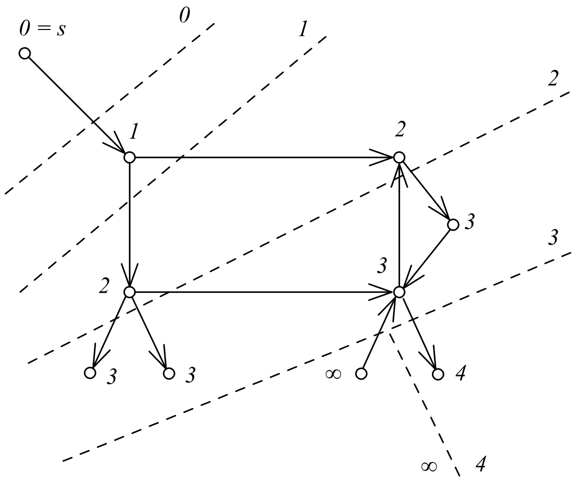
„$\Leftarrow$“: Dokážeme sporem. Uvažme, že existuje vrchol dostupný, ale algoritmem neoznačený. Vezměme takovýto „špatný“ vrchol v, který je s nejblíže. Uvažme nejkratší cestu (s,v): $s, \ldots, u, v$. Předchozí vrchol na této cestě u musí být „dobrý“. Vrchol u se dostane do fronty, pak je z ní vybrán a tím se zpracuje i vrchol $v \Rightarrow$ spor.

Předchozím algoritmem jsme pouze zjistili, které vrcholy jsou z s dosažitelné. Nyní si tento algoritmus modifikujeme tak, abychom našli $d(s,v)$ pro všechna $v \in V(G)$.

Rozšířený algoritmus: (Prohledávání do šířky (BFS))

$z(*) \leftarrow 0, z(s) \leftarrow 1$
$D(*) \leftarrow +\infty, D(s) \leftarrow 0$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$:
    $v \leftarrow Get(Q)$
    for $\forall w : (v, w) \in E$
        if $z(w) = 0$ then
            $z(w) \leftarrow 1$
            $D(w) \leftarrow D(v) + 1$
            $Put(Q, w)$

Průběh tohoto algoritmu si můžeme ukázat na následujícím obrázku („prasečí graf“):



Prasečí graf

Definice: *Vrstva* $L_i \subseteq V : L_0 = \{s\}$, $L_{i+1} = \{w : w$ označen při procházení vrcholu $v \in L_i\}$

Lemma V (vzdálenosti): Na konci BFS je $\forall v : D(v) = d(s, v)$.

Důkaz:

a) pro nedosažitelné vrcholy: OK (algoritmus je neoznačí $\Rightarrow D(v) = \infty$)

b) pro dosažitelné vrcholy: Víme, že pro vrchol $v \in L_i$ platí $D(v) = i$, stačí tedy zjistit, zda $\forall i : \forall v \in L_i$ je $d(s, v) = i$. To ověříme matematickou indukcí podle i:

pro $i = 0$: evidentní,

pro $i > 0$: Při procházení vrstvy L_{i-1} musí být vrchol v takový, že $d(s, v) = i$, označen a zároveň nemohl být označen dříve a tudíž patří do L_i. ♡

Všimněme si, že v grafu existují pouze tři typy hran vzhledem k našemu algoritmu:

- dopředu o jednu vrstvu

- dozadu o jednu nebo víc vrstev
- ve vrstvě

Abychom našli i samotné cesty, nejen jejich délky, modifikujeme svůj algoritmus tak, že si pro každý vrchol v, který označíme, budeme pamatovat jeho předchůdce $p(v)$:

$z(*) \leftarrow 0, z(s) \leftarrow 1$
$D(*) \leftarrow +\infty, D(s) \leftarrow 0$
$p(*) \leftarrow ?$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$:
    $v \leftarrow Get(Q)$
    for $\forall w : (v, w) \in E$
        if $z(w) = 0$ then
            $z(w) \leftarrow 1$
            $D(w) \leftarrow D(v) + 1$
            $p(w) \leftarrow v$
            $Put(Q, w)$

Pozorování: Pokud $p(v) \neq ? \Rightarrow D(v) = D(p(v)) + 1$.

Lemma S (stromové): Graf (W, F), kde $W =$ dosažitelná část V, $F = \{(v, p(v))\}$ je strom orientovaný do kořene s. Navíc ukazuje nejkratší cestu pro každý dosažitelný vrchol (pro každý vrchol v tomto grafu, existuje jediná orientovaná cesta do kořene s, která je zároveň nejkratší cesta z s do tohoto vrcholu).

Nyní uvažme, že délky hran l nejsou všechny stejně dlouhé:

Ukážeme si trik pro ohodnocené hrany: hranu délky l podrozdělíme na l hran délky 1 a pak spustíme BFS. Algoritmus poběží s časovou složitostí $O((m+n)l_{max})$. To není zrovna moc výhodné, a proto si ukážeme tzv. *Budíkovou taktiku*:

Místo toho, abychom každou hranu podrozdělovali, si budeme pro každý vrchol pamatovat „čas budíku“ b. Časem $b(v)$ budíku vrcholu v rozumíme čas, kdy se do tohoto vrcholu vlna dostane, když se bude šířit po dosud známých hranách. Na začátku nastavíme $\forall v \in V : b(v) = l(s, v)$ a pokud při běhu algoritmu objevíme hranu, která nám umožní dostat se do v rychleji, tak hodnotu $b(v)$ změníme. Algoritmus pak bude pracovat tak, že bude spát, dokud nezazvoní nějaký budík (což znamená, že jsme dorazili do nějakého vrcholu).

Tato taktika nás přivede k Dijkstrovu algoritmu, kde se ze dvoustavového $z(v)$ stane třístavové $z(v)) \in \{\mathbf{N}, \mathbf{V}, \mathbf{H}\}$, kde $\mathbf{N}$ znamená neviděn, $\mathbf{V}$ viděn a $\mathbf{H}$ hotov. Podle hodnoty $z(v)$ se pak $D(v)$ rovná $+\infty$ pro $z(v) = \mathbf{N}$, času budíku pro $z(v) = \mathbf{V}$ nebo vzdálenosti $d(s, v)$ pro $z(v) = \mathbf{H}$.

Algoritmus: (Dijkstrův algoritmus)

$z(*) \leftarrow \mathbf{N}, z(s) \leftarrow \mathbf{V}$
$D(*) \leftarrow +\infty, D(s) \leftarrow 0$

```
p(*) ← ?
while ∃v : z(v) = V :
    v ← vrchol, pro nějž z(v) = V a zároveň D(v) je minimální
    z(v) ← H
    for ∀w : (v, w) ∈ E:
        if D(w) > D(v) + l(v, w) then
            D(w) ← D(v) + l(v, w)
            z(w) ← V
            p(w) ← v
```

Věta: Pokud $\forall v, w : l(v, w) \in \mathbb{Z}^+ \cup \{\infty\}$, pak Dijkstrův algoritmus najde nejkratší cesty v čase $\mathcal{O}(n^2)$.

Důkaz: V průběhu algoritmu se maximálně n-krát hledá vrchol v s minimálním $D(v)$, maximálně m-krát se přenastavuje $D(v)$. Pokud bude struktura D uložena jako pole (hledání minima v $\mathcal{O}(n)$, změna hodnoty $\mathcal{O}(1)$), bude celková časová složitost Dijkstrova algoritmu $\mathcal{O}(n^2+m) = \mathcal{O}(n^2)$. Fakt, že se algoritmus zastaví je evidentní, protože se stav žádného vrcholu nevrací z **H** do **V**.

Úplný důkaz správnosti a časové složitosti i pro hrany ohodnocené reálnými čísly bude na příští přednášce. ♡

Nahlédněme nyní, zda by vedlo k vylepšení časové složitosti, kdybychom při implementaci Dijkstrova algoritmu použili jako datovou strukturu pro ukládání viděných vrcholů různé typy hald. Prováděli bychom s nimi operace *DeleteMin* (maximálně n-krát) a *Decrease* (maximálně m-krát). Zmíněné operace mají pro různé typy hald následující časové složitosti:

	binární halda	*k-regulární halda*	*Fibonacciho halda*
DeleteMin	$\mathcal{O}(\log n)$	$\mathcal{O}(k \log_k n)$	$\mathcal{O}(\log n)$
Decrease	$\mathcal{O}(\log n)$	$\mathcal{O}(\log_k n)$	$\mathcal{O}(1)$
celková složitost	$\mathcal{O}((m+n)\log n)$	$\mathcal{O}(m \log_{m/n} n)$	$\mathcal{O}(n \log n + m)$

Použijeme-li klasickou haldu, dostaneme celkovou časovou složitost $\mathcal{O}((m+n)\log n)$, což znamená, že jsme si pomohli v případech, kdy je graf „řídký“. U k-regulární haldy se celková časová složitost rovná $\mathcal{O}(nk \log_k n + m \log_k n)$, ideálně pro $k = m/n$: $\mathcal{O}(m \log_{m/n} n)$. Prozkoumejme, jak se nám v tomto případě bude měnit časová složitost pro různě „husté“ grafy:

- $m \approx n \to \mathcal{O}((m+n)\log n)$
- $m \approx n^2 \to \mathcal{O}(m)$
- $m \approx n^{1+\epsilon} \to \mathcal{O}(m) \ldots (k = m/n = n^\epsilon)$

Při použití Fibonacciho haldy je pak celková časová složitost rovna $\mathcal{O}(n \log n + m)$.

Nyní si předvedeme trochu jiný algoritmus na prohledávání grafu. Mějme matici sousednosti A ke grafu $G = (V, E)$: $A_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$, jinak $A_{ij} = 0$. Potom

$A^2_{ij} = \sum_{k=1}^{n} A_{ik}A_{kj}$. Tento součet je nenulový právě tehdy, když vrcholy i,k,j tvoří sled délky 2. Jinými slovy A^2_{ij} se rovná počtu sledů délky 2 v G z i do j. Obdobně A^k_{ij} = počet sledů délky k. Pokud k matici A přičteme jednotkovou matici $\mathbb{E}_n$ (čímž si přimyslíme smyčky), dostaneme po umocnění:

$(A + \mathbb{E}_n)^k_{ij} \neq 0 \Leftrightarrow \exists\ i, j$-sled délky maximálně k.

Výstupem algoritmu tedy bude matice $(A + \mathbb{E}_n)^n$. Můžeme si také všimnout, že pokud matici $A + \mathbb{E}_n$ umocníme na číslo větší než n, tak výsledek bude pořád správný, jen bude výpočet trvat déle. Takže algoritmus může pracovat v iteracích umocňováním matice na druhou: $M_0 = A + \mathbb{E}_n$, $M_1 = M_0^2$, $M_{i+1} = M_i^2$. Výstupní matice bude tudíž:

$M_{\lceil \log n \rceil (ij)} \neq 0 \Leftrightarrow \exists$ cesta z i do j

Na výpočet dosažitelnosti vrcholů v grafu tedy stačí $\mathcal{O}(\log n)$ násobení matic, a pokud na toto použijeme Strassenův algoritmus, dostaneme ve výsledku algoritmus se složitostí $\mathcal{O}(n^{\log_2 7} \log n)$.