Kapitola 13

Kombinatorická optimalizace

Tři příklady

Pohyb robotu ve skladu

Popis skladu, popis robotu (poloha a směr), příkazy (vykonání každého příkazu trvá 1 vteřinu),

`START:` pokud robot stojí, začne se pohybovat v přímém směru a urazí vzdálenost 0,5m, po vykonání příkazu je v pohybu,

`MOVE(`x`):` pokud je robot v pohybu, pokračuje v pohybu v přímém směru do vzdálenosti x metrů, po vykonání příkazu je v pohybu, x může nabývat hodnot $x = 1, 2, 3$,

`STOP:` pokud je robot v pohybu, začne ve stejném směru zpomalovat a během 0,5m se zastaví,

`TURN(`*dir*`):` pokud robot stojí, otočí se o 90^o ve směru *dir*, který může nabývat hodnot $+$, což znamená otočení proti směru hodinových ručiček, a $-$, která znamená otočení po směru hodinových ručiček, po vykonání příkazu robot opět stojí.

Pomocí těchto příkazů můžeme naprogramovat pohyb robotu po skladu. Příklad dvou různých programů pro pohyb mezi dvěma stejnými místy. Hledáme pohyb, který trvá nejkratší možnou dobu. *Horní odhad (mez)* a *dolní odhad (mez)* pro celkovou dobu trvání přesunu.

Horní odhad získáme tak, že ukážeme příklad pohybu. Najít nějaký dolní odhad je složitější.

Tvrzení 13.1 *Má-li počáteční poloha robotu souřadnice* (i, j)*, v ní je robot otočený jižním směrem, a konečná poloha robotu má souřadnice* (k, l)*, kde*

$k > i$ a $l > j$, pak nejkratší cesta trvá aspoň

$$6 + \left\lceil \frac{k-i-1}{3} \right\rceil + \left\lceil \frac{l-j-1}{3} \right\rceil$$

vteřin. Symbol $\lceil x \rceil$, kde x je reálné číslo, označuje nejmenší celé číslo větší nebo rovné x.

Důkaz. Na přednášce. □

Matenatický model úlohy

Orientovaný graf, vrcholy možné polohy robotu, hrany odpovídají změnám polohy robotu po jednom příkazu. Vrcholy grafu jsou trojice (i, j, dir), kde $dir \in \{N, S, E, W\}$ a i, j jsou čísla tvaru $t/2$, kde t je přirozené číslo. Velikost čísel i, j je dána velikostí skladu. Můžeme například předpokládat, že $1 \leq i \leq 49$ a $1 \leq j \leq 79$, pokud má sklad velikost 50×80 metrů.

Hrany určené jednotlivými příkazy popíšeme následovně. Napřed příkaz `START:`

$(i, j, N) \longrightarrow (i, j + 1/2, N)$, $i \in \{1, 2, \ldots, 49\}, j \in \{1, 2, \ldots, 78\}$,
$(i, j, S) \longrightarrow (i, j - 1/2, S)$, $i \in \{1, 2, \ldots, 49\}, j \in \{2, 3, \ldots, 79\}$,
$(i, j, E) \longrightarrow (i + 1/2, j, E)$, $i \in \{1, 2, \ldots, 48\}, j \in \{1, 2, \ldots, 79\}$,
$(i, j, W) \longrightarrow (i - 1/2, j, W)$, $i \in \{2, 3, \ldots, 49\}, j \in \{1, 2, \ldots, 79\}$.

Některé z těchto hran nemusí existovat, pokud jsou v okolí bodu o souřadnicích (i, j) nějaké překážky.

Příkaz `TURN`$(+)$ vede k následujícím hranám.

$(i, j, N) \longrightarrow (i, j, W)$, $i \in \{1, 2, \ldots, 49\}, j \in \{1, 2, \ldots, 79\}$,
$(i, j, W) \longrightarrow (i, j, S)$, $i \in \{1, 2, \ldots, 49\}, j \in \{1, 2, \ldots, 79\}$,
$(i, j, S) \longrightarrow (i, j, E)$, $i \in \{1, 2, \ldots, 49\}, j \in \{1, 2, \ldots, 79\}$,
$(i, j, E) \longrightarrow (i, j, N)$, $i \in \{1, 2, \ldots, 49\}, j \in \{1, 2, \ldots, 79\}$.

Podobně můžeme vypsat hrany pro všechny ostatní příkazy. Některé hrany nebudou do grafu zařazené kvůli překážkám ve skladu. Řešení úlohy spočívá v nalezení nejkratší orientované cesty v grafu např. mezi počátečním vrcholem (i, j, S) a koncovým vrcholem (k, l, W).

Erdösovo číslo

Matematická formulace opět spočívá v tom, že máme nějaký neorientovaný graf. Jeho vrcholy jsou jednotliví vědci (matematici), dva vrcholy jsou spojené hranou, pokud tito dva vědci napsali společný vědecký článek. Najít Erdösovo číslo $e(X)$ nějakého matematika X znamená najít nejkratší cestu v tomto grafu z vrcholu E odpovídajícího P. Erdösovi do vrcholu odpovídajícího badatelovi X. Horní odhad Erdösova čísla $e(X)$ najdeme tak,

že ukážeme nějakou cestu z vrcholu E do vrcholu X. Dokázat dolní odhad čísla $e(X)$ je mnohem obtížnější.

Ideální štafeta

Štafetový závod v běhu sestává ze 14 úseků různé délky a obtížnosti. Máme k dispozici 14 běžců, ze kterých máme sestavit ideální štafetu, tj. přiřadit jim jednotlivé úseky podle jejich výkonnosti tak, aby celkový čas byl co nejmenší. Můžeme to udělat tak, že v rámci tréninku necháme vždy všechny členy týmu uběhnout každý úsek štafety a změříme jejich čas. Výsledky tréninků můžeme zapsat do čtvercové matice $\mathbf{A} = (a_{ij})$ řádu 14, kde číslo a_{ij} je čas, kterého dosáhl běžec i na úseku j. Naším úkolem je vybrat z matice $\mathbf{A}$ 14 prvků tak, abychom z každého řádku vybrali jedno číslo (každý člen týmu běží nějaký úsek) a z každého sloupce jedno číslo (každý úsek štafety běží pouze jeden člen týmu) tak, aby jejich součet byl co nejmenší.

Také tuto úlohu můžeme formulovat pomocí grafů. Vrcholy grafu G budou množina I členů týmu a množina J úseků běhu. Mezi každým vrcholem $i \in I$ a vrcholem $j \in J$ bude neorientovaná hrana. Tato hrana bude ohodnocena číslem a_{ij}, které bude znamenat dobu, jakou trvalo při tréninku běžci i uběhnout úsel j. Hledáme *perfektní párování* v grafu G, tj. množinu hran F, tak aby z každého vrcholu grafu G vycházela právě jedna hrana množiny F, pro kterou je souhodnot hran z F co nejmenší.

Také v tomto případě je jednoduché najít některý horní odhad času, který potřebuje ideální štafeta k proběhnutí celé trasy. Najdeme jakékoliv perfektní párování a sečteme hodnoty hran, které do tohoto párování patří. Dolní odhad můžeme najít například tak, že sečteme nejkratší časy, kterých členové týmu dosáhli na jednotlivých úsecích.

Hledání nejkratší cesty v grafu

Prohledávání grafu do šířky

Máme graf $G = (V, E)$ (orientovaný nebo neorientovaný), V je množina vrcholů a E je množina hran. První algoritmus pro nalezení nekratší cesty z daného vrcholu $s \in V$ do libovolného jiného vrcholu $v \in V$ se nazývá *prohledávání grafu do šířky*, anglicky *breadth-first search*, zkráceně *BFS*. Budeme potřebovat jednu další datovou strukturu *queue* Q, česky *fronta*, kterou je v každém okamžiku nějaká posloupnost

$$Q = v_1, v_2, \ldots, v_r$$

vrcholů grafu (V, E). Fronta Q bude inicializována vložením jednoho vrcholu $Q := s$.

Kromě toho lze s datovou strukturou Q provádět tři operace.

head(Q) vrátí první vrchol v_1 fronty $Q = v_1, v_2, \ldots, v_r$,

enqueue(Q) přidá nový vrchol v jako poslední prvek ke Q,

dequeue(Q) odebere první prvek v_1 z $Q = v_1, v_2, \ldots, v_r$.

Algoritmus BFS systematicky prohledává graf G gtak, že pro každé přirozené číslo k napřed probere všechny vrcholy, které jsou ve vzdálenosti k od vrcholu s, než najde jakýkoliv vrchol ve vzdálenosti $k + 1$ od s.

Po ukončení algoritmu údaj $d[v]$ pro $v \in V$ obsahuje délku nějaké nejkratší cesty z vrcholu s do vrcholu v, a pro každé $v \neq s$ hodnota $\pi[v]$ obsahuje předchůdce v v nějaké nejkratší cestě z s do v, případně se rovná `NIL`, pokud žádná taková cesta neexistuje. Algoritmus BFS rovněž označí každý vrchol v v okamžiku, kdy se s ním poprvé setká. To zapíšeme jako *mark* v.

Algoritmus BFS(G=(V,E),s)

```
FOR u ∈ V \ {s} DO
    d[u] := ∞
    π[u] := NIL
END
mark s
d[s] := 0
Q := s
WHILE q ≠ ∅ DO
    u := head(Q)
    FOR all v such that (u, v) ∈ E DO
        IF v not marked THEN
            mark v
            d[v] := d[u] + 1
            π[v] := u
            enqueu(Q, v)
        END
    END
    dequeue(Q)
END
```

Napřed odhadneme dobu trvání algoritmu pomocí několika drobných pozorování. Každý vrchol je připojen do fronty Q nejvýše jednou ihned poté, co je označen, proto cyklus `WHILE` proběhne nejvýše $|V|$-krát. Operace s frontou Q, tj. head(Q), enqueue(Q, v) a dequeue(Q), vyžadují každá konstantní dobu. Proto celková doba provádění operací mimo cyklus `FOR` uvnitř `WHILE`

cyklu se rovná $O(|V|)$. Uvnitř tohoto `FOR` cyklu zpracováváme každou hranu $(u, v) \in E$ nejvýše jednou. Doba, kterou vyžaduje zpracování jedné hrany je konstantní, čili průběh `FOR` cyklu vyžaduje čas $O(|E|)$. Platí proto

Lemma 13.2 *Algoritmus BFS vyžaduje čas* $O(|V| + |E|)$.

Vzdálenost mezi vrcholy $s, v \in V$ v grafu G, tj. počet hran v nejkratší cestě mezi vrcholy s a v označíme $\delta(s, v)$.

Lemma 13.3 *Platí* $\delta(s, v) \leq d[v]$ *pro každý vrchol* $v \in V$.

Důkaz. Algoritmus BFS najde nějakou cestu mezi vrcholy s a $v \neq s$ délky $d[v]$, vrcholy této cesty v opačném pořadí jsou

$$v, \pi[v], \pi[\pi[v]], \ldots, s.$$

Proto $\delta(s, v) \leq d[v]$ pro libovolný vrchol $v \in V$, $v \neq s$. Dokazovaná neorvnost platí rovněž v případě $v = s$, neboť $d[s] = 0$. □

Lemma 13.4 *Předpokládejme, že v nějakém okamžiku průběhu algoritmu BFS fronta* $Q = v_1, v_2, \ldots, v_r$. *Potom platí*

- *existuje přirozené číslo* l *takové, že* $d[l] \in \{l, l + 1\}$ *pro každé* $i = 1, 2, \ldots, r$,
- $d[v_i] \leq d[v_{i+1}]$ *pro libovolné* $i = 1, 2, \ldots, r - 1$.

Důkaz. Budeme postupovat indukcí podle počtu operací *enqueue* a *dequeue*. Na počátku je $Q = s$ a tvrzení očividně platí.

Nyní předpokládejme, že v průběhu algoritmu je $Q = v_1, v_2, \ldots, v_r$ a fronta Q splňuje obě podmínky. Operace *dequeue* je zachová. Pokud uděláme operaci enqueue(Q, v) v případě, že $d[v_1] < d_[v_r]$, pak $d[v] = d[v_1]+1 = d[v_r]$ a obě vlastnosti z tvrzení lemmatu platí. Pokud $d_[v_1] = d_[v_r] = l$, pak $d[v] = d[v_1] + 1 = l + 1$ a obě tvrzení lemmatu jsou opět splněná. □

Věta 13.5 *Po skončení algoritmu BFS platí pro každý vrchol* $v \in V$, *že* $\delta(s, v) = d[v]$.

Důkaz. Budeme postupovat indukcí podle $\delta(s, v)$. Je-li $\delta(s, v) = 0$, platí $v = s$ a $d[s] = 0$. Protože se v průběhu algoritmu hodnota $d[v]$ nemění pro žádný vrchol $v \in V$, platí rovněž na konci algoritmu $d[s] = 0 = \delta(s, s)$.

Předpokládejme nyní, že $\delta(s,v) = k > 0$ a že tvrzení věty platí pro všechny vrcholy $u \in V$, pro které je $\delta(s,u) < k$. Musí existovat vrchol $u \in V$ takový, že $\delta(s,u) = k-1$ a $(u,v) \in E$. Je to vrchol předcházející v v libovolné cestě z vrcholu s do vrcholu v, která má nejkratší možnou délku. Označme u_0 první z těchto vrcholů u, který se objeví ve frontě Q. Každý z těchto vrcholů u se musí objevit ve frontě Q, protože podle indukčního předpokladu platí $d[u] = \delta(s,u) = k-1$, proto hodnota $d[u] = k-1$ byla v průběhu algoritmu stanovena a ihned poté byl vrchol u přidán do Q.

Pokud vrchol v nebyl označen v okamžiku, kdy se vrchol u_0 stal prvním členem fronty Q, dostaneme $d[v] = d[u_0]+1 = k$ a důkaz je dokončen. Může se stát, že v okamžiku, kdy se vrchol u_0 stal prvním členem fronty Q, byl už vrchol v označen? V takovém případě by byl vrchol v označen v okamžiku, kdy byl nějaký jiný vrchol u_1 na prvním místě fronty Q a existovala by hrana (u_1,v). V takovém případě by muselo platit $\delta(s,u_1) < \delta(s,u_0) = k-1$, neboť vrchol u_0 byl první mezi vrcholy u, ze kterých vede hrana do vrcholu v a pro které platí $\delta(s,u) = k-1$. Potom by ale platilo $\delta(s,v) = \delta(s,u_1)+1 < k$, což je spor s předpokladem $\delta(s,v) = k$. Tento spor dokazuje, že vrchol v je ve skutečnosti neoznačen v okamžiku, kdy se u_0 stal prvním členem posloupnosti Q. □

Dijkstrův algoritmus

V tomto případě budeme předpokládat, že hrany grafu $G = (V,E)$ jsou ohodnocené nezápornými reálnými čísly, kterým říkáme *váha* hrany (u,v). Budeme ji označovat $w(u,v)$ nebo také $w(e)$, je-li $e = (u,v)$. Je-li

$$p : u = u_0, u_1, \ldots, u_r = v$$

nějaká cesta z vrcholu u do vrcholu v v grafu G, pak číslo

$$w(p) = \sum_{i=1}^{r-1} w(u_i, u_{i+1})$$

nazýváme váhou cesty p. *Vzdáleností* $\delta(u,v)$ nazýváme nejmenší váhu všech cest z vrcholu u do vrcholu v v grafu G. V případě, že $w(e) = 1$ pro každou hranu $e \in E$, je vzdálenost $\delta(u,v)$ prostě počtem hran v nejkratší cestě z vrcholu u do vrcholu v v grafu $G = (V,E)$.

Dijkstrův algoritmus počítá vzdálenost $\delta(u,v)$ mezi libovolnými dvěma vrcholy grafu $G = (V,E)$, ve kterém má každá hrana $e \in E$ přiřazenou nezápornou váhu $w(e)$. Stejně jako v případě BFS algoritmu po skončení Dijkstrova algoritmu známe pro každý vrchol $v \in V$ hodnotu $d[v]$, váhu nejkratší cesty z nějakého pevně zvoleného vrcholu $s \in V$ do vrcholu v.

Dále je výstupem Dijkstrova lagoritmu vrchol $\pi[v]$, který předchází vrchol v v nějaké cestě s nejmenší váhou z vrcholu s do vrcholu v. Narozdíl od algoritmu BFS se hodnota $d[v]$ může v průběhu Dikstrova lagoritmu měnit.

Také Dijkstrův algoritmus bude využívat datovou strukturu–frontu Q, která bude posloupností nějakých vrcholů grafu G uspořádaných podle současné velikosti funkce d od nejmenší po největší. Můžeme zjišťovat, je-li fronta Q prázdná. S frontou Q můžeme dělat dvě operace:

- extract_min(Q) odstraní z Q vrchol s nejmenší hodnotou $d[v]$ mezi všemi vrcholy fronty Q,
- decrease_key(Q, v, ρ), která stanoví novou hodnotu $d[v] = \rho$, pokud pro původní hodnotu $d[v]$ platilo $d[v] < \rho$.

Dijkstrův algoritmus postupně vytváří množinu vrcholů S, pro které už byla nejkratší cesta z vrcholu s nalezena. Tato množina je na počátku prázdná a postupně se zvětšuje.

Dijkstra(G, s)

```
FOR u ∈ V DO
    d[u] := ∞
    π[u] := NIL
END
d[s] := 0
S := ∅
Q := V
WHILE Q ≠ ∅ DO
    u := extract_min(Q)
    S := S ∪ {u}
    FOR všechny v ∈ V takové, že (u, v) ∈ E DO
        IF d[v] > d[u] + w(u, v) THEN
            d[v] := d[u] + w(u, v)
            π[v] := u
            decrease_key(Q, v, d[v])
        END
    END
END
```

Lemma 13.6 *Dijkstrův algoritmus vyžaduje čas* $O(|V|^2 + |E|)$.

Důkaz. V cyklu `FOR` uvnitř cyklu `WHILE` zpracováváme každou hranu $(u, v) \in E$ nejvýše jednou. Proto cyklus `FOR` vyžaduje celkem čas $O(|E|)$. Při každém průběhu cyklu `WHILE` odstraníme z fronty Q jeden vrchol, celkový počet průběhů cyklu `WHILE` se tak rovná $|V|$. Při každém průběhu musíme najít vrchol $v \in Q$ s nejmenší hodnotou $d[v]$. To vyžaduje prohlédnout všechny vrcholy fronty Q, která má nejvýše $|V|$ prvků. Toto prohlížení vyžaduje čas $O(|V|)$. Celkem tedy `WHILE` cyklus vně cyklu FOR vyžaduje čas $O(|V|^2)$. Celkem tak Dijkstrův algoritmus vyžaduje čas $O(|V|^2 + |E|)$. □

Věta 13.7 *V okamžiku, kdy je v Dijkstrově algoritmu vrchol v zařazen do množiny S, platí rovnost $d[v] = \delta(s, v)$. Po skončení Dijkstrova algoritmu tak pro každý vrchol v platí $d[v] = \delta(s, v)$.*

Důkaz. Budeme postupovat sporem. Předpokládejme, že existuje vrchol $u \in V$, pro který v okamžiku zařazení do množiny S platí $d[u] \neq \delta(s, u)$. Buď u_0 první vrchol, pro který v okamžiku zařazení do S platí $d[u_0] \neq \delta(s, u_0)$. Potom platí

- $u_0 \neq s$,

neboť s je první vrchol zařazený do S a to v okamžiku, kdy $d[s] = 0 = \delta(s, s)$. Dále platí, že

- existuje nějaká cesta z s do vrcholu u_0 v grafu G,

protože v opačném případě by platilo $\delta(s, u_0) = \infty = d[u_0]$ v okamžiku zařazení u_0 do množiny S.

Uvažujme nyní nejkratší cestu z s do u_0. Protože platí $s \in S$ a zatím $u_0 \notin S$, existuje v této cestě první vrchol y, pro který platí $y \notin S$. Jeho předchůdce označíme x. Protože $x \in S$, platí $d[x] = \delta(s, x)$ protože u_0 byl první vrchol, pro který v okamžiku zařazení do S platila nerovnost $d[u_0] \neq \delta(s, u_0)$.

Dále si uvědomme, že nějaká nejkratší cesta z s do y musí procházet vrcholem y. (V opačném případě by totiž bylo možné zkrátit zvolenou nejkratší cestu z s do u_0 procházející vrcholem x.) To ale znamená, že

$$\delta(s, y) = \delta(s, x) + w(x, y) = d[x] + w(x, y).$$

V okamžiku zařazení x do množiny S bzyla ale hodnota $d[y]$ upravena na $d[y] := d[x] + w(x, y) = \delta(s, y)$, pokud již $d[y]$ tuto hodnotu neměla.

To znamená

$$d[y] = \delta(s, y) \leq \delta(s, u_0) \leq d[u_0],$$

neboť každá hrana má nezápornou váhu a protože $d[u_0]$ je v každém okamžiku horním odhadem pro vzdálenost $\delta(s, u_0)$.

Naopak platí $d[u_0] \leq d[y]$, neboť u_0 a y nejsou v tomto okamžiku prvky S a vrchol u_0 byl vybrán pro zařazení do S z fronty Q, protože hodnota $d[u_0]$ byla menší nebo rovná d-hodnotě libovolného jiného vrcholu z fronty Q. Tím je dokázána rovnost $d[u_0] = d[y] = \delta(s, u_0)$, což je ve sporu s výběrem vrcholu u_0. □