IV003 Algoritmy a datové struktury II

Ivana Černá

Fakula informatiky, Masarykova univerzita

Jaro 2014

Technické řešení této výukové pomůcky je spolufinancováno Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OP Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

George Pólya: How to Solve It?

"What is the difference between method and device?
A method is a device which you used twice."

Obsah

Literatúra

- T. Cormen, Ch. Leiserson, R. Rivest, C. Stein:*Introduction to Algorithms*. Second Edition. MIT Press, 2001
- J. Kleinberg, and E. Tardos:*Algorithm Design*. Addison-Wesley, 2006
- S. Dasgupta, Ch. Papadimitriou, U. Vazirani:*Algorithms*. McGraw Hill, 2007.
- *viz Interaktívna osnova predmetu*

obrázky použité v prezentáciach, pokiaľ nie je explicitne uvedené inak, sú prevzaté z publikácie T. Cormen, Ch. Leiserson, R. Rivest, C. Stein: Introduction to Algorithms*. Second Edition. MIT Press, 2001*

Elektronická podpora v IS MU

- Interaktívna osnova predmetu

  https://is.muni.cz/auth/el/1433/jaro2014/IV003/index.qwarp
- Diskusné fórum predmetu

kompletné informácie o organizácii prednášky sú uvedené v Interaktívnej osnove predmetu

Zložitosť

1. Zložitosť
   - Zložitosť problémov a algoritmov
   - Analýza algoritmu
   - Amortizovaná zložitosť
2. Dátové štruktúry
3. Metódy návrhu algoritmov
4. Grafové algoritmy
5. Algoritmy pre prácu s reťazcami

Zložitosť problémov a algoritmov

Zložitosť algoritmu

- výpočtový model
- zložitostné kritérium
- zložitosť ako funkcia dĺžky vstupu
- asymptotická zložitosť algoritmu
- typy zložitosti
  - zložitosť v najlepšom prípade
  - zložitosť v najhoršom prípade
  - priemerná zložitosť
  - očakávaná zložitosť (vážený priemer)
- zložitosť problému vs. zložitosť algoritmu.

Zložitosť problému

- dolný odhad zložitosti problému
  - dôkazové techniky
- horný odhad zložitosti problému
  - zložitosť konkrétneho algoritmu pre daný problém
- zložitosť problému

Dolný odhad zložitosti problému - techniky

- Informačná metóda
- Metóda redukcie
- Metóda sporu

Dolný odhad zložitosti problému - Informačná metóda

Princíp

riešenie problému v sebe obsahuje isté množstvo informácie
v každom kroku výpočtu sme schopní určiť len časť tejto informácie

permutácie vygenerovať všetky permutácie n-prvkovej postupnosti
počet rôznych permutácií je $n!$ a preto dolný odhad je $\Omega(n!)$
(*zložitosť problému je* $\Theta(n!)$)

polynóm evaluovať $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ v bode x
dolný odhad $\Omega(n)$ – spracovanie všetkých koeficientov
(*zložitosť problému je* $\Theta(n)$)

matice násobenie dvoch celočíselných matíc rozmerov $n \times n$
dolný odhad $\Omega(n^2)$ – veľkosť výstupu (*horný dohad ??*)

TSP dolný odhad $\Omega(n^2)$ – počet hrán
(*najlepší známy algoritmus je exponenciálny*)

problém: atomizácia informácie

Rozhodovacie stromy

- model výpočtu
- vnútorný vrchol stromu zodpovedá *dotazu* na vstupnú inštanciu
- hrany zodpovedjú možným odpovediam na dotaz
- list stromu zodpovedá výstupu

príklad: vyhľadávanie v B-strome

Rozhodovacie stromy - dolný odhad zložitosti

- časová zložitosť algoritmu reprezentovaného rozhodovacím stromom je dĺžka najdlhšej cesty z koreňa do listu
- ak vstup má N možných riešení (výstupov), tak strom musí mať (aspoň) N listov
- ak každý dotaz má (nanajvýš) k rôznych odpovedí, žak hĺbka stromu musí byť aspoň $\lceil \log_k N \rceil = \Omega(\log N)$
- $\Omega(\log N)$ je dolným odhadom pre časovú zložitosť problému

Rozhodovacie stromy - slovník

- vstup: utriedené pole $A[1 \dots n]$, číslo x
- úloha: rozhodnúť, či x sa vyskytuje v A a ak áno, tak určiť jeho pozíciu
- pre vstup x je n možných riešení (n možných pozícií, na ktorých môže byť x uložené)
- dolný odhad $\Omega(\log n)$ pre problém slovníku
- dolný odhad je *tesný*, viz algoritmus pre vyhľadávanie v utriedenom poli

Rozhodovacie stromy - triedenie

- vstup: postupnosť $(x_1, \ldots, x_n)$ navzájom rôznych čísel
- výstup: permutácia Π taká, že $x_{\Pi(1)} < x_{\Pi(2)} < \ldots < x_{\Pi(n)}$
- počet rôznych permutácií je $n!$
- triedenie k-árnym rozhodovacím stromom $\Rightarrow$ dolný odhad $\Omega(\log n!)$
- pre zjednodušenie použijeme Stirlingovu aproximáciu

$$n! = \left(\frac{n}{e}\right)^n \sqrt{2\Pi n}\left(1 + \Theta\left(\frac{1}{n}\right)\right) > \left(\frac{n}{e}\right)^n$$

$$\lceil \log_k(n!) \rceil > \left\lceil \log_k \left(\frac{n}{e}\right)^n \right\rceil = \lceil n \log_k n - n \log_k e \rceil = \Omega(n \log n)$$

Dolný odhad zložitosti problému - Redukcia

Princíp

poznáme dolný odhad pre Q
Q redukujeme na P
dolný odhad pre Q je aj dolným odhadom pre P

Q opakuje sa niektoré číslo v danej postupnosti čísel $x_1, \ldots x_n$?

P minimálna kostra v Euklidovskom grafe

redukcia body $(x_1, 0), \ldots (x_n, 0)$
v postupnosti sa opakuje číslo práve keď mimimálna kostra obsahuje hranu dĺžky 0

záver dolný odhad $\Omega(n \log n)$ pre problém P

používa sa pre porovnanie relatívnej obtiažnosti problémov

Dolný odhad zložitosti problému - Metóda sporu

Princíp

Varianta A za predpokladu, že algoritmus má zložitosť menšiu než uvažovanú hranicu, vieme skonštruovať vstup, pre ktorý nevypočíta korektné riešenie.
Varianta B za predpokladu, že algoritmus nájde vždy korektné riešenie, vieme skonštruovať vstup, pre ktorý zložitosť výpočtu presiahne uvažovanú hranicu.

Problém i-teho prvku postupnosti

Vstup postupnosť $x = (x_1, \ldots, x_n)$ navzájom rôznych čísel a číslo $i \in \mathbb{N}$
Úloha nájsť i-ty prvok v poradí v postupnosti x

Príklad: pre $x = (7, 9, 3, 2, 5)$ *a* $i = 4$ *je hľadaným číslom 7*

metóda rozhodovacieho stromu dáva dolný odhad $\Omega(\log n)$ *???*

Dolný odhad, maximálny prvok, metóda sporu

Lema 1

Nech algoritmus $\mathcal{A}$ je založený na porovnávaní prvkov a nech $\mathcal{A}$ rieši problém maximálneho prvku. Potom $\mathcal{A}$ musí na každom vstupe vykonať aspoň $n-1$ porovnaní.

Dôkaz (Varianta A)
Nech $x = (x_1, \ldots, x_n)$ je vstup dĺžky n, na ktorom $\mathcal{A}$ vykoná menej než $n-1$ porovnaní a nech x_r je maximálny prvok v x.
Potom v x musí existovať prvok x_p taký, že $p \neq r$ a v priebehu výpočtu x_p nebol porovnávaný so žiadnym prvkom väčším než on sám. Existencia takého prvku plynie z počtu vykonaných porovnaní.
Ak v x zmeníme hodnotu prvku x_p na $x_r + 1$, tak $\mathcal{A}$ určí ako maximálny prvok x_r – spor. $\square$

Metóda protivníka

- alternatívna prezentácia metódy sporu (varianta A)
- na výpočet sa pozeráme ako na hru medzi *protivníkom* a algoritmom
- protivník si volí vstupnú inštanciu
- algoritmus kladie protivníkovi dotazy na vstupnú inštanciu, protivník na dotazy odpovedá tak, aby prinútil algoritmus urobiť *čo najviac práce*
- ak algoritmus urobí počas výpočtu málo dotazov na vstup, tak existuje niekoľko vstupov, ktoré sú konzistentné s položenými dotazmi (a odpoveďami)
- protivník si vyberie ten vstup, ktorý je v spore s výstupom algoritmu

Metóda protivníka - maximálny prvok

- protivník si zvolí vstup $x_i = i$ pre $i = 1, \ldots, n$
- vždy, keď protivník poskytne algoritmu informáciu $x_i < x_j$, označí si x_i, pretože algoritmus vie (mal by vedieť), že x_i nemôže byť maximálny prvok
- prvok x_n nie je nikdy označený
- ak algoritmus počas výpočtu urobí menej ako $n - 1$ dotazov, musí existovať ďalší neoznačený prvok $x_k \neq x_n$
- protivník môže zmeniť hdontu x_k na $n + 1$
- oba vstupy sú konzistentné s položenými dotazmi a odpoveďami, ale algoritmus dá pre oba vstupy rovnaký výstup

Metóda protivníka - minimálny a maximálny prvok

- protivník si označí každý prvok dvomi značkami: $+$ znamená, že daný prvok môže byť maximálný, - znamená, že môže byť minimálny
- ak algoritmus porovná 2 prvky s dvomi značkami, protivník každému prvku odstráni 1 značku
- vo všetkých ostaných prípadoch môže protivník zodpovedať dotaz tak, že odstráni maximálne jednu značku
- ak by na konci výpočtu zostala dvom prvok značka $+$ resp. -, tak algoritmus nemôže dať korektný výstup
- nanajvýš $\lfloor n/2 \rfloor$ dotazov odstráni 2 značky, ostatné dotazy odstránia maximálne jednu značku
- na začiatku výpočtu je $2n$ značiek, na konci môžu zostať len 2, preto celkový počet dotazov musí byť aspoň $2n - 2 - \lfloor n/2 \rfloor = \lceil 3n/2 \rceil$
- dolný odhad je tesný *(viz sekcia Rozdeľ a panuj)*

Metóda protivníka - i-ty prvok postupnosti

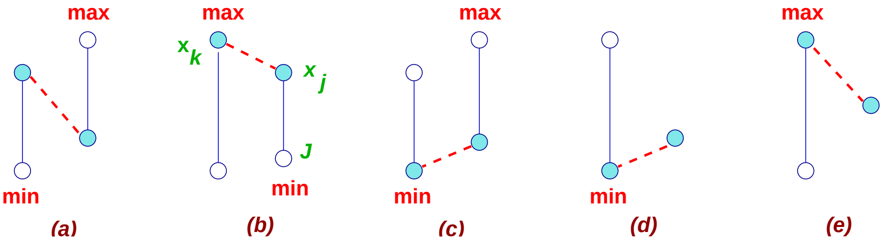
Veta 2

Každý algoritmus založený na porovnávaní prvkov, ktorý rieši problém i-teho prvku postupnosti n prvkov pre $1 \leq i \leq \lceil n/2 \rceil$, musí urobiť pre každú vstupnú postupnosť aspoň $n + i - 2$ porovnaní.

Dôkaz. Označme $M(n, i)$ počet porovnaní potrebných na nájdenie i-teho prvku n prvkovej postupnosti čísel. Dokážeme, že pre každé n a každé $1 \leq i \leq \lceil n/2 \rceil$ platí $M(n, i) \geq n + i - 2$. Pre $i > \lceil n/2 \rceil$ platí $M(n, i) = M(n, n - i + 1)$, stačí ak čísla v postupnosti prenásobíme číslom -1.

Nech $\mathcal{B}$ je fixovaný algoritmus pre problém i-teho prvku a $(x = (x_1, \ldots, x_n),\ i)$ jeho fixovaný vstup. Uvážme výpočet $\mathcal{B}$ na x až do okamihu, keď sa prvý krát porovnávajú prvky x_k a x_j také, že aspoň jeden z nich už bol porovnávaný s niektorým ďalším prvkom postupnosti.

Môže nastať niekoľko prípadov. Porovnávame maximá dvoch usporiadaných párov (prípad (b) na obrázku), minimá dvoch usporiadaných párov (prípad (c)), minimum jedného a maximum druhého usporiadaného páru (prípad (a)), minimum jedného páru s prvkom, ktorý ešte nebol porovnávaný (prípad (d)) a maximum jedného páru s prvkom, ktorý ešte nebol porovnávaný (prípad (e)).



V prípade (b) sa porovnávajú maximá dvoch predošlých porovnaní. Uvážme postupnosť $\tilde{x}$ ktorá vznikne z x tak, že číslo x_k nahradíme číslom `max`, ktoré bude v $\tilde{x}$ maximálnym a číslo J nahradíme číslo `min`, ktoré bude minimálnym. Výpočty $\mathcal{B}$ na x a $\tilde{x}$ sú až do uvažovaného porovnania zhodné a preto výpočet $\mathcal{B}$ na $\tilde{x}$ obsahuje 3 porovnania, v ktorých vystupujú `max` a `min`.

Uvážme postupnosť $\tilde{\tilde{x}}$, ktorá vznikne z $\tilde{x}$ odstránením prvkov `max` a `min`. i-ty prvok postupnosti $\tilde{x}$ je zhodný s $(i-1)$-vým prvkom $\tilde{\tilde{x}}$. Preto $\mathcal{B}$ na $\tilde{x}$ musí urobiť aspoň toľko porovnaní prvkov rôznych od `max` a `min` ako $\mathcal{B}$ na $\tilde{\tilde{x}}$, tj. aspoň $M(n-2, i-1)$.

Spolu dostávame, že $\mathcal{B}$ na $\tilde{x}$ musí urobiť aspoň $3 + M(n-2, i-1)$ porovnaní, tj.

$$M(n,i) \quad \geq 3 + M(n-2, i-1) \tag{1}$$

Analogický vzťah platí pre situácie (a) a (c).
Pre prípad (e) odvodíme

$$M(n,i) \quad \geq 2 + M(n-1, i) \tag{2}$$

a pre prípad (d)

$$M(n,i) \quad \geq 2 + M(n-1, i-1). \tag{3}$$

Indukciou dokážeme, že $M(n,i) \geq n+i-2$.
Tvrdenie platí pre $n=2$ a $i=1$, pretože $M(2,1)=1$. Predpokladajme, že pre všetky $m<n$ a $r \leq \lceil m/2 \rceil$ platí $M(m,r) \geq m+r-2$. Ukážeme že platí pre n a všetky $i \leq \lceil n/2 \rceil$. Pretože $i-1 \leq \lceil n/2 \rceil - 1 = \lceil (n-2)/2 \rceil$, tak v prípade (1)

$$\begin{aligned} M(n,i) &\geq M(n-2,i-1)+3 \\ &\geq (n-2)+(i-1)-2+3 \quad \text{podľa ind.predpokladu} \\ &= n+i-2 \end{aligned}$$

V prípade (2) ak $i \leq \lceil (n-1)/2 \rceil$ tak

$$\begin{aligned} M(n,i) &\geq M(n-1,i)+2 \\ &\geq (n-1)+i-2+2 \quad \text{podľa ind.predpokladu} \\ &> n+i-2 \end{aligned}$$

Ak n je liché a $i = (n+1)/2$, tak $n - i = \lceil (n-1)/2 \rceil$ a preto

$$\begin{aligned}
M(n,i) &\geq M(n-1,i) + 2 \\
&= M(n-1, n-i) + 2 \\
&\geq (n-1) + (n-i) - 2 + 2 \quad \text{podľa ind.predpokladu} \\
&= 2n - i - 1 \\
&= n + i - 2 \quad \text{pretože } n = 2i - 1
\end{aligned}$$

V prípade (3) využijeme $i - 1 \leq \lceil n/2 \rceil - 1 \leq \lceil (n-1)/2 \rceil$

$$\begin{aligned}
M(n,i) &\geq M(n-1, i-1) + 2 \\
&\geq (n-1) + i - 1 - 2 + 2 \quad \text{podľa ind.predpokladu} \\
&= n + i - 2
\end{aligned}$$

Analýza algoritmu

Analýza algoritmu

korektnosť	čiastočná korektnosť
	konečnosť
zložitosť	

Analýza algoritmu – SELECT

- algoritmus SELECT pre problém *i*-teho prvku
- využíva princíp rekurzie
- zložitosť algoritmu je lineárna voči dĺžke postupnosti

Medián postupnosti $x = (x_1, \ldots, x_n)$ navzájom rôznych čísel je $\lceil n/2 \rceil$-ty prvok v poradí v postupnosti x (*prostredný prvok*)

Algoritmus SELECT pre problém i-teho prvku

1. n prvkov rozdeľ do $\lceil \frac{n}{5} \rceil$ skupín , z ktorých každá (s možnou výnimkou jednej skupiny) obsahuje 5 prvkov.
2. nájdi medián každej z $\lceil \frac{n}{5} \rceil$ skupín
3. rekurzívne volaj SELECT pre postupnosť mediánov nájdených v kroku 2 a hodnotu $\lceil \lceil \frac{n}{5} \rceil / 2 \rceil$ (t.j. nájdi medián d z $\lceil \frac{n}{5} \rceil$ mediánov nájdených v kroku 2)
4. prvky postupnosti x (okrem d) rozdeľ do 2 skupín
   skupina 1 prvky menšie než d (označme ich počet m)
   skupina 2 prvky väčšie než d (ich počet je $n - m - 1$)
5. ak $m + 1 = i$ tak d je hľadaný i-ty prvok postupnosti x
   ak $i \leq m$ tak rekurzívne volaj SELECT pre skupinu 1 a číslo i
   ak $i > m + 1$ tak rekurzívne volaj SELECT pre skupinu 2 a číslo $i - m - 1$

Algoritmus Select – korektnosť

domáca úloha

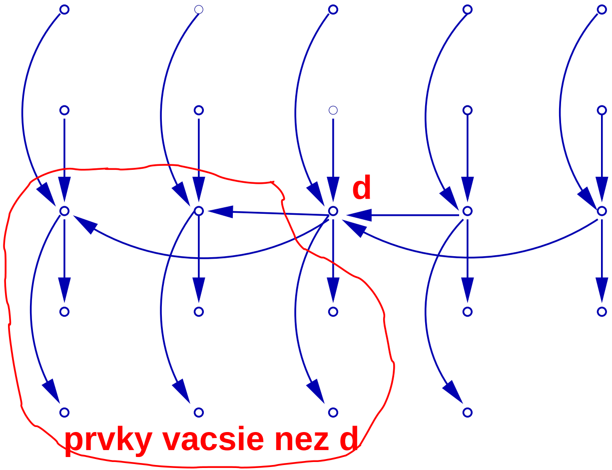
Algoritmus SELECT – analýza zložitosti

1. $\mathcal{O}(n)$
2. $\mathcal{O}(n)$ (medián 5-tich prvkov nájdeme v konštantnom čase)
3. $T(\lceil \frac{n}{5} \rceil)$
4. $\mathcal{O}(n)$
5. závisí od počtu prvkov menších (skupina 1) resp. väčších než d (skupina 2)
   buď $T(m)$ alebo $T(n-m-1)$.

funkcia $T(k)$ označuje zložitosť algoritmu SELECT pre postupnosť k prvkov

Algoritmus SELECT – analýza zložitosti

Odhad počtu prvkov menších a väčších než medián d



Algoritmus $\textsc{Select}$ – analýza zložitosti

Mohutnosť skupiny 1: každá skupina, ktorá má medián aspoň d, obsahuje (aspoň) 3 prvky väčšie než d
(s výnimkou skupiny obsahujúcej d a skupiny, ktorá nemusí byť úplná)
počet prvkov väčších než d je preto aspoň

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5}\right\rceil\right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

záver: počet prvkov menších než d je maximálne $\frac{7n}{10} + 6$

Mohutnosť skupiny 2: analogicky odvodíme, že počet prvkov menších než d je aspoň $\frac{3n}{10} - 6$. Z toho plynie, že počet prvkov väčších než d je maximálne $\frac{7n}{10} + 6$.

v najhoršom prípade sa v bode 5. procedúra $\textsc{Select}$ volá pre postupnosť $\frac{7n}{10} + 6$ prvkov

Algoritmus SELECT – analýza zložitosti

$$T(n) = \begin{cases} f & \text{pre n} \leq 80, \\ & f \text{ je vhodná konštanta} \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + en & n > 80 \end{cases}$$

Indukciou k n dokážeme $T(n) \leq cn$, kde c je vhodná konštanta.

1. pre $n \leq 80$ tvrdenie platí
2.

$$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c(7n/10 + 6) + en \\ &\leq cn/5 + c + 7cn/10 + 6c + en \\ &= 9cn/10 + 7c + en \\ &= cn + (-cn/10 + 7c + en) \end{aligned}$$

Ak zvolíme c tak, aby $-cn/10 + 7 + en \leq 0$, dostávame $T(n) \leq cn$.

Problém párovania

motivácia dve skupiny objektov, hľadáme ich priradenie (párovanie), ktoré rešpektuje vzájomné preferencie

príklady absolventi vs. zamestnávatelia, zoznamovacie agentúry

problém nestabilita: *A_1 prijme zamestnanie u Z_1, potom mu ale miesto ponúkne Z_2 a od Z_1 odíde; Z_1 hľadá nového absolventa . . .*

Formulácia problému

množiny $M = \{m_1, \ldots, m_n\}$ a $W = \{w_1, \ldots, w_n\}$ (muži a ženy)

párovanie je množina S usporiadaných dvojíc z $M \times W$ taká, že každé $m \in M$ a $w \in W$ sa vyskytuje maximálne v jednej dvojici

úplné párovanie je párovanie, v ktorom sa každý prvok z M a W vyskytuje práve v jednej dvojici

preferencie: každý muž $m \in M$ hodnotí (usporiada) všetky ženy. *m preferuje w pred w'* práve ak w hodnotí vyššie než w'. Nie je povolené, aby dve ženy mali rovnaké hodnotenie. Symetricky sa definujú preferencie pre ženy.

nestabilita: S je úplné párovanie, $(m, w), (\overline{m}, \overline{w}) \in S$,
m preferuje $\overline{w}$ pred w a zároveň $\overline{w}$ preferuje m pred $\overline{m}$
dvojica $(m, \overline{w})$ je *nestabilitou* voči S

stabilné párovanie je úplné párovanie S také, že neexistuje nestabilita voči S

Formulácia problému

problém stabilného párovania

- existuje stabilné párovanie pre každú množinu preferenčných zoznamov?
- ak existuje stabilné párovanie, dokážeme ho efektívne skonštruovať?

príklad

$$\begin{array}{llcll} m: & w > \overline{w} & \qquad & w: & m > \overline{m} \\ \overline{m}: & w > \overline{w} & & \overline{w}: & m > \overline{m} \end{array}$$

stabilné párovanie $(m, w), (\overline{m}, \overline{w})$
nestabilné párovanie $(m, \overline{w}), (\overline{m}, w)$

Algoritmus Gale - Shapley

iniciálne nastavenie: všetci $m \in M$ a $w \in W$ sú voľní

```
while existuje voľný muž m, ktorý ešte nedal návrh všetkým ženám do
        vyber m uvedených vlastností
        nech w je žena s najvyššou preferenciou, ktorej ešte m nedal návrh
        if w je voľná
           then (m, w) vytvoria dvojicu
            else w tvorí dvojicu s m̄
                 if w preferuje m̄ pred m
                    then m zostane voľný
                     else (m, w) vytvoria dvojicu
                          m̄ sa stane voľný
                 fi
        fi
od
return množina S vytvorených dvojíc
```

Analýza G-S algoritmu - zložitosť

Lema 3

G-S algoritmus skončí po nanajvýš n^2 iteráciach **while** *cyklu.*

Dôkaz: Definujeme *mieru progresu* ako funkciu $\mathcal{P}$ takú, že $\mathcal{P}(t)$ je počet takých dvojíc (m, w), že m dal návrh w v priebehu prvých t iterácii **while** cyklu. Zrejme $\mathcal{P}$ je rastúca funkcia a existuje len n^2 rôznych párov (m, w). Preto počet iterácii **while** cyklu je zhora ohraničený hodnotou n^2. $\square$

Analýza G-S algoritmu - korektnosť

1. w je od okamžiku prvého návrhu neustále súčasťou páru a jej partneri majú postupne vyššiu a vyššiu preferenciu
2. postupnosť žien, ktorým dáva m návrh má klesajúce preferencie
3. ak m je v niektorom okamihu výpočtu algoritmu voľný, tak existuje žena, ktorej ešte nedal návrh
   (ak by taká neexistovala, tak podľa 1. tvoria všetky pár a nemôže existovať voľný muž)
4. množina S, ktorú vypočíta algoritmus, tvorí úplné párovanie
   (ak by na konci existoval voľný muž, tak musel urobiť návrhy všetkým ženám, čo je v spore s 3.)

Analýza G-S algoritmu - korektnosť

Lemma 4

Algoritmus G-S vypočíta stabilné párovanie

Dôkaz. Podľa 4. je vypočítaná množina S úplným párovaním. Predpokladajme, že existuje nestabilita voči S tvorená pármi (m, w) a $(\overline{m}, \overline{w})$. Z konštrukcie algoritmu plynie, že posledný návrh, ktorý m vo výpočte urobil, bol návrh w.
Ak m pred posleným návrhom nedal návrh $\overline{w}$, tak m preferuje w pred $\overline{w}$, čo je spor s predpokladom o nestabilite (*konkrétne s tým, že m preferuje $\overline{w}$ pred w*).
Ak m pred posleným návrhom dal návrh $\overline{w}$, tak $\overline{w}$ ho odmietla a vytvorila pár s $\widetilde{m}$, ktorého preferovala viac (podľa 1.). Na konci výpočtu tvorí $\overline{w}$ pár s $\overline{m}$. Buď $\overline{m} = \widetilde{m}$ alebo $\overline{w}$ preferuje $\overline{m}$ pred $\widetilde{m}$ (podľa 1.). V oboch prípadoch dostávame spor s predpokladom o nestabilite (*konkrétne s tým, že $\overline{w}$ preferuje m pred $\overline{m}$*). $\square$

Nobelova cena za ekómiu, 2012

Obrázek: Loyd S. Shapley, Alvin E. Roth

Amortizovaná zložitosť

Amortizovaná zložitosť

Technika umožňujúca presnejšie určenie zložitosti.
Uvažujme výpočet, v ktorom sa postupne vykonajú operácie $I_1, \ldots, I_n$.

Klasický prístup

Analyzujeme zložitosť každej operácie. Výsledná zložitosť je súčtom zložitostí jednotlivých operácií.

Technika amortizácie

Analyzujeme postupnosť ako celok.

Používajú sa metódy

- zoskupení
- účtov
- potenciálových funkcií

Príklad - zásobník

operácie $\text{Push}(S, x)$, $\text{Pop}(S)$, $\text{Multipop}(S, k)$

$\text{Push}(S, x)$ vloží do zásobníka S prvok x

$\text{Pop}(S)$ vyberie vrchný prvok zo zásobníka S

$\text{Multipop}(S, k)$ vyberie k prvkov z S resp. vyprázdni zásobník ak $\mid S \mid < k$

Uvažujme postupnosť n operácií, každá operácia je Pop, Push alebo Multipop.

Push a Pop majú zložitosť 1.
V postupnosti n operácií má operácia Multipop v najhoršom prípade zložitosť n.

Postupnosť n operácií má v najhoršom prípade zložitosť n^2.

Príklad - binárne počítadlo

- k-bitové počítadlo implementované ako pole $A[0 \dots k-1]$
- hodnota počítadla je $x = \sum_{i=0}^{k-1} A[i]2^i$
- iniciálna hodnota počítadla je 0
- operácia $\textsc{Inc}(A)$ zvýši hodnotu počítadla o 1
- $\textsc{Inc}(A)$
  $i \leftarrow 0$
  while $i \leq k-1 \wedge A[i] = 1$ **do** $A[i] \leftarrow 0; \quad i \leftarrow i+1$ **od**
  if $i \leq k-1$ **then** $A[i] \leftarrow 1$ **fi**

Zložitosť operácie $\textsc{Inc}$ je počet preklopených bitov v A, tj. maximálne k.

Zložitosť postupnosti n operácií $\textsc{Inc}$ je v najhoršom prípade $n \cdot k$ (pre $n < 2^{k+1}$).

Metóda zoskupení

Operácie rozdelíme do skupín a analyzujeme zložitosť celej skupiny operácii súčasne.

Zásobník

- Skupina 1 operácie Push: súčet ich zložitostí nepresiahne n
- Skupina 2 operácie Pop a Multipop: súčet ich zložitostí (= *počet prvkov vybraných zo zásobníka*) nemôže byť väčší než počet vykonaných operácií Push (= *počet prvkov vložených do zásobníka*). Zložitosť celej skupiny preto nepresiahne n.

celá postupnosť n operácií má v najhoršom prípade zložitosť $2n$

Metóda zoskupení

Binárne počítadlo

Skupina 1 preklopenie bitu $A[0]$
realizuje sa pri každom volaní $\textsc{Inc}$
celková zložitosť skupiny je n

Skupina 2 preklopenie bitu$A[1]$
realizuje sa pri každom druhom volaní $\textsc{Inc}$
celková zložitosť skupiny je $\lfloor \frac{n}{2} \rfloor$

Skupina 3 preklopenie bitu $A[2]$
realizuje sa pri každom štvrtom volaní $\textsc{Inc}$
celková zložitosť skupiny je $\lfloor \frac{n}{4} \rfloor$

... ...

počet skupín je $k-1$
pre $n < 2^{k+1}$ je počet skupín nanajvýš $\lfloor log n \rfloor$

postupnosť n operácii $\textsc{Inc}$ má zložitosť $\sum_{i=0}^{\lfloor log n \rfloor} \lfloor n/2^i \rfloor < 2n$

Metóda účtov

Každej operácii priradíme kredit (číslo), ktoré môže byť rôzne od jej skutočnej ceny (zložitosti).

Pri realizácii operácie zaplatíme jej skutočnú cenu kreditmi podľa nasledovných pravidiel:

- ak cena operácie je menšia alebo rovná kreditu operácie, tak za operáciu zaplatíme toľko kreditov, aká je jej cena, a zvyšné kredity uložíme na účet,
- ak cena operácie je väčšia ako kredit operácie, tak kredity potrebné na zaplatenie operácie vezmeme z účtu

Metóda účtov

Počiatočný stav účtu je 0 kreditov.

Ak počas celého výpočtu je počet kreditov na účte nezáporný,
tak súčet kreditov vykonaných operácií je $\geq$ cena vykonaných operácií.

Varianta

Kredity priradíme objektom dátovej štruktúry, nad ktorou sa operácie realizujú. Cena operácie sa zaplatí kreditmi objektov, s ktorými operácia manipuluje.

Terminológia

Amortizovaná cena operácie $=$ počet kreditov priradených operácii.

Metóda účtov

Zásobník

Operácia	Cena	Kredity
PUSH	1	2
POP	1	0
MULTIPOP	$\min\{k, \lvert S \rvert\}$	0

v okamihu, keď objekt vkladáme do zásobníka, tak si predplatíme *jeho výber*

Operáciu PUSH zaplatíme 1 kreditom, 1 kredit dáme na účet.
Operácie POP a MULTIPOP zaplatíme kreditmi z účtu.

Ľahko overíme, že počas celého výpočtu platí invariant počet kreditov na účte je rovný počtu prvkov v zásobníku.
Z toho plynie, že zostatok na účte nikdy neklesne pod 0.

Celková zložitosť postupnosti n operácií je $\leq$ súčet kreditov vykonaných operácií.
Súčet kreditov vykonaných operácií je $\leq 2n$.

Metóda účtov

Binárne počítadlo

Operácia	Cena	Kredity
Nastavenie bitu na 1	1	2
Nastavenie bitu na 0	1	0

Operáciu nastavenie hodnoty premennej $A[i]$ na 1 zaplatíme jedným kreditom a zároveň 1 kredit dáme na účet premennej $A[i]$.
Operáciu nastavenia hodnoty $A[i]$ na 0 zaplatíme kreditom, ktorý má na účte premenná $A[i]$.

Počas celého výpočtu platí invariant ak hodnota premennej $A[i]$ je 1, tak premenná má na svojom účte 1 kredit.

Počas operácie INC sa pre maximálne jednu premennú mení hodnota 1. Preto amortizovaná cena operácie INC je 2.

Cena zložitosti postupnosti n operácií INC je $2n$.

Metóda potenciálovej funkcie

Operácie sa realizujú nad dátovou štruktúrou. Potenciálová funkcia Φ priradí každej hodnote (obsahu) dátovej štruktúry číslo.

Uvažujme postupnosť n operácií. Nech skutočná cena i-tej operácie v tejto postupnosti je c_i.

Označme D_0 iniciálnu hodnotu dátovej štruktúry a D_i jej hodnotu po vykonaní i-tej operácie.

Definujeme amortizovanú cenu i-tej operácie $\hat{c}_i$ predpisom

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

Súčet amortizovaných cien operácií postupnosti je

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

Za predpokladu $\Phi(D_n) \geq \Phi(D_0)$ platí

$$\textstyle\sum_{i=1}^{n} \hat{c}_i \geq \displaystyle\sum_{i=1}^{n} c_i$$

t.j. súčet amortizovaných cien operácií je horným odhadom pre zložitosť celej postupnosti operácií.

Aby sme zabezpečili platnosť podmienky $\Phi(D_n) \geq \Phi(D_0)$, definujeme potenciálovú funkciu tak, aby pre každú hodnotu D dátovej štruktúry platilo, že jej potenciál $\Phi(D)$ je aspoň tak veľký, ako potenciál počiatočnej hodnoty D_0 dátovej štruktúry.

Metóda potenciálovej funkcie

Zásobník

Dátová štruktúre je zásobník.
Potenciálová funkcia = počet prvkov v zásobníku.

$$
\begin{array}{lll}
\textsc{Push} & \hat{c}_i & = 1 + (|S| + 1) - |S| = 2 \\
\textsc{Pop} & \hat{c}_i & = 1 + |S| - (|S| + 1) = 0 \\
\textsc{Multipop} & \hat{c}_i & = \begin{cases} k + (|S| - k) - |S| & = 0 \quad \text{ak} \quad |S| > k \\ |S| + 0 - |S| & = 0 \quad \text{ak} \quad |S| \leq k \end{cases}
\end{array}
$$

Pre všetky $1 \leq i \leq n$ platí $\Phi(D_i) \geq \Phi(D_0)$.

Zložitosť postupnosti n operácií je $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \leq 2n$

Metóda potenciálovej funkcie

Binárne počítadlo

Dátová štruktúra je pole $A[0 \dots k-1]$.
Potenciálová funkcia je počet 1 v poli.

Označme b_i počet 1 v poli $A[0 \dots k-1]$ po vykonaní i-tej operácie $\textsc{Inc}$.

Skutočná cena i-tej operácie $\textsc{Inc}$ je $t_i + 1$, kde t_i je počet bitov preklopených z 1 na 0.
Amortizovaná cena i-tej operácie je

$$\hat{c}_i = t_i + 1 + (b_{i-1} - t_i + 1) - b_{i-1} = 2.$$

Potenciálová funkcia je vždy nezáporná, potenciálová funkcia pre počiatočnú hodnotu poľa je 0.

Zložitosť postupnosti n operácií $\textsc{Inc}$ je $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \leq 2n$.

Dynamické tabuľky

Podporované operácie sú TABLE INSERT (vloží do tabuľky novú položku, položky sa ukladajú lineárne za seba) a TABLE DELETE (z tabuľky odstráni poslednú položku). Dopredu nie je známy počet položiek a teda ani veľkosť tabuľky, ktorú máme alokovať.

Dynamické riešenie: pri naplnení tabuľky alokujeme novú, väčšiu tabuľku a položky do nej presunieme. Analogicky, ak po odobraní položiek je tabuľka nenaplnená, alokujeme novú, menšiu tabuľku a položky do nej presunieme.

Pre neprázdnu tabuľku T definujeme faktor $\alpha(T)$ ako pomer počtu položiek uložených v tabuľke a veľkosti tabuľky. Prázdna tabuľka (tj. tabuľka, ktorá nemá žiadne sloty) má veľkosť 0 a jej faktor je 1.

Dynamické tabuľky - skutočná cena

- na začiatku alokujeme prázdnu tabuľku
- do tabuľky, ktorej faktor je < 1, môžeme vložiť novú položku
- ak chceme vložiť novú položku do tabuľky s faktorom 1, alokujeme najprv novú tabuľku s dvojnásobnou veľkosťou, položky starej tabuľky do nej presunieme a vložíme novú položku
- ak chceme odobrať položku z tabuľky, ktorej faktor je $\leq 1/2$, tak alokujeme novú tabuľku polovičnej veľkosti, prvky starej tabuľky do nej presunieme a položku odoberieme

Cena jednej operácie TABLE INSERT alebo TABLE DELETE je v najhoršom prípade rovná počtu prvkov, ktoré presúvame do novej tabuľky.

Cena postupnosti n operácií typu TABLE INSERT a TABLE DELETE je preto v najhoršom prípade $\mathcal{O}(n^2)$.

Analyzujme amortizovanú zložitosť postupnosti n operácií TABLE INSERT.

Metóda zoskupení

- Skutočná cena i-tej operácie je i ak i je mocninou 2 a je 1 inak.
- Prvú skupinu tvoria operácie, ktorých skutočná cena je 1. Ich zložitosť je v súčte $< n$.
- Druhú skupinu tvoria operácie, pri ktorých sa alokuje nová tabuľka. Každá nová tabuľka má dvojnásobnú veľkosť. Súčet zložitostí týchto operácii je preto $\leq \sum_{j=0}^{\lfloor \log n \rfloor} 2^j$.

$$\begin{aligned}\sum_{i=1}^{n} c_i &\leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \\ &< n + 2n \\ &= 3n\end{aligned}$$

Metóda kreditov

Každej operácii priradíme 3 kredity.

- 1 kredit zaplatí vloženie položky do tabuľky
- 1 kredit dostane na svoj účet položka, ktorú sme práve vložili do tabuľky
- 1 kredit dáme na účet položke, ktorá je už v tabuľke a nemá na svojom účte žiaden kredit

Predpokladajme, že sme práve alokovali novú tabuľku veľkosti m a žiadna z $m/2$ položiek tabuľky nemá na svojom účte žiaden kredit. Než sa tabuľka znovu naplní, tak každá z týchto $m/2$ položiek, ako aj každá z $m/2$ nových položiek, bude mať na svojom účte 1 kredit. Tieto kredity sa použijú na presun položiek do novej tabuľky.

Metóda potenciálovej funkcie

Označme $num[T]$ počet položiek tabuľky a $size[T]$ veľkosť tabuľky T.
Definujme potenciálovú funkciu predpisom $\Phi(T) = 2 \cdot num[T] - size[T]$.

Ak i nie je mocninou 2, tak amortizovaná cena i-tej operácie je

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2num[T_i] - size[T_i]) - (2num[T_{i-1}] - size[T_{i-1}]) \\ &= 1 + 2 = 3\end{aligned}$$

Pre i rovné mocnine 2 je amortizovaná cena

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= num[T_i] + (2num[T_i] - size[T_i]) - (2num[T_{i-1}] - size[T_{i-1}]) \\ &= 3\end{aligned}$$

Dátové štruktúry

Haldy

Prehľad

- zásobník, fronta (PUSH, POP)
- spájané zoznamy jednosmerné a dvojsmerné (SEARCH, INSERT, DELETE)
- slovníky (SEARCH, INSERT, DELETE)
- vyhľadávacie stromy (+ MINIMUM, MAXIMUM, PREDECESSOR, SUCESSOR)
  - obecný vyhľadávací strom
  - vyvážené vyhľadávacie strom
    AVL, bielo-čierne stromy, B-stromy, 2-3 stromy
- haldy (INSERT, DELETE, MINIMUM)
  - *k*-árna halda (špeciálne binárna halda)
  - Fibonacciho halda
  - 2-3-4 halda
- reprezentácia disjunktných množín (INSERT, FIND, UNION)
  - spájané zoznamy
  - UNION-FIND štruktúra

Haldy

Halda je dátová štruktúra pre reprezentáciu množiny prvkov.
Predpokladáme, že nad prvkami množiny je definované úplné usporiadanie.

- Halda je les stromov, kde každý vrchol obsahuje *kľúč*. Kľúčom je prvok reprezentovanej množiny.
- Pre každý podstrom haldy platí, že kľúč koreňa podstromu je menší alebo rovný než kľúče všetkých jeho následníkov.

Podporované operácie

$\textsc{Make Heap}()$ vytvorí prázdnu haldu

$\textsc{Insert}(H, x)$ do haldy H vloží prvok x

$\textsc{Minimum}(H)$ nájde minimálny prvok v H

$\textsc{Extract Min}(H)$ z haldy H odstráni minimálny prvok

$\textsc{Delete}(H, x)$ z haldy H odstráni prvok x

$\textsc{Union}(H_1, H_2)$ vytvorí novú haldu zjednotením háld H_1, H_2

Haldy - prehľad zložitosti operácií

Operácia	Binárna halda	Binomiálna halda	Fibonacciho halda
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$
INSERT	$\Theta(\log n)$	$\Theta(1)$ *	$\Theta(1)$
UNION	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$ *
DELETE	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$ *
DECREASE-KEY	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$ *

* amortizovaná zložitosť

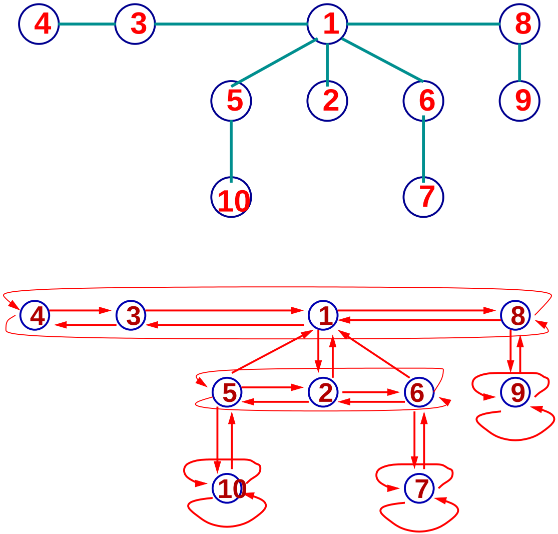
Fibonacciho halda

- modifikácia binomiálnej haldy
- efektívna realizáciu operácií Union, Insert a Decrease_Key
- nezhoršuje amortizovanú zložitosť ostatných operácií
- vhodné použitie: ak počet operácií Extract_Min a Delete je malý

Základné princípy

- ukazateľ na minimálny prvok
- štruktúra môže obsahovať viac stromov
- operácie, ktoré nie sú potrebné, odkladáme a vykonáme ich až keď je to nevyhnutné. Konkrétne, pri Union a Insert realizujeme jednoduché spojenie koreňových zoznamov (konštantná zložitosť). Stromy spájame až pri volaní operácie Decrease_Key.

Fibonacciho halda - schéma



Fibonacciho halda - schéma

Vrchol stromu je záznam s údajmi

p	ukazateľ na otca	*key*	kľúč
degree	počet synov	*mark*	binárny príznak
left	ukazateľ na ľavého brata	*right*	ukazateľ na pravého brata
child	ukazateľ na syna		

Halda je záznam s údajmi

min	ukazateľ na vrchol Fibonacciho haldy, ktorý obsahuje minimálny kľúč
n	aktuálny počet vrcholov v halde H

Potenciálová funkcia

Pre analýzu zložitosti operácií nad Fibonacciho haldou využijeme metódu potenciálovej funkcie.

Pre haldu H

- $t(H)$ je počet stromov v koreňovom zozname haldy H
- $m(H)$ je počet označených vrcholov (príznak $x.mark = true$)
- $\Phi(H) = t(H) + 2m(H)$ je potenciál haldy H

Amortizovaná analýza využíva horné ohraničenie $D(n)$ na maximálny stupeň vrchola vo Fibonacciho halde s n vrcholmi. Platí

$$D(n) = \mathcal{O}(\log n)$$

(tento fakt dokážeme na záver kapitolky o Fibonacciho haldách)

Vytvorenie prázdnej haldy

MAKE_FIB_HEAP()

- vytvorí a alokuje objekt H s parametrami $H.n = 0$, $H.min = Nil$
- potenciál prázdnej haldy je 0
- skutočná cena vytvorenia prázdnej haldy je $\mathcal{O}(1)$
- amortizovaná cena (tj. súčet skutočnej ceny a zmeny potenciálu) je $\mathcal{O}(1)$

Vloženie nového prvku do haldy

$\textsc{Fib\_Insert}(H, x)$

$x.degree \leftarrow 0$

$x.p \leftarrow Nil$

$x.child \leftarrow Nil$

$x.mark \leftarrow \textsc{false}$

if $min[H] = Nil$

then vytvor koreňový zoznam pre H obsahujúci jediný vrchol x

$H.min \leftarrow x$

else vlož x do koreňového zoznamu haldy H

if $x.key < H.min.key$ **then** $H.min \leftarrow x$ **fi fi**

$H.n \leftarrow H.n + 1$

potenciál pôvodnej haldy H je $\Phi(H) = t(H) + 2m(H)$

potenciál výslednej haldy H' je $\Phi(H') = t(H) + 1 + 2m(H)$

zmena potenciálu $\Phi(H) - \Phi(H') = 1$

skutočná cena vloženia prvku je $\mathcal{O}(1)$

amortizovaná cena je $\mathcal{O}(1)$

Nájdenie minimálneho prvku

FIB_MINIMUM(H)

využijeme ukazateľ $H.min$ na koreň obsahujúci minimálny prvok

skutočná cena = amortizovaná cena $= 1$

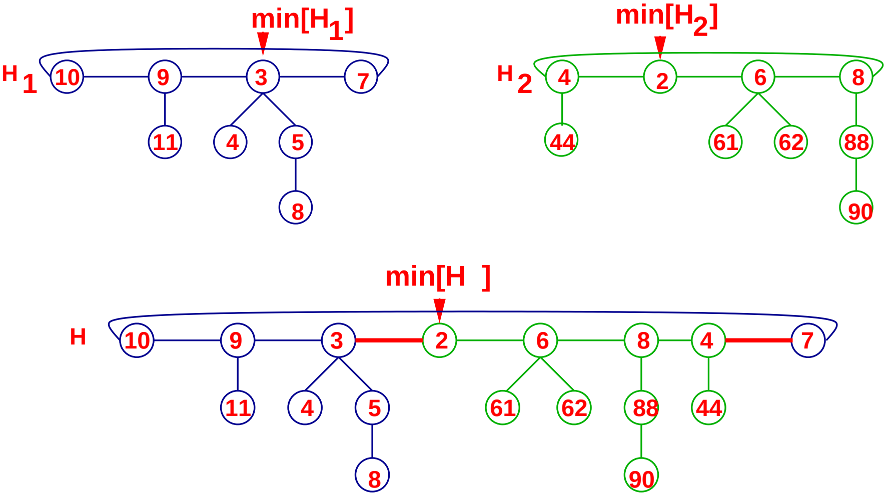
Zjednotenie dvoch háld

- koreňové zoznamy háld H_1 a H_2 spojíme do jedného zoznamu
- aktualizujeme hodnoty $H.min$ a $H.n$

$\textsc{Fib\_Union}(H_1, H_2)$
$H \leftarrow \textsc{Make\_Fib\_Heap}()$
spoj zoznamy H_1 a H_2 do zoznamu H
if $(H_1.min = \text{NIL}) \vee (H_2.min \neq \text{NIL} \wedge H_2.min < H_1.min)$
  then $H.min \leftarrow H_2.min$
  else $H.min \leftarrow H_1.min$ **fi**
$H.n \leftarrow H_1.n + H_2.n$
return H

zmena potenciálu je $\Phi(H) - (\Phi(H_1) + \Phi(H_2)) =$
$(t(H) + 2m(H)) - ((t(H_1) + 2m(H_1)) + (t(H_2) + 2m(H_2)) = 0$
skutočná cena je $\mathcal{O}(1)$
amortizovaná cena je $\mathcal{O}(1)$

Zjednotenie dvoch háld



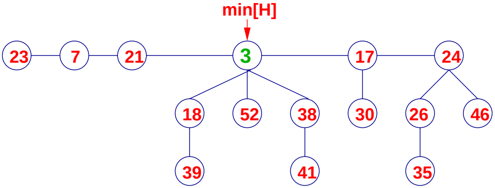
Odstránenie minimálneho prvku z haldy

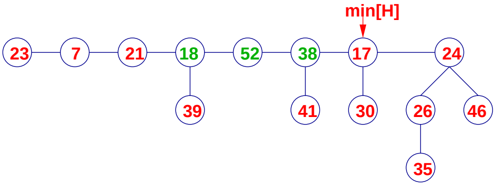
- všetkých synov koreňa z obsahujúceho minimálny prvok pripojíme do koreňového zoznamu H
- odstránime z
- voláme Consolidate a upravíme haldu tak, aby neobsahovala dva stromy rovnakého stupňa
- v upravenej halde prechádzame koreňový zoznam a hľadáme nový minimálny prvok

Odstránenie minimálneho prvku z haldy

```
FIB_EXCTRACT_MIN(H)
z ← H.min
if z ≠ NIL
   then for pre každého syna x vrcholu z do
              pridaj x do koreňového zoznamu haldy H
              x.p ← NIL od
           odstráň z zo zoznamu H
           if z = z.right
              then H.min ← NIL    (z bol jediný vrchol haldy)
               else H.min ← z.right
                    CONSOLIDATE(H) fi
           H.n ← H.n − 1
fi
return z
```

Odstránenie minimálneho prvku z haldy





Odstránenie minimálneho prvku z haldy – CONSOLIDATE

Procedúra CONSOLIDATE nájde minimálny prvok novej haldy a zároveň redukuje počet stromov v halde.
Opakovane realizuje nasledovné kroky.

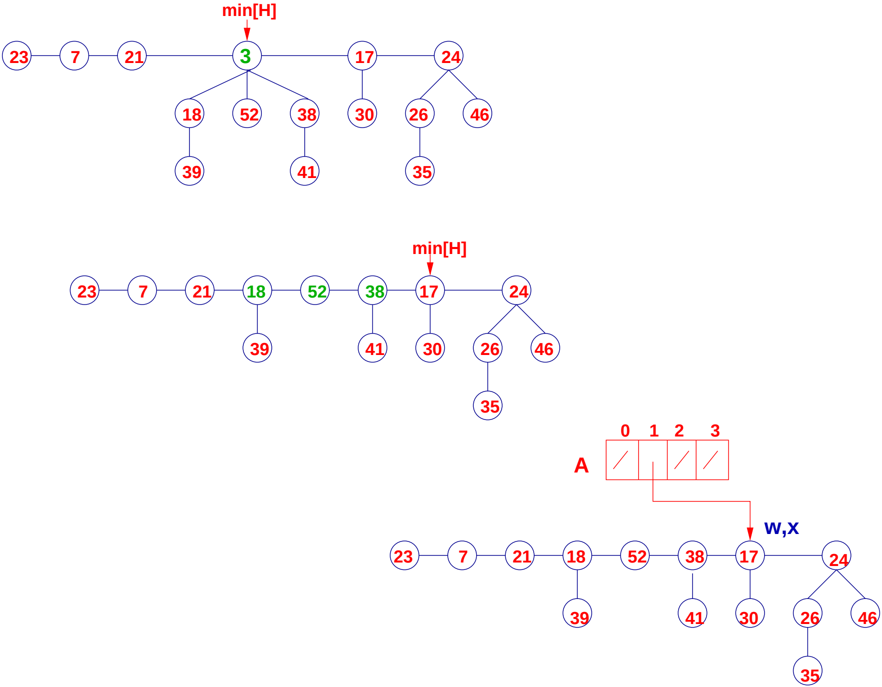
- Nájde v koreňovom zozname dva vrcholy x a y s rovnakým stupňom, *búno* $x.key \leq y.key$.
- Pripojí y k x tak, že odstráni vrhcol y z koreňového zoznamu a urobí vrchol y synom vrchola x.
  Pripojenie realizuje operácia FIB_LINK(H, y, x), ktorá zároveň zvýši atribút $x.degree$ a nastaví $y.mark$ na *false*.

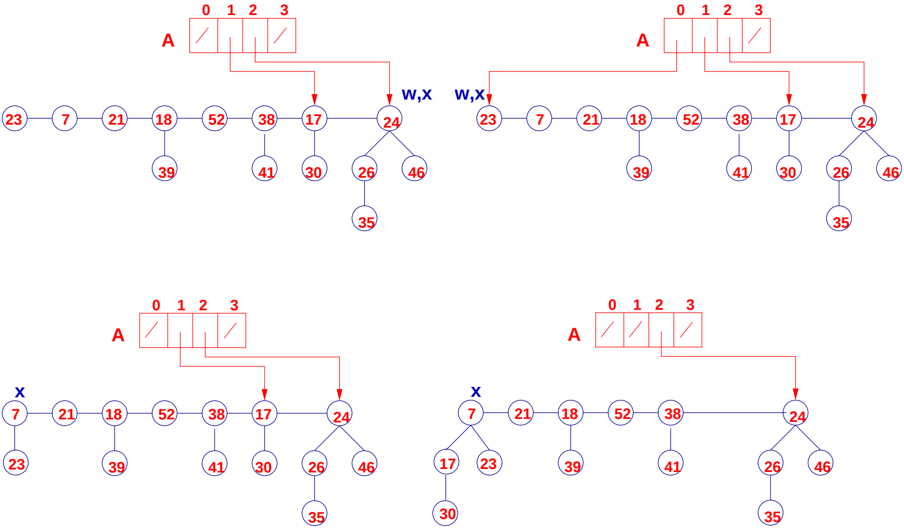
Procedúra CONSOLIDATE využíva pomocné pole $A[0 \ldots D(H.n)]$, v ktorom si uchováva informáciu o koreňoch podľa ich stupňa.
Ak $A[i] = y$, tak y je v aktuálnej halde koreňom s $y.degree = i$.

```
CONSOLIDATE(H)
for i ← 0 to D(H.n) do A[i] ← NIL od
for pre každý vrchol w v koreňovom zozname haldy H do
    x ← w
    d ← x.degree
    while A[d] ≠ NIL do
        y ← A[d]
        if x.key > y.key then exchange x ↔ y fi
        FIB_LINK(H, y, x)
        A[d] ← NIL
        d ← d + 1 od
    A[d] ← x od
H.min ← NIL
for i ← 0 to D(H.n) do
    if A[i] ≠ NIL then pridaj A[i] do koreňového zoznamu haldy H
                          if H.min = NIL ∨ A[i].key < H.min.key
                             then H.min ← A[i] fi fi
od
```

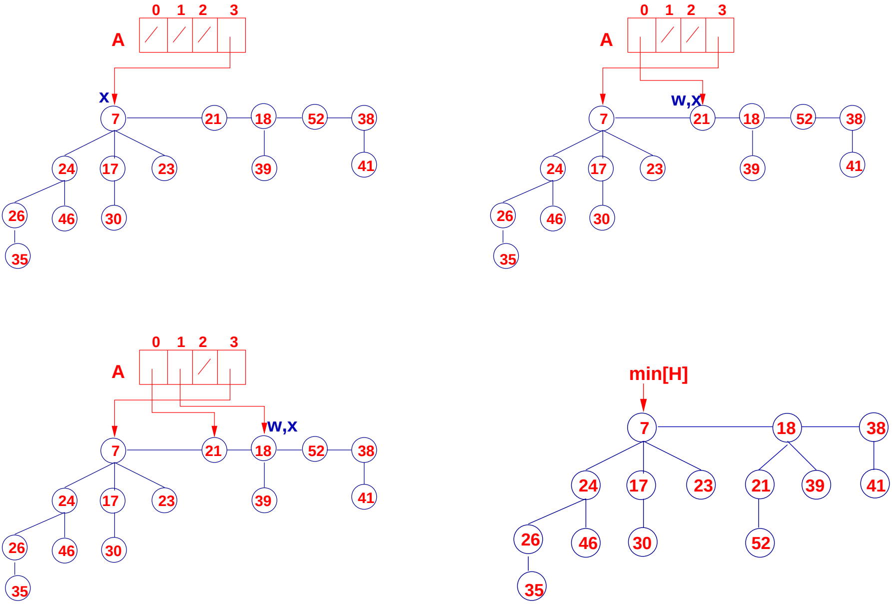
CONSOLIDATE - pokračovanie

FIB_LINK(H, y, x)
odstráň y z koreňového zoznamu haldy H
priraď vrchol y ako syna vrcholu x
$x.degree \leftarrow x.degree + 1$
$y.mark \leftarrow$ FALSE


min[H]
23 7 21 3 17 24
18 52 38 30 26 46
39 41 35
min[H]
23 7 21 18 52 38 17 24
39 41 30 26 46
35
0 1 2 3
A
w,x
23 7 21 18 52 38 17 24
39 41 30 26 46
35



A
0 1 2 3
w,x
23 7 21 18 52 38 17 24
39 41 30 26 46
35
A
0 1 2 3
w,x
23 7 21 18 52 38 17 24
39 41 30 26 46
35
A
0 1 2 3
x
7 21 18 52 38 17 24
23 39 41 30 26 46
35
A
0 1 2 3
x
7 21 18 52 38 24
17 23 39 41 26 46
30 35


A
0 1 2 3
x
7
21
18
52
38
24
17
23
39
41
26
46
30
35
A
0 1 2 3
w,x
7
21
18
52
38
24
17
23
39
41
26
46
30
35
A
0 1 2 3
w,x
7
21
18
52
38
24
17
23
39
41
26
46
30
35
min[H]
7
18
38
24
17
23
21
39
41
26
46
30
52
35

Odstránenie minimálneho prvku z haldy – zložitosť

- H označuje Fibonacciho haldu tesne pred aplikáciou operácie FIB_EXCTRACT_MIN
- cyklus 4 - 6 procedúry FIB_EXCTRACT_MIN má zložitosť $\mathcal{O}(D(n))$
- cykly na riadkoch 2 a 14 - 18 procedúry CONSOLIDATE majú zložitosť $\mathcal{O}(D(n))$
- pre cyklus 3 - 12 procedúry CONSOLIDATE platí
  - na začiatku volania procedúry CONSOLIDATE je veľkosť koreňového zoznamu nanajvýš $D(n) + t(H) - 1$
  - pri každej iterácii **while** cyklu sa odstráni jeden strom z koreňového zoznamu, tj. celkový počet iterácií **while** cyklu je zhora ohraničený hodnotou $D(n) + t(H)$
- celková skutočná zložitosť odstránenia prvku je $\mathcal{O}(D(n) + t(H))$

Zložitosť - pokračovanie

- potenciál haldy pred odstránením prvku je $\Phi(H) = t(H) + 2m(H)$
- potenciál po odstránení prvku je nanajvýš $(D(n) + 1) + 2m(H)$, pretože výsledná halda má nanajvýš $D(n) + 1$ koreňov a príznak *mark* sa u žiadneho vrchola nezmenil
- amortizovaná zložitosť je nanajvýš

$$\mathcal{O}(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H))$$

$$= \mathcal{O}(D(n)) + \mathcal{O}(t(H)) - t(H)$$

$$= \mathcal{O}(D(n))$$

intuitívne - spojenie dvoch stromov do jedného sa zaplatí poklesom potenciálu spôsobeným znížením počtu stromov v halde

Zníženie hodnoty kľúča

- Ak sa znížením kľúča vo vrchole x poruší vlastnosť haldy, tak namiesto výmeny vrcholov (ako v binárnej halde) odrežeme celý podstrom s koreňom x a urobíme ho novým stromom haldy, tj. pripojíme ho do koreňového zoznamu ($\textsc{Cut}$).
- V dôsledku prerezávania sa môže stať, že strom vysokého stupňa má veľmi málo vrcholov a následne halda obsahuje veľmi veľa stromov. Preto pri opakovanom odrezávaní podstromov zároveň znižujeme stupeň stromu.
- Konkrétne: ak niektorému vrcholu odrežeme druhého syna (príznak *mark*), tak odrežeme aj tento vrchol od jeho otca a v prípade potreby postupujeme s odrezávaním smerom ku koreňu ($\textsc{Cascading\_Cut}$)

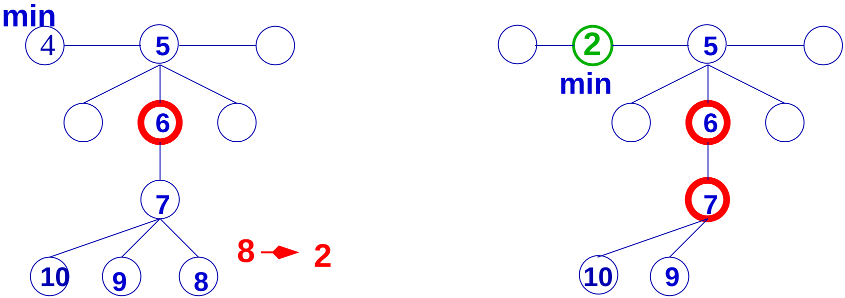
Zníženie hodnoty kľúča

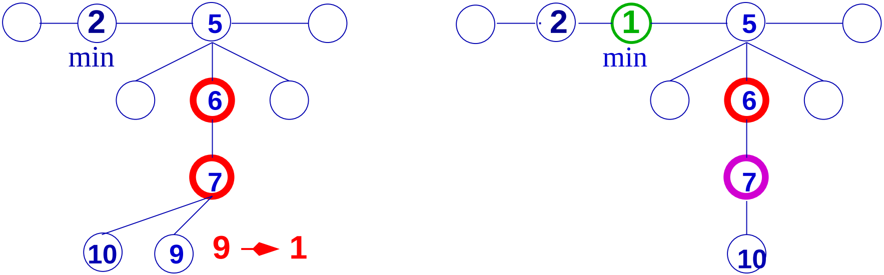
$\textsc{Fib\_Decrease\_Key}(H, x, k)$
if $k > x.key$ **then** chyba, nový kľúč je väčší než pôvodný **fi**
$x.key \leftarrow k$
$y \leftarrow x.p$
if $y \neq \textsc{nil} \wedge x.key < y.key$
  then $\textsc{Cut}(H, x, y)$
    $\textsc{Cascading\_Cut}(H, y)$ **fi**
if $x.key < H.min.key$
  then $H.min \leftarrow x$ **fi**

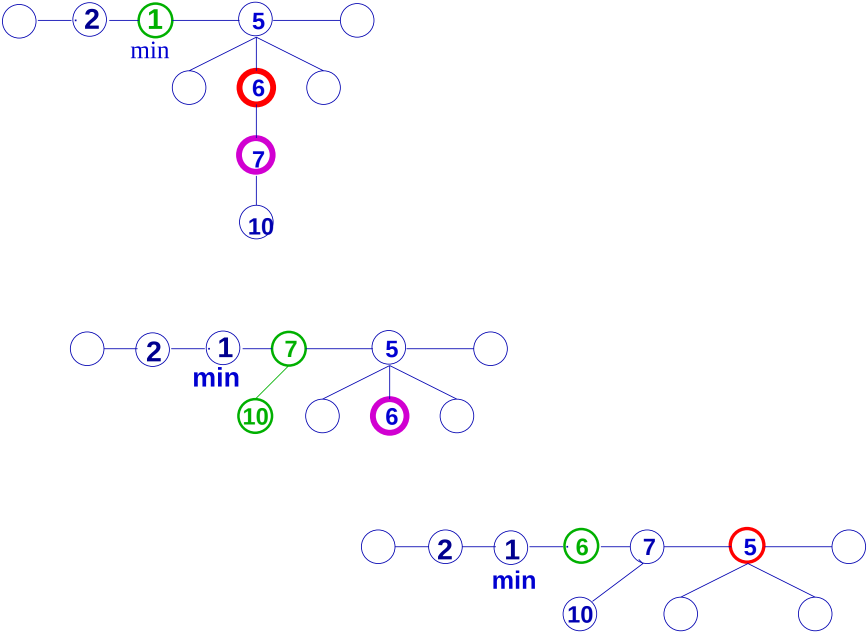
Znížnie hodnoty kľúča

$\textsc{Cut}(H, x, y)$
odstráň x zo zoznamu detí vrcholu y
$y.degree \leftarrow y.degree - 1$
pridaj x do koreňového zoznamu haldy H
$x.p \leftarrow \textsc{nil}$
$x.mark \leftarrow \textsc{false}$

$\textsc{Cascading\_Cut}(H, y)$
$z \leftarrow y.p$
if $z \neq \textsc{nil}$
  then if $y.mark = \textsc{false}$
    then $y.mark \leftarrow \textsc{true}$
    else $\textsc{Cut}(H, y, z)$
      $\textsc{Cascading\_Cut}(H, z)$
  fi
fi


min
4
5
6
7
10
9
8
8
2
2
min
5
6
7
10
9


2
5
min
6
7
10
9
9 → 1
2
1
5
min
6
7
10


2
1
min
5
6
7
10
2
1
min
7
5
10
6
2
1
min
6
7
5
10

Zníženie hodnoty kľúča – skutočná zložitosť

- skutočná cena operácie FIB_DECREASE_KEY je $\mathcal{O}(1)$ plus cena následného odrezávania stromov
- predpokladajme, že volanie procedúry FIB_DECREASE_KEY vyvolá c volaní procedúry CASCADING_CUT (1 volanie na riadku 7 a $c - 1$ rekurzívnych volaní).
- samotná procedúra CASCADING_CUT (bez rekurzívnych volaní) má zložitosť $\mathcal{O}(1)$
- skutočná cena operácie FIB_DECREASE_KEY je preto $\mathcal{O}(c)$

Zníženie hodnoty kľúča – zmena potenciálu

- H označuje Fibonacciho haldu tesne pred volanímprocedúry
- po zavolaní CUT sa vytvorí nový strom s koreňom x a jeho príznak *mark* sa nastaví na *false* [1]
- každé volanie CASCADING_CUT, s výnimkou posledného, odreže vrchol s príznakom $mark = true$ a zmení príznak vrcholu na *false*
- po ukončení procedúry FIB_DECREASE_KEY obsahuje halda $t(H) + c$ stromov a nanajvýš $m(H) - c + 2$ vrcholov s príznakom $mark = true$[2]
- zmena potenciálu je

$$((t(H) + c) + 2(m(H) - c + 2)) - (t(H) + 2m(H)) = 4 - c$$

[1] nemusí to nutne znamenať zmenu hodnoty

[2] $c - 1$ vrcholov sa odznačí v CASCADING_CUT a jeden sa označí pri poslednom volaní CASCADING_CUT

Zníženie hodnoty kľúča – amortizovaná zložitosť

amortizovaná zložitosť operácie Fib_Decrease_Key je

$$\mathcal{O}(c) + 4 - c = \mathcal{O}(1)$$

- potenciál haldy obsahuje člen, ktorý je dvojnásobkom počtu vrcholov s príznakom
- pri odrezaní vrcholu y s príznakom a zmene jeho príznaku na *false* sa zníži potenciál o 2
- zníženie potenciálu *zaplatí* jednak samotnú operáciu odrezania, jednak zvýšenie počtu stromov v halde

Odstránenie vrchola

symbol $-\infty$ používame pre označenie hodnoty nižšej než je kĺúč ktoréhokoľvek vrchola haldy

$\textsc{Fib\_Delete}(H, x)$
$\textsc{Fib\_Decrease\_Key}(H, x, -\infty)$
$\textsc{Fib\_Extract\_Min}(H)$

amortizovaná zložitosť operácie je súčtom amortizovaných cien operácií $\textsc{Fib\_Decrease\_Key}$ a $\textsc{Fib\_Extract\_Min}$, tj. $\mathcal{O}(D(n))$

Maximálny stupeň vrcholu

pre vyjadrenie amortizovanej zložitosti operácií FIB_EXTRACT_MIN a FIB_DELETE potrebujeme horné ohraničenie $D(n)$ na stupeň vrchola Fibonacciho haldy s n vrcholmi

Lemma 5

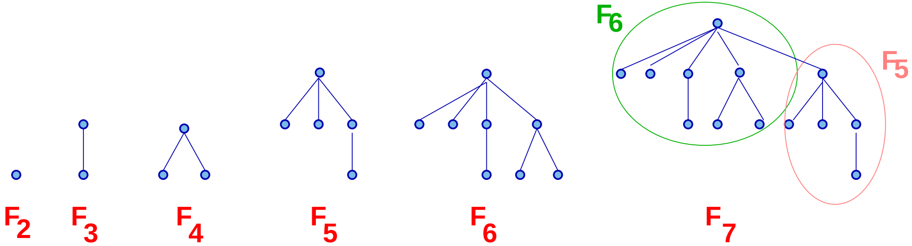
Nech x je ľubovoľný vrchol Fibonacciho haldy, nech $x.degree = k$. Označme $y_1, y_2, \ldots, y_k$ synov vrchola x a to v tom poradí, v akom boli pripojení k vrcholu x (y_1 ako prvý, y_k ako posledný). Potom $y_1.degree \geq 0$ a $y_i.degree \geq i - 2$ pre $i = 2, 3, \ldots, k$.

Dôkaz

- Nerovnosť $y_1.degree \geq 0$ je zrejmá.
- Uvážme vrchol y_i. Vo chvíli, keď sa stal synom x, tak x mal aspoň synov $y_1, \ldots, y_{i-1}$. Pri spájaní stromov vždy spájame stromy rovnakého stupňa, preto vo chvíli keď sa y_i stal synom x, mal aj y_i aspoň $i - 1$ synov. Odvtedy sme mu maximálne 1 syna odrezali. Preto má y_i stupeň aspoň $i - 2$. □

Maximálny stupeň vrcholu

Ukázali sme, že vo Fibonacciho halde má i-ty syn každého vrchola aspoň $i-2$ synov. Aký je minimálny počet vrcholov v strome, ktorý má uvedenú vlastnosť a jeho stupeň je i?



Fibonacciho postupnosť
$$F_k = \begin{cases} 0 & \text{pre } k = 0 \\ 1 & \text{pre } k = 1 \\ F_{k-1} + F_{k-2} & \text{pre } k \geq 2 \end{cases}$$

Maximálny stupeň vrchola

Lemma 6

Nech x je ľubovoľný vrchol Fibonacciho haldy, nech $k = x.degree$. Potom počet vrcholov v strome s koreňom x je aspoň $F_{k+2} \geq \Phi^k$, kde $\Phi = (1 + \sqrt{5})/2$.

Dôsledok 7

Maximálny stupeň $D(n)$ vrchola vo Fibonacciho halde s n vrcholmi je $\mathcal{O}(\log(n))$.

Reprezentácia disjunktných množín

Reprezentácia disjunktných množín

dátové štruktúry pre reprezentáciu disjunktných množín,
z ktorých každá má svojho jednoznačne určeného reprezentanta

podporované operácie

$\textsc{Make\_Set}(x)$ vytvorí množinu obsahujúcu prvok x

$\textsc{Union}(H_1, H_2)$ vytvorí novú množinu zjednotením množín H_1 a H_2

$\textsc{Find\_Set}(x)$ nájde reprezentanta množiny obsahujúcej prvok x

operácia $\textsc{Find\_Set}$ *dovoľuje efektívne testovať, či dva prvky patria do tej istej množiny*

implementácia pomocou

- reverzných stromov (*reversed trees*)
- plytkých stromov (spájaných zoznamov) (*shallow threaded trees*)
- stromov s kompresiou (*trees with path compresion*)

Reverzné stromy

- každú množinu reprezentujeme ako strom
- jeden vrchol stromu zodpovedá jednému prvku množiny
- každý vrchol obsahuje ukazateľ na svojho rodiča
- koreň ukazuje sám na seba a je reprezentantom množiny

Reverzné stromy – implementácia

$\textsc{Make\_Set}(x)$
$x.parent \leftarrow x$

$\textsc{Find\_Set}(x)$
while $x \neq x.parent$ **do** $x \leftarrow x.parent$ **od**
return x

$\textsc{Union}(x, y)$
$\overline{x} \leftarrow \textsc{Find\_Set}(x)$
$\overline{y} \leftarrow \textsc{Find\_Set}(y)$
$\overline{y}.parent \leftarrow \overline{x}$

zložitosť

- $\textsc{Make\_Set}$ a $\textsc{Union}$ majú konštantnú zložitosť
- zložitosť $\textsc{Find\_Set}$ závisí od hĺbky stromu a môže byť až lineárna k počtu prvkov prehľadávanej množiny

Reverzné stromy – optimalizácia

- pri spájaní dvoch stromov sa koreň stromu s menšou hĺbkou stane synom koreňa stromu s väčšou hĺbkou
- s každým vrcholom asociujeme informáciu o hĺbke stromu, ktorého je daný vrchol koreňom

Reverzné stromy – optimalizovaná implementácia

$\textsc{Make\_Set}(x)$
$x.parent \leftarrow x$
$x.depth \leftarrow 0$

$\textsc{Find\_Set}(x)$
while $x \neq x.parent$ **do** $x \leftarrow x.parent$ **od**
return x

$\textsc{Union}(x, y)$
$\overline{x} \leftarrow \textsc{Find\_Set}(x)$
$\overline{y} \leftarrow \textsc{Find\_Set}(y)$
if $\overline{x}.depth > \overline{y}.depth$
then $\overline{y}.parent \leftarrow \overline{x}$
else $\overline{x}.parent \leftarrow \overline{y}$
if $\overline{x}.depth = \overline{y}.depth$ **then** $\overline{y}.depth \leftarrow \overline{y}.depth + 1$ **fi**
fi

Reverzné stromy – optimalizovaná zložitosť

Lemma 8

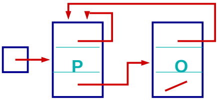
Pri reprezentácii množín reverznými stromami je zložitosť postupnosti obsahujúcej n operácií UNION, FIND_SET *a* MAKE_SET *je* $\mathcal{O}(n \log n)$.

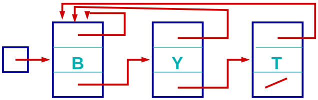
Dôkaz Indukciou k $d = x.depth$ overíme, že strom s koreňom x má aspoň $2^{x.depth}$ vrcholov. Pre $d = 0$ tvrdenie platí. Pre $d > 0$ uvažujme moment, keď x sa stane koreňom stromu hĺbky d. Situácia nastane pri spájaní dvoch stromov hĺbky $d - 1$, ktoré podľa IP majú aspoň 2^{d-1} vrcholov. Keďže každý strom má maximálne n vrcholov, jeho hĺbka nemôže byť väčšia než $\log n$.
MAKE_SET má konštantnú zložitosť
UNION a FIND_SET majú zložitosť úmernú hĺbke stromu, tj. $\mathcal{O}(\log n)$. $\square$

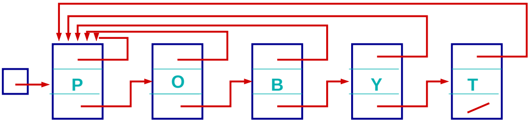
Plytké stromy = spájané zoznamy

- množinu reprezentujeme ako spájaný zoznam, prvý prvok zoznamu je reprezentantom množiny
- každý prvok zoznamu obsahuje ukazateľ na nasledujúci prvok zoznamu a na reprezentanta
- reprezentant obsahuje údaj o kardinalite množiny

alternatívne: množinu reprezentujeme stromom hĺbky 1, každý list ukazuje na koreň (=reprezentant množiny) a na svojho brata







Spájané zoznamy – operácie

MAKE_SET(x) strom s 1 vrcholom

UNION(H_1, H_2) • vyžaduje spojenie stromov a aktualizáciu ukazateľov na reprezentanta
• vždy pripájame množinu s menším počtom prvkov k množine s väčším počtom prvkov
• zložitosť je úmerná kardinalite množiny, ktorú pripájame

FIND_SET(x) konštantná zložitosť
(*použije sa ukazateľ na prvý prvok zoznamu*)

Spájané zoznamy – implementácia

$\textsc{Make\_Set}(x)$
$x.leader \leftarrow x$
$x.next \leftarrow x$
$x.size \leftarrow 1$

$\textsc{Find\_Set}(x)$
return $x.leader$

$\textsc{Weighted\_Union}(x, y)$
$\overline{x} \leftarrow \textsc{Find\_Set}(x)$
$\overline{y} \leftarrow \textsc{Find\_Set}(y)$
if $\overline{x}.size > \overline{y}.size$
  then $\textsc{Union}(\overline{x}, \overline{y})$
    $\overline{x}.size \leftarrow \overline{x}.size + \overline{y}.size$
  else $\textsc{Union}(\overline{y}, \overline{x})$
    $\overline{x}.size \leftarrow \overline{x}.size + \overline{y}.size$
fi

$\textsc{Union}(x, y)$
$\overline{x} \leftarrow \textsc{Find\_Set}(x)$
$\overline{y} \leftarrow \textsc{Find\_Set}(y)$
$y \leftarrow \overline{y}$
$y.leader \leftarrow \overline{x}$
while $y.next \neq Nil$
  do $y \leftarrow y.next$
  $y.leader \leftarrow \overline{x}$ **od**
$y.next \leftarrow \overline{x}.next$
$\overline{x}.next \leftarrow \overline{y}$

Spájané zoznamy – zložitosť

Lemma 9

Pri reprezentácii množín spájanými zoznamami je zložitosť postupnosti obsahujúcej n operácií WEIGHTED_UNION *a* FIND_SET *a m operácií* MAKE_SET *rovná $\mathcal{O}(m + n \log n)$.*

Dôkaz

Operácie MAKE_SET a FIND_SET majú konštantnú zložitosť.
Amortizovanú zložitosť WEIGHTED_UNION určíme metódou zoskupení.

Spájané zoznamy – zložitosť (pokračovanie).

Amortizovanú zložitosť WEIGHTED_UNION určíme metódou zoskupení tak, že určíme, koľko krát sa v priebehu výpočtu môže zmeniť hodnota $x.leader$ prvku x. Indukciou ľahko overíme, že ak hodnota $x.leader$ sa zmenila k krát, tak prvok x patrí do množiny, ktorá má aspoň 2^k prvkov. Po skončení postupnosti operácií má najväčšia množina kardinalitu nanajvýš n. Preto hodnota $x.leader$ sa zmení nanajvýš $\log n$ krát.

Každá operácia zjednotenia zníži počet množín o 1, po skončení postupnosti operácií máme $m - n$ množín a nanajvýš n prvkov patrí do viacprvkovej množiny. Každému prvku, ktorý patrí do viacprvkovej množiny, sme zmenili hodnotu $x.leader$ nanajvýš $\log n$ krát.

Amortizovaná zložitosť WEIGHTED_UNION je $\mathcal{O}(\log n)$, celková zložitosť operácií zjednotenia v uvažovanej postupnosti operácií je $\mathcal{O}(n \log n)$ □

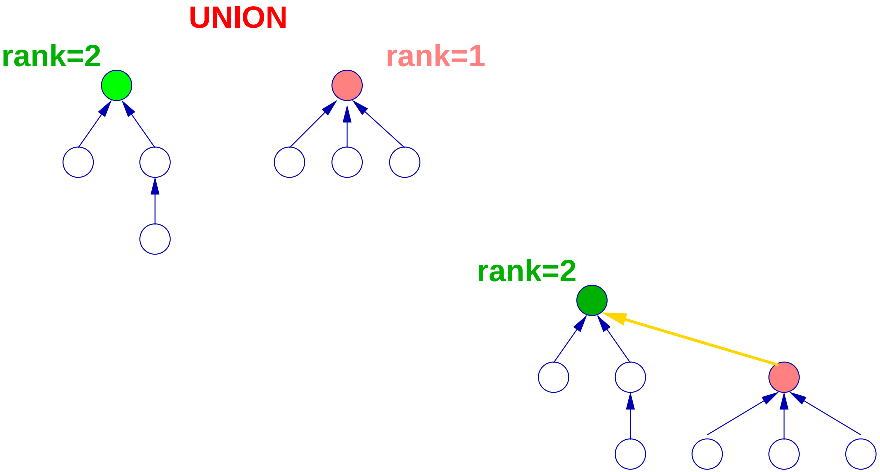
Strony s kompresiou

motivácia

- reverzné stromy majú logaritmickú zložitosť FIND_SET, ostatné operácie sú konštantné
- plytké stromy majú amortizovaná logaritmickú zložitosť UNION, ostatné operácie sú konštantné
- ideálne riešenie: všetky operácie konštantné ???

princíp

- každý vrchol x obsahuje údaj $x.rank$, ktorý zhora ohraničuje výšku podstromu s koreňom x
- rank stromu s koreňom r je $r.rank$
- UNIONje realizovaná ako spojenie dvoch stromov do jedného
- pri spájaní sa koreň stromu s menším rankom sa stane synom koreňa stromu s väčším rankom
- kompresia stromu

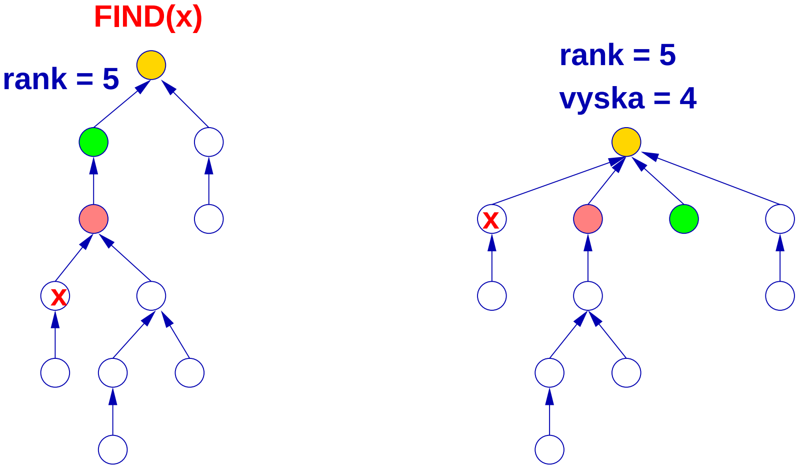

UNION
rank=2
rank=1
rank=2

Stromy s kompresiou – operácia FIND_SET

operácia FIND_SET

- sledujeme cestu od vrcholu obsahujúceho x do koreňa
- následne prechádzame cestu ešte raz a každý vrchol na ceste z x do koreňa stromu napojíme na koreň (*otcom každého vrchola na ceste sa stane koreň stromu*)

Stromy s kompresiou – operácia FIND_SET



Stromy s kompresiou – implementácia

$\textsc{Make\_Set}(x)$

$x.parent \leftarrow x$

$x.rank \leftarrow 0$

$\textsc{Union}(x, y)$

$\textsc{Link}(\textsc{Find\_Set}(x), \textsc{Find\_Set}(y))$

$\textsc{Link}(x, y)$

```
if x.rank > y.rank then y.parent ← x
                    else x.parent ← y
                         if x.rank = y.rank
                           then y.rank ← y.rank + 1 fi
fi
```

$\textsc{Find\_Set}(x)$

```
if x ≠ x.parent then x.parent ← FIND_SET(x.parent) fi
return x.parent
```

Lemma 10

Pri reprezentácii množín stromami s kompresiou je zložitosť postupnosti obsahujúcej m operácií UNION, FIND_SET *a* MAKE_SET, *z toho n operácií je* MAKE_SET, *rovná $\mathcal{O}(m \cdot \alpha(n))$.*

Definícia funkcie $\alpha(n)$

Pre prirodzené čísla $k \geq 0$ a $j \geq 1$ definujeme funkciu $A_k(j)$ predpisom

$$A_k(j) \stackrel{\text{def}}{=} \begin{cases} j+1 & \text{pre } k = 0 \\ \underbrace{A_{k-1}(A_{k-1}(\ldots A_{k-1}}_{j+1 \text{ krát}}(j))) & \text{pre } k \geq 1 \end{cases}$$

Funkcia $\alpha(n)$ je inverzná k funkcii $A_k(n)$

$$\alpha(n) \stackrel{\text{def}}{=} \min\{k \mid A_k(1) \geq n\}$$

Stromy s kompresiou – zložitosť

$A_1(j) = A_0(A_0(\ldots A_0(j)..)) = 2j + 1$
$A_2(j) = A_1(A_1(\ldots A_1(j)..)) = 2^{j+1}(j+1) - 1$
$A_3(1) = 2047$
$A_4(1) = 16^{512} \gg 10^{80}$

$$\alpha(n) = \begin{cases} 0 & \text{pre } 0 \leq n \leq 2 \\ 1 & \text{pre } n = 3 \\ 2 & \text{pre } 4 \leq n \leq 7 \\ 3 & \text{pre } 8 \leq n \leq 2047 \\ 4 & \text{pre } 2048 \leq n \leq A_4(1) \approx 10^{80} \end{cases}$$

Dátová štruktúra UNION-FIND a Kruskalov algoritmus

Úlohou je pre daný neorientovaný graf $G = (V, H)$ s ohodnotením hrán $w : H \to \mathbb{N}$ nájsť najlacnejšiu kostru. Algoritmus využíva dátovú štruktúru UNION-FIND na reprezentáciu disjunktných množín; každá množina reprezentuje jeden strom v aktuálnom lese. Na začiatku tvorí každý vrchol grafu izolovaný strom. Postupne uvažujeme hrany grafu v neklesajúcom poradí podľa ich ohodnotenia. Ak koncové vrcholy hrany (u, v) patria do rôznych stromov, tak hranu pridáme do kostry a príslušné stromy zjednotíme.

KRUSKAL$((V, H), w)$
for každý vrchol $v \in V$ **do** MAKE_SET(v) **od**
utrieď hrany z H do neklesajúcej postupnosti podľa w
for každú hranu (u, v) v danom poradí **do**
    if FIND_SET$(u) \neq$ FIND_SET(v)
      then $K \leftarrow K \cup \{(u, v)\}$; UNION$(u, v)$ **fi od**
return K

Metódy návrhu algoritmov

1. Zložitosť
2. Dátové štruktúry
3. Metódy návrhu algoritmov
   - Rozdeľ a panuj
   - Dynamické programovanie
   - Hladové algoritmy
   - Backtracking
4. Grafové algoritmy
5. Algoritmy pre prácu s reťazcami

Rozdeľ a panuj

Rozdeľ a panuj

1. problém rozdeľ na podproblémy
2. vyrieš podproblémy
3. z riešení podproblémov zostav riešenie problému

Maximálny a minimálny prvok

- problém nájdenia maximálneho a minimálneho prvku postupnosti $S[1 \dots n]$
- zložitostné kritérium - počet porovnaní prvkov

$\textsc{Max}(S)$
$max \leftarrow S[1]$
for $i = 2$ **to** n **do**
  if $S[i] > max$ **then** $max \leftarrow S[i]$ **fi**
od

minimum nájdeme medzi zvyšnými $n - 1$ prvkami podobne

celkove $(n-1) + (n-2)$ porovnaní

Maximálny a minimálny prvok – Prístup Rozdeľ a panuj

1. pole rozdeľ na dve (rovnako veľké) podpostunosti
2. nájdi minimum a maximum oboch podpostupností
3. maximálny prvok postupnosti je väčší z maximálnych prvkov oboch podpostupností
   podobne minimálny prvok

$\textsc{MaxMin}(x, y)$
nájdi minimálny a maximálny prvok v postupnosti $S[x \dots y]$
if $y = x$ **then return** $(S[x], S[x])$ **fi**
if $y = x + 1$ **then return** $(\max(S[x], S[y]), \min(S[x], S[y]))$ **fi**
if $y > x + 1$ **then** $(A1, B1) \leftarrow \textsc{MaxMin}(x, \lfloor (x+y)/2 \rfloor)$
$(A2, B2) \leftarrow \textsc{MaxMin}(\lfloor (x+y)/2 \rfloor + 1, y)$
return $(\max(A1, A2), \min(B1, B2))$ **fi**

Maximálny a minimálny prvok – analýza

Korektnosť indukciou vzhľadom k $n = y - x + 1$ ukážeme, že $\textsc{MaxMin}(x, y)$ vráti maximálnu a minimálnu hodnotu postupnosti

Zložitosť

$$T(n) = \begin{cases} 1 & \text{pre } n = 2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & \text{inak} \end{cases}$$

Indukciou k n overíme, že $T(n) \leq \frac{5}{3}n - 2$

1. pre $n = 2$ platí $\frac{5}{3} \cdot 2 - 2 > 1 = T(2)$
2. predpokladajme platnosť nerovnosti pre všetky hodnoty $2 \leq i < n$, dokážeme jej platnosť pre n

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 && \text{indukčný predp.} \\ &\leq \frac{5}{3}\lfloor n/2 \rfloor - 2 + \frac{5}{3}\lceil n/2 \rceil - 2 + 2 = \frac{5}{3}n - 2 \end{aligned}$$

Problém inverzií

motivácia

porovnanie zoznamu preferencií

formulácia problému

- je daná postupnosť vzájomne rôznych čísel $a_1, \ldots, a_n$
- inverziou v postupnosti je dvojica indexov i, j takých, že $i < j$ a zároveň $a_i > a_j$
- úlohou je nájsť všetky inverzie v danej postupnosti čísel

príklad

postupnosť 1, 4, 6, 8, 2, 5 má 5 inverzií

naivný algoritmus

otestuje všetky dvojice indexov, zložitosť $\mathcal{O}(n^2)$

Problém inverzií — prístup Rozdeľ a panuj

1. postupnosť rozdelíme na dve podspostupnosti $a_1, \ldots, a_{\lceil n/2 \rceil}$ a $a_{\lceil n/2 \rceil + 1}, \ldots, a_n$
2. v každej z podpostupností spočítame inverzie
3. spočítame inverzie medzi prvkami rôznych podpostupností

- ak chceme, aby časová zložitosť algoritmu bola $T(n) = \mathcal{O}(n \log n)$, tak musí platiť $T(n) \leq 2T(n/2) + cn$ *(pre vhodnú konštantu c)*
- ako vyriešiť úlohu 3. v čase cn ?
- pri riešení úlohy 2. súčasne s počítaním inverzií utriedime podpostupnosti
- úlohu 3. vyriešime spojením dvoch utriedených podpostupností do jednej utriedenej postupnosti, pričom zároveň počítame inverzie medzi prvkami podpostupností

Problém inverzií — riešenie úlohy 3.

- $B = b_1, b_2, \ldots, b_k$
- $C = c_1, c_2, \ldots, c_l$
- prepokladáme, že
  - prvky v oboch postupnostiach sú utriedené vzostupne
  - všetky prvky v postupnosti B majú vo vstupnej postupnosti menší index než prvky postupnosti C
- prvky $b_1, \ldots, b_{i-1}$ a $c_1, \ldots, c_{j-1}$ sú už zatriedené
- porovnávame prvok b_i s prvkom c_j

	menší z porovnávaných prvkov zaradíme do výstupnej postupnosti
$b_i < c_j$	b_i nie je v inverzii so žiadnym z prvkov $c_j, c_{j+1}, \ldots, c_l$
$c_j < b_i$	c_j je v inverzii so všetkými prvkami $b_i, b_{i+1}, \ldots, b_k$ a preto k počtu inverzií pripočítame $k - i + 1$

Rozdeľ a panuj – príklady

- algoritmus MergeSort *(triedenie spájaním)*
- nájdenie najbližšej dvojice bodov v rovine
- násobenie celých čísel
- násobenie matíc
- FFT (*Fast Fourier Transformation*)

Dynamické programovanie - princípy

1. charakterizuj štruktúru problému
   - *problém je možné rozložiť na podproblémy*
   - *optimálne riešenie problému v sebe obsahuje optimálne riešenia podproblémov*
2. rekurzívne definuj hodnotu optimálneho riešenia
3. vypočítaj hodnotu optimálneho riešenia *zdola-nahor*
4. z vypočítaných hodnôt zostav optimálne riešenie

technika je vhodná pre riešenie optimalizačných problémov,
u ktorých dochádza k prekrývaniu podproblémov

Rozvrh udalostí

- daných je n udalostí označených $1, \ldots, n$
- každá udalosť i je špecifikovaná svojim začiatkom s_i a koncom f_i ($s_i \leq f_i$) a hodnotou v_i
- dve udalosti sú kompatibilné, ak sa neprekrývajú (i, j sú kompatibilné ak $s_i \geq f_j$ alebo $s_j \geq f_i$)
- úlohou je nájsť takú množinu vzájomne kompatibilných úloh $S \subseteq \{1, \ldots, n\}$, pre ktorú hodnota $\sum_{i \in S} v_i$ je maximálna možná
- predpokladáme usporiadanie udalostí podľa koncového času, $f_1 \leq f_2 \leq \cdots \leq f_n$

Rozvrh udalostí – terminológia

- udalosť i je pred udalosťou j práve ak $i < j$
- pre udalosť j definujeme $p(j)$ ako najväčší index $i < j$ taký, že udalosti i a j sú disjuntné (kompatibilné)
  ak index požadovaných vlastností neexistuje, tak $p(j) = 0$
- príklad

Rozvrh udalostí – identifikácia podproblémov

nech $\mathcal{O}$ je optimálne riešenie

udalosť n buď patrí alebo nepatrí do optimálneho riešenia $\mathcal{O}$

n patrí do $\mathcal{O}$

- žiadna z udalostí $p(n) + 1, \ldots, n - 1$ nepatrí do $\mathcal{O}$
- $\mathcal{O}$ musí obsahovať optimálne riešenie problému pre udalosti $\{1, \ldots, p(n)\}$
  dôkaz sporom

n nepatrí do $\mathcal{O}$

- $\mathcal{O}$ je optimálnym riešením pre udalosti $\{1, \ldots, n - 1\}$

nájsť optimálne riešenie pre udalosti $\{1, \ldots, n\}$ znamená nájsť optimálne riešenia pre menšie problémy tvaru $\{1, \ldots, j\}$

Rozvrh udalostí – cena optimálneho riešenia

- označme pre $1 \leq j \leq n$ symbolom $\mathcal{O}_j$ optimálne riešenie pre udalosti $\{1, \ldots, j\}$; nech $OPT(j)$ je hodnota $\mathcal{O}_j$
- definujeme $\mathcal{O}_0 = \emptyset$ a $OPT(0) = 0$
- hľadané riešenie $\mathcal{O}$ je práve $\mathcal{O}_n$
- platí

$$OPT(j) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{ak } 0 < j \leq n \end{cases}$$

- udalosť j patrí do $\mathcal{O}_j$ práve vtedy, ak

$$v_j + OPT(p(j)) \geq OPT(j-1)$$

Výpočet optimálneho riešenia – Rozdeľ a panuj

$\textsc{ComputeOpt}(j)$
if $j = 0$
  then return (0)
  else return $(\max(v_j + \textsc{ComputeOpt}(p(j)), \textsc{ComputeOpt}(j-1)))$
fi

korektnosť indukciou k j

zložitosť
$T(n) = 2T(n-1) + c$
(exponenciálna!)

Výpočet hodnoty optimálneho riešenia – Memoizácia

- algoritmus $\textsc{ComputeOpt}$ rieši $n+1$ rôznych podproblémov $\textsc{ComputeOpt}(0), \ldots, \textsc{ComputeOpt}(n)$
- exponenciálna zložitosť je daná opakovaným riešením podproblémov
- výsledky vyriešených podproblémov si ukladáme do pomocnej tabuľky

$\textsc{M-ComputeOpt}(j)$
if $j = 0$
  then return (0)
   else if $M[j]$ definované **then return** $(M[j])$
      else $M[j] = \max(v_j + \textsc{M-ComputeOpt}(p(j)),$
        $\textsc{M-ComputeOpt}(j-1)))$
    fi fi

zložitosť čas $\mathcal{O}(n)$ + priestor pre tabuľku M
miera progresu: počet definovaných položiek tabuľky M
pri každom rekurzívnom volaní sa definuje 1 položka

Výpočet optimálneho riešenia – Memoizácia

- ak nás zaujíma nielen hodnota optimálneho riešenia $OPT(n)$, ale aj jeho štruktúra $\mathcal{O}_n$, musíme naviac počítať hodnoty $S[j] = \mathcal{O}_j$
- množina $S[j]$ môže mať až n prvkov a preto zápis $S[j]$ si môže vyžadovať dodatočný čas $\mathcal{O}(n)$
- efektívnejšia varianta: dopočítať $\mathcal{O}_n$ až po vypočítaní hodnoty $OPT(n)$ s využitím faktu, že udalosť j patrí do $\mathcal{O}_j$ práve vtedy, ak $v_j + OPT(p(j)) \geq OPT(j-1)$

$\textsc{FindSolution}(j)$

if $j = 0$ **then print** $-$
    else if $v_j + M[p(j)] \geq M[j-1]$
        then print $j, \textsc{FindSolution}(p(j))$
        else print $\textsc{FindSolution}(j-1)$
    fi fi

zložitosť $\mathcal{O}(n)$

Výpočet optimálneho riešenia - Dynamické programovanie

- využijeme rekurzívny vzťah medzi cenou riešení podproblémov

$$OPT(j) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{ak } 0 < j \leq n \end{cases}$$

- ceny počítame v poradí $OPT(0)$, $OPT(1)$, ..., $OPT(n)$

Iterative-ComputeOpt(j)
$M[0] = 0$
for $j = 1$ **to** n **do**
    $M[j] = \max\{v_j + M[p(j)], M[j-1]\}$
od

zložitosť $\mathcal{O}(n)$
korektnosť daná korektnosťou rekurzívneho vzťahu

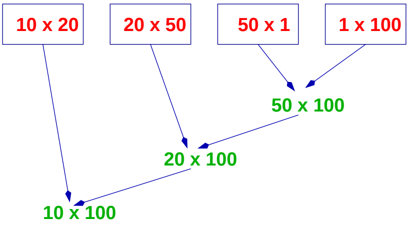
Dynamické programovanie - sumár

- počet rôznych podproblémov problému je polynomiálny
- riešenie pôvodného problému sa dá jednoducho vypočítať z riešení podproblémov
- existuje prirodzené usporiadanie podproblémov od *najmenších* po *najväčie*
- existuje jednoduchá rekurencia, ktorá umožňuje určiť riešenie podproblému pomocou riešenia *menších* podproblémov

Násobenie matíc

- je daná postupnosť matíc $\langle A_1, \ldots, A_n \rangle$
- úlohou je vypočítať ich súčin
- zložitostné kritérium je počet násobení čísel
- zložitosť výpočtu závisí od poradia, v akom násobíme matice
- úlohou je navrhnúť algoritmus, ktorý určí, v akom poradí sa majú matice násobiť tak, aby celková zložitosť výpočtu bola minimálna
- poradie násobenia určuje ozátvorkovanie postupnosti

Násobenie matíc – príklad



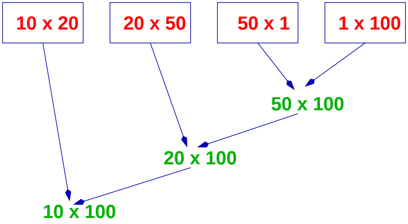
celkova cena = 125 000

cena = 5 000

cena = 100 000

cena = 20 000

Násobenie matíc – príklad

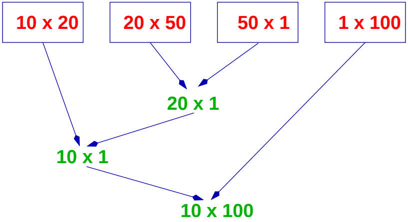


celkova cena = 125 000

cena = 5 000

cena = 100 000

cena = 20 000



celkova cena = 2 200

cena = 1 000

cena = 200

cena = 1 000

Násobenie matíc – triviálny prístup

vyskúšame všetky možnosti ozátvorkovania a pre každdú z nich určíme počet potrebných násobení

počet rôznych spôsobov, ktorými môžeme uzátvorkovať postupnosť n matíc je

$$P(n) = \begin{cases} 1 & \text{pre } n = 1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k) & \text{pre } n > 1 \end{cases}$$

indukciou k n overíme, že

$$P(n) \geq 2^{n-2}$$

Násobenie matíc – identifikácia podproblémov

- úlohou je nájsť optimálne ozátvorkovanie postupnosti matíc $\langle A_1 \dots A_n \rangle$
- pre indexy $i \leq j$ označme symbolom $A_{i\dots j}$ súčin $A_i \cdots A_j$
- pre výpočet $A_{i\dots j}$ musíme najprv vypočítať pre nejaký index k súčiny $A_{i\dots k}$ a $A_{k+1\dots j}$ a nakoniec vynásobiť matice $A_{i\dots k}$ a $A_{k+1\dots j}$
- podproblémy problému ozátvorkovania $\langle A_i \dots A_j \rangle$ sú problémy ozátvorkovania postupností $\langle A_i \dots A_k \rangle$ a $\langle A_{k+1} \dots A_j \rangle$
- nech optimálne ozátvorkovanie rozdelí $\langle A_i \dots A_j \rangle$ medzi A_k a A_{k+1}; ľahko overíme (sporom), že aj ozátvorkovanie $\langle A_i \dots A_k \rangle$ a $\langle A_{k+1} \dots A_j \rangle$ musí byť optimálne
- otázka vhodného indexu k: riešime podproblémy pre všetky indexy $k = i \dots j$
- ak poznáme optimálne riešenie všetkých podproblémov, tak vieme určiť optimálne riešenie celého problému

Násobenie matíc – cena optimálneho riešenia

- matica A_i má rozmery $p_{i-1} \times p_i$
- definujme funkciu $m : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ takú, že $m[i,j]$ označuje minimálny počet násobení čísel potrebný na výpočet $A_{i..j}$
- platí

$$m[i,j] \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } i = j \\ \min_{i \le k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{ak } i < j \end{cases}$$

- pretože $m[i,j]$ určuje len cenu optimálneho riešenia ale nie optimálny spôsob ozátvorkovania, definujeme $s[i,j]$ ako tú hodnotu k, pre ktorú sa realizuje minimum
- problém optimálneho ozátvorkovania sa dá vyjadriť ako problém výpočtu funkcií m a s

Výpočet ceny optimálneho riešenia – Rozdeľ a panuj

function $m[i,j]$
if $i = j$ **then return** (0)
else return $(\min_{i \leq k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\})$
fi

zložitosť

nech $T(n)$ označuje je zložitosť výpočtu hodnoty $m[i,j]$ pre $j - i + 1 = n$
pre $n > 0$ a vhodnú konštantu d platí

$$T(n) = \sum_{k=1}^{n-1}(T(k) + T(n-k)) + dn = 2\sum_{k=1}^{n-1} T(k) + dn$$

využijeme vtah $T(n) - T(n-1) = 2T(n-1) + d$ a dokážeme, že

$$T(n) = \Theta(3^n)$$

Výpočet ceny optim. riešenia – dynamické programovanie

- vychádzame z rekurencie

$$m[i,j] \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } i = j \\ \min_{i \leq k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & \text{ak } i < j \end{cases}$$

- usporiadanie je dané dĺžkou postupnosti matíc
- hodnoty počítame v poradí
  $m[1,1], m[2,2], \ldots, m[n,n]$
  $m[1,2], m[2,3] \ldots, m[n-1,n]$
  $m[1,3], m[2,4] \ldots, m[n-2,n]$
  ......
  $m[1,n]$

Optimálne Násobenie Matíc$\langle A_1, \ldots, A_n \rangle$
for $i = 1$ **to** n **do** $m[i,i] \leftarrow 0$ **od**
for $l = 2$ **to** n **do**
    for $i = 1$ **to** $n - l + 1$ **do**
        $j \leftarrow i + l - 1$
        $m[i,j] \leftarrow \infty$
        for $k = i$ **to** $j - 1$ **do**
            $q \leftarrow m[i,k] + m[k+1,j] + p_{i-1}p_k p_j$
            if $q < m[i,j]$ **then** $m[i,j] \leftarrow q;\ s[i,j] \leftarrow k$ **fi**
        od
    od
od
return (m, s)

zložitosť: $T(n) = \mathcal{O}(n^3)$

Zostavenie optimálneho riešenia

```
POSTUP NÁSOBENIA(s, i, j)
if i = j then print A_i
         else print "("
              POSTUP NÁSOBENIA(s, i, s[i, j])
              POSTUP NÁSOBENIA(s, s[i, j] + 1, j)
              print ")"
fi
```

Najdlhšia spoločná podpostupnosť

- nech $X =< x_1, \ldots, x_m >$, $Y =< y_1, \ldots, y_n >$ a $Z =< z_1, \ldots, z_k >$ sú postupnosti symbolov
- Z je podpostupnosťou postupnosti X práve ak existuje rastúca postupnosť indexov $1 \leq i_1 < \ldots < i_k \leq m$ taká, že pre všetky $j = 1, \ldots, k$ platí $x_{i_j} = z_j$
  analogicky pre Y
- Z je spoločnou podpostupnosťou postupností X a Y práve ak Z je podpostupnosťou X a zároveň je podpostupnosťou Y
- problém najdlhšej spoločnej podpostupnosti je pre dané dve postupnosti symbolov X a Y nájsť ich najdlhšiu spoločnú podpostupnosť (NSP)

NSP – identifikácia podproblémov

pre postupnosť $X =< x_1, \ldots, x_m >$ definujeme jej i-ty prefix ako
$X_i \stackrel{\text{def}}{=} < x_1, \ldots, x_i >$ (pre $i = 0, \ldots, m$)

nech $X =< x_1, \ldots, x_m >$ a $Y =< y_1, \ldots, y_n >$ sú postupnosti symbolov a nech $Z =< z_1, \ldots, z_k >$ je ich najdlhšia spoločná podpostupnosť (NSP)

1. ak $x_m = y_n$ tak
   - $z_k = x_m = y_n$ a
   - Z_{k-1} je NSP postupností X_{m-1} a Y_{n-1}
2. ak $x_m \neq y_n$ a $z_k \neq x_m$ tak
   - Z je NSP postupností X_{m-1} a Y
3. ak $x_m \neq y_n$ a $z_k \neq y_n$ tak
   - Z je NSP postupností X a Y_{n-1}

NSP – odnota optimálneho riešenia

- definujme funkciu $c : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ takú, že $c[i,j]$ označuje dĺžku najdlhšej spoločnej podpostupnosti postupností X_i a Y_j

$$c[i,j] \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } i = 0 \text{ alebo } j = 0 \\ c[i-1, j-1] + 1 & \text{ak } i,j > 0 \text{ a } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{ak } i,j > 0 \text{ a } x_i \neq y_j \end{cases}$$

- problém NSP je redukovaný na problém výpočtu hodnoty $c[m,n]$
- pri výpočte nepoužívame rekurziu, ale postupujeme od hodnôt s menším indexom

  poradie výpočtu: $\begin{array}{l} c[1,1], \dots, c[1,n], \\ c[2,1], \dots, c[2,n], \\ \quad\vdots \qquad \vdots \\ c[m,1], \dots, c[m,n] \end{array}$

NSP – výpočet hodnoty optimálneho riešenia

```
NSP(X,Y)
for i = 1 to m do c[i,0] ← 0 od
for j = 0 to n do c[0,j] ← 0 od
for i = 1 to m do
    for j = 1 to n do
        if X_i = Y_j then c[i,j] ← c[i−1,j−1] + 1
                          b[i,j] ← ♣
                  else if c[i−1,j] ≥ c[i,j−1]
                       then c[i,j] ← c[i−1,j]
                            b[i,j] ← ♠
                       else c[i,j] ← c[i,j−1]
                            b[i,j] ← ♡
                  fi fi od od
return (c, b)
```

NSP – zostavenie optimálneho riešenia

```
PRINT_NSP(b, X, i, j)
if i = 0 ∨ j = 0 then return  fi
if b[i, j] = ♣ then PRINT_NSP(b, X, i − 1, j − 1)
                    print x_i
               else if b[i, j] = ♠
                      then PRINT_NSP(b, X, i − 1, j)
                       else PRINT_NSP(b, X, i, −1)
                    fi
fi
```

Batoh (a varianty)

- varianty rozvrhu udalostí
- každá udalosť má svoju dĺžku (trvanie)
  na rozdiel od varianty uvažovanej na strane 137 udalosti nemajú určený začiatok a koniec
- cieľom je vybrať udalosti tak, aby poslucháreň bola maximálne vyťažená

Súčet čísel (subset sum, SS)

- daných je n udalostí $\{1, \ldots, n\}$, každá udalosť i má svoju nezápornú váhu w_i
- dané je ohraničenie W
- úlohou je vybrať podmnožinu S množiny udalostí tak, aby platilo $\sum_{i \in S} w_i \leq W$ a pritom suma $\sum_{i \in S} w_i$ bola maximálna možná

Problém batohu (knapsack)

- daných je n udalostí $\{1, \ldots, n\}$, každá udalosť i má svoju nezápornú váhu w_i a hodnotu h_i
- dané je ohraničenie W
- úlohou je vybrať podmnožinu S množiny udalostí tak, aby platilo $\sum_{i \in S} w_i \leq W$ a pritom suma $\sum_{i \in S} h_i$ bola maximálna možná

SS – metóda dynamického programovania

- podobne ako pre problém rozvrhu udalostí (strana 137) identifikujeme ako podproblém úlohu nájsť optimálny rozvrh pre prvých i udalostí
- analogicky s riešením pre rozvrh udalostí označme $OPT(i)$ cenu najlepšieho riešenia pre udalosti $\{1, \ldots, i\}$
- pre identifikáciu podproblémov znovu použijeme udalosť n

SS – identifikácia podproblémov

nech $\mathcal{O}$ je optimálne riešenie

udalosť n buď patrí alebo nepatrí do optimálneho riešenia $\mathcal{O}$

n nepatrí do $\mathcal{O}$

- $\mathcal{O}$ je optimálnym riešením pre udalosti $\{1, \ldots, n-1\}$
- $OPT(n) = OPT(n-1)$

n patrí do $\mathcal{O}$

- každá z udalostí $\{1, \ldots, n-1\}$ môže patriť do optimálneho riešenia
- udalosť n znížila ohraničenie W na hodnotu $W - w_n$
- potrebujeme najlepšie riešenie pre udalosti $\{1, \ldots, n-1\}$, ktoré má zároveň váhu nanajvýš $W - w_n$

musíme uvažovať viac podproblémov

podproblémom je každá iniciálna množina udalostí $\{1, \ldots, i\}$ a každé ohraničenie w, kde $0 \leq w \leq W$

SS – identifikácia podproblémov

označme $OPT(i, w)$ hodnotu optimálneho riešenia pre udalosti $\{1, \ldots, i\}$, ktoré neprekročí ohraničenie w, tj.

$$OPT(i, w) = \max_{S} \sum_{j \in S} w_j$$

pričom maximum uvažujeme cez všetky podmnožiny $S \subseteq \{1, \ldots, i\}$ pre ktoré platí $\sum_{j \in S} w_j \leq w$

- cena optimálneho riešenia je $OPT(n, W)$
- ak $n \notin \mathcal{O}$ tak $OPT(n, W) = OPT(n-1, W)$
- ak $n \in \mathcal{O}$ tak $OPT(n, W) = w_n + OPT(n-1, W - w_n)$

SS – identifikácia podproblémov

funkciu OPT definujeme pre hodnoty $0 \le n$, $0 \le W$ predpisom

$$OPT(i,w) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } i = 0 \\ OPT(i-1,w) & \text{ak } i \ge 1 \text{ a } w < w_i \\ \max\{w_i + OPT(i-1, w-w_i), & \\ \qquad OPT(i-1,w)\} & \text{ak } i \ge 1 \text{ a } w \ge w_i \end{cases}$$

hodnoty funkcie OPT počítame v poradí

$OPT(0,0), OPT(0,1), \dots, OPT(0,W)$
$OPT(1,0), OPT(1,1), \dots, OPT(1,W)$
$\vdots$
$OPT(n,0), OPT(n,1), \dots, OPT(n,W)$

časová zložitosť výpočtu funkcie OPT je $\mathcal{O}(nW)$
optimálne riešenie sa z hodnôt funkcie OPT dá vypočítať v čase $\mathcal{O}(n)$

Batoh

$OPT(i, w)$ je hodnota optimálneho riešenia pre udalosti $\{1, \dots, i\}$, tj.

$$OPT(i, w) = \max_S \sum_{j \in S} v_j$$

pričom maximum uvažujeme cez všetky podmnožiny $S \subseteq \{1, \dots, i\}$ pre ktoré platí $\sum_{j \in S} w_j \leq w$

$$OPT(i, w) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } i = 0 \\ OPT(i-1, w) & \text{ak } i \geq 1 \text{ a } w < w_i \\ \max\{v_i + OPT(i-1, w - w_i), & \\ \qquad OPT(i-1, w)\} & \text{ak } i \geq 1 \text{ a } w \geq w_i \end{cases}$$

Ďalšie príklady algoritmov dynamického programovania

- Floydov algoritmus
- Warshallov algoritmus
- . . . .

Hladové algoritmy

Hladové algoritmy

1. technika je vhodná pre riešenie optimalizačných problémov, ktoré majú optimálnu subštruktúru (optimálne riešenie problému v sebe obsahuje optimálne riešenie podproblémov)
2. pre získanie optimálneho riešenia stačí poznať optimálne riešenie jedného z podproblémov. Tento podproblém vyberáme na základe kritéria lokálnej optimality, tj. bez znalosti jeho riešenia
3. vo výpočte postupujeme zhora nadol

technika nie je použiteľná pre všetky optimalizačné problémy

Problém mincí

- k dispozícii máme mince hodnoty $h_1, \dots, h_k$
- počet mincí každej hodnoty je neobmedzený
- úlohou je pre dané N zaplatiť sumu N za použitia čo najmenšieho počtu mincí
- problém má optimálnu subštruktúru, pretože ak použijeme mincu hodnoty h_i, tak na zaplatenie sumy $N - h_i$ musíme opäť použiť optimálny počet mincí
- označme $MIN[i]$ minimálny počet mincí, ktoré sú potrebné na zaplatenie sumy i

$$MIN[i] \stackrel{\text{def}}{=} \begin{cases} 0 & \text{pre } i = 0 \\ 1 + \min_{1 \le j \le k}\{MIN[i - h_j]\} & \text{pre } i > 0 \end{cases}$$

Problém mincí – výber stratégie

dynamický prístup počítame hodnoty $MIN[1], \ldots, MIN[N]$
zložitosť je $\mathcal{O}(N \cdot k)$, algoritmus vždy vypočíta optimálnu hodnotu

hladový prístup použijeme kritérium lokálnej optimality
ak máme zaplatiť hodnotu i, tak vyberieme mincu h_x maximálnej hodnoty takú, že $h_x \leq i$

Problém mincí – hladový algoritmus

MINCE(i)
predpokladáme, že $h_1 > h_2 > \ldots > h_k$
$s \leftarrow 0$
for $m = 1$ **to** k **do**
    while $i \geq h_m$ **do** $i \leftarrow i - h_m$
        $s \leftarrow s + 1$ **od**
od
return s

Pre mince hodnoty 6, 4, 1 a sumu 8 algoritmus nenájde optimálne riešenie!!

Problém pásky

- je daných n súborov dĺžky $m_1, m_2, \ldots, m_n$
- súbory sa majú uložiť na pásku; všetky súbory sa budú z pásky čítať
- prístupový čas k súboru i je rovný súčtu dĺžok súborov, ktoré sú na páske uložené pred ním, plus jeho dĺžka m_i
- úlohou je uložiť súbory na pásku v takom poradí, aby sa súčet prístupových časov k jednotlivým súborom minimalizoval
- problém má optimálnu subštruktúru, pretože ak ako prvý uložíme na pásku súbor i, tak ostatné súbory musíme usporiadať optimálne.

hladový algoritmus

kritérium lokálneho výberu: spomedzi ešte neuložených súborov vyber najkratší a ulož ho na pásku
(tj. ulož súbory na pásku od najkratšieho po najdlhší)

Problém pásky – korektnosť

Lemma 11

Každá permutácia $(i_1, \ldots, i_n)$ súborov taká, že postupnosť $m_{i_1}, \ldots, m_{i_n}$ je neklesajúca, má minimálny súčet prístupových časov.

Dôkaz Nech $\pi = (i_1, \ldots, i_n)$ je permutácia $(1, \ldots, n)$ taká, že postupnosť $m_{i_1}, \ldots, m_{i_n}$ nie je neklesajúca. Preto existuje $1 \leq j < n$ pre ktoré $m_{i_j} > m_{i_{j+1}}$. Nech π' je permutácia,ktorá vznikne z π prehodením i_j a i_{j+1}. Cena permutácií π a π' je

$$C(\pi) = \sum_{k=1}^{n} \sum_{j=1}^{k} m_{i_j} = \sum_{k=1}^{n} (n-k+1) m_{i_k}$$

$$
\begin{aligned}
C(\pi') &= \sum_{k=1}^{j-1}(n-k+1)m_{i_k} + (n-j+1)m_{i_{j+1}} + (n-j)m_{i_j} \\
&\quad + \sum_{k=j+2}^{n}(n-k+1)m_{i_k}
\end{aligned}
$$

Z toho

$$
\begin{aligned}
C(\pi) - C(\pi') &= (n-j+1)(m_{i_j} - m_{i_{j+1}}) + (n-j)(m_{i_{j+1}} - m_{i_j}) \\
&= m_{i_j} - m_{i_{j+1}} > 0
\end{aligned}
$$

Preto π nemá minimálny súčet prístupových časov. Naviac ľahko overíme, že všetky permutácie požadovaných vlastností majú rovnaký (a teda minimálny) súčet prístupových časov. $\square$

Rozvrh udalostí

- daných je n udalostí (prednášok) označených $1, \ldots, n$
- každá udalosť i je špecifikovaná svojim začiatkom $s(i)$ a koncom $f(i)$ ($s(i) \leq f(i)$)
- dve udalosti sú kompatibilné, ak sa neprekrývajú
- úlohou je nájsť čo najväčšiu množinu vzájomne kompatibilných udalostí (*prednášok, ktoré sa môžu konať v jednej poslucharni*)

Kritéria lokálneho výberu

- vyber udalosť s najnižším $s(i)$
- vyber udalosť s najnižším $f(i) - s(i)$
- vyber udalosť, ktorá má najmenej kolízií (tj. počet udalostí, ktoré sú s ňou kompatibilné, je maximálny)
- vyber udalosť s najnižším $f(i)$

Hladový algoritmus

HLADOVÝ ROZVRH

predpokladáme, že $f(1) \leq f(2) \leq \ldots \leq f(n)$

$A \leftarrow \{1\}$
$j \leftarrow 1$
for $i = 2$ **to** n **do**
    if $s(i) \geq f(j)$ **then** $A \leftarrow A \cup \{i\}$
                                    $j \leftarrow i$
    fi
od
return A

Zložitosť $\mathcal{O}(n \log n)$
(*ak započítavame aj čas potrebný pre usporiadanie udalostí*)

Hladový algoritmus – korektnosť

označenie

A riešenie nájdené algoritmom HLADOVÝ ROZVRH

$\mathcal{O}$ optimálne riešenie

potrebujeme dokázať

1. A je kompatibilná (*zrejmé z konštrukcie* A)
2. A je maximálna (*tj.* $|A| = |\mathcal{O}|$)

Hladový algoritmus – korektnosť

označenie

A riešenie nájdené algoritmom HLADOVÝ ROZVRH

$\mathcal{O}$ optimálne riešenie

potrebujeme dokázať

1. A je kompatibilná (*zrejmé z konštrukcie A*)
2. A je maximálna (*tj.* $|A| = |\mathcal{O}|$)

notácia

- $i_1, i_2, \ldots, i_k$ označujeme udalosti z množiny A v poradí, v akom sme ich počas výpočtu pridávali do A
- $\mathcal{O} = \{j_1, \ldots, j_m\}$ b.ú.n.o. platí $s(j_1) < \ldots < s(j_m)$

cieľ

- dokázať $k = m$

Lemma 12

Pre všetky $r \leq k$ platí $f(i_r) \leq f(j_r)$.

Dôkaz indukciou vzhľadom k hodnote r

1. ak $r = 1$, tak $f(i_1) \leq f(j_1)$ z definície algoritmu
2. indukčný predpoklad: $f(i_{r-1}) \leq f(j_{r-1})$

- platí $f(j_{r-1}) \leq s(j_r)$ (*pretože $\mathcal{O}$ je kompatibilná*)
- j_r patrí do množiny udalostí kompatibilných s $\{i_1, \ldots, i_{r-1}\}$
- algoritmus vyberie udalosť s minimálnym časom ukončenia
- $f(i_r) \leq f(j_r)$ $\square$

Theorem 13

Algoritmus HLADOVÝ ROZVRH *nájde optimálne riešenie problému udalostí.*

Dôkaz sporom

- nech A nie je optimálna, tj. $|A| < |\mathcal{O}|$, $k < m$
- uvážme $j_{k+1} \in \mathcal{O}$. Platí $s(j_{k+1}) \geq f(j_k) \geq f(i_k)$
  (prvá nerovnosť vyplýva z kompatibility $\mathcal{O}$, druhá z Lemmatu)
- dostávame spor s tým, že algoritmus ukončil svoj výpočet preto, že žiadna z udalostí nebola kompatibilná s i_k $\square$

Rozvrh udalostí - varianta s penalizáciami

- daných je n udalostí (udalostí) označených $1, \ldots, n$
- každá udalosť i je špecifikovaná svojim trvaním $t(i)$ a termínom $d(i)$, do ktorého musí byť realizovaná (*deadline*)
- prípustné riešenie priradí každej udalosti i interval $[s(i), f(i)]$ taký, že $t(i) = f(i) - s(i)$ a intervaly priradené úlohám sú kompatibilné
- latencia udalosti i v prípustnom riešení je rozdiel $f(i) - d(i)$; ak $f(i) \leq d(i)$, tak latencia je 0
- cena prípustného riešenia je maximum latencií udalostí
- cieľom je nájsť riešenie s minimálnou cenou

terminológia: o prípustnom riešení hovoríme ako o rozvrhu

Hladové kritéria

- usporiadame udalosti podľa $t(i)$ (od najkratšej k najdlhšej)
  protipríklad: $t(1) = 1, d(1) = 100, t(2) = 10, D(2) = 10$, *kritérium neberie v úvahu deadliny*
- usporiadame udalosti podľa $d(i) - t(i)$ (od najmenšej rezervy)
  protipríklad: $t(1) = 1, d(1) = 2, t(2) = 10, D(2) = 10$
- usporiadame udalosti podľa $d(i)$ (v rastúcom poradí)

Hladový algoritmus

HLADOVÝ P-ROZVRH

predpokladáme, že $d(1) \leq d(2) \leq \ldots \leq d(n)$

predpokladáme, že rozvrh začína v čase s

$f \leftarrow s$

for $i = 1$ **to** n **do**

udalosti i priraď interval $[f, f + t(i)]$

$f \leftarrow f + t(i)$

od

Zložitosť $\mathcal{O}(n \log n)$

(*ak započítavame aj čas potrebný pre usporiadanie udalostí*)

Analýza algoritmu Hladový P-Rozvrh

označenie

- A riešenie nájdené algoritmom Hladový P-Rozvrh
- $\mathcal{O}$ optimálne riešenie

pozorovanie
rozvrh A nemá prestávku (interval, v ktorom sa nekoná žiadna udalosť)

Lemma 14 (1)

Existuje optimálny rozvrh, ktorý nemá prestávku.

Dôkaz *zrejmý*

inverzia

rozvrh B má inverziu ak udalosť i predchádza v B udalosť j a $d(j) < d(i)$

pozorovanie

rozvrh A nemá inverziu

Lemma 15 (2)

Všetky rozvrhy bez inverzií a bez prestávok majú rovnakú latenciu.

Dôkaz

- dva rozvrhy bez prestávok a bez inverzií sa môžu líšiť len poradím udalostí s rovnakým deadlinom
- tieto udalosti sú usporiadané tesne za sebou
- maximálnu latenciu má posledná z nich a to bez ohľadu na ich vzájomné usporiadanie

Lemma 16

Existuje optimálny rozvrh bez inverzie a bez prestávky.

Dôkaz nech $\mathcal{O}$ je optimálny rozvrh; postupnou úpravou transformuje $\mathcal{O}$ na rozvrh $\overline{\mathcal{O}}$, ktorý má požadované vlastnosti a rovnakú latenciu ako $\mathcal{O}$

1. podľa Lemmy (1) môžeme predpokladať, že $\mathcal{O}$ nemá prestávku
2. ak $\mathcal{O}$ má inverziu, tak existuje dvojica udalostí i a j takých, že j nasleduje v rozvrhu bezprostredne za i a $d(j) < d(i)$
3. ak v rozvrhu vzájomne zameníme udalosti i a j, tak počet inverzií v rozvrhu sa zníži o 1
4. rozvrh so zamenenými udalosťmi i a j nemá väčšiu latenciu než pôvodný rozvrh

Theorem 17

Rozvrh A je optimálny.

Dôkaz

- existuje optimálny rozvrh bez inverzií
- všetky rozvrhy bez inverzií majú rovnakú latenciu
- A nemá inverziu

Príklady hladových algoritmov

- Dijkstrov algoritmus pre problém najkratších ciest z daného vrchola
- Kruskalov a Primov algoritmus pre najlacnejšie kostry
- Huffmanove kódy

Backtracking

Backtracking

heuristická metóda, používaná prevažne pre optimalizačné problémy

inteligentné prehľadávanie

- prehľadávane priestoru potenciálnych (prípustných) riešení
- počet potenciálnych (prípustných) riešení je vysoký (exponenciálny)
- nad priestorom potenciálnych riešení definujeme stromovú štruktúru
- stromovú štruktúru prehľadávame do hĺbky
- prehľadávanie zefektívnime odrezávaním ciest, ktoré dokázateľne nevedú k hľadanému riešeniu

Návrh backtrackovacieho algoritmu

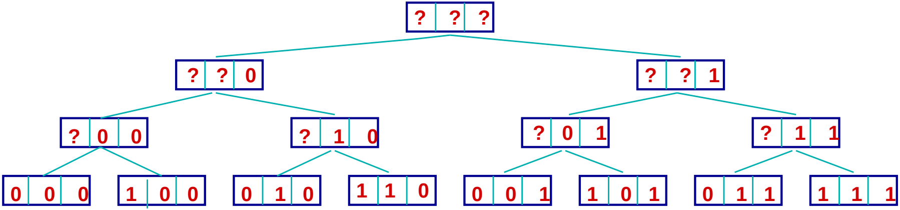
1. voľba spôsobu reprezentácie potenciálnych riešení s využitím kombinatorických štruktúr
2. generovanie potenciálnych riešení technikou Rozdeľ a panuj
3. testovanie požadovanej vlastnosti
4. odrezávanie neperspektívnych ciest

Problém výberu objektov

- daných je n objektov váhy $v_1, \ldots, v_n$ a číslo $V \in \mathbb{N}$
- úlohou je vybrať objekty tak, aby súčet ich váh bol presne V

1. potenciálne riešenia sú všetky spôsoby výberu objektov
2. riešenia budeme reprezentovať ako binárne reťacze, konkrétne pomocou binárneho poľa $A[1..n]$; hodnota $A[i] = 1$ reprezentuje informáciu, že objekt i bol zaradený do výberu
3. metódou rozdeľ a panuj generujeme všetky binárne reťazce dĺžky n
4. každý (čiastočn vygenerovaný) reťazec reprezentuje (čiastočný) výber objektov
5. ak vybrané objekty presiahnu váhový limit V, nepokračujeme v ďalšom generovaní binárneho reťazca

Problém výberu objektov - algoritmus



$\textsc{Vyber Objektov}(m, k)$

if $m = 0$ **then if** $k = 0$ **then** print A **fi**

else $A[m] \leftarrow 0$

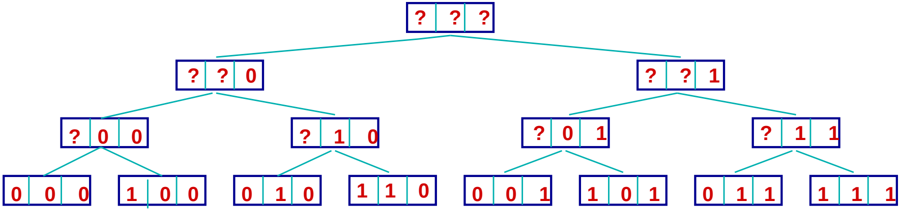
$\textsc{Vyber Objektov}(m - 1, k)$

if $v_m \leq k$ **then** $A[m] \leftarrow 1;$

$\textsc{Vyber Objektov}(m - 1, k - v_m)$ **fi**

fi

Problém výberu objektov - algoritmus



- strom prehľadávame do hĺbky
- pole A napĺňame zprava doľava
- cesty odrezávame (nepokračujeme v generovaní binárneho reťazca) v okamihu, keď váha vybraných objektov prekročí daný limit

Poznámka: technikou dynamického programovania sa problém dá riešiť v čase $\mathcal{O}(n \cdot V)$.

Hamiltonovský cyklus

Hamiltonovský cyklus v orientovanom grafe je cyklus, ktorý navštívi každý vrchol grafu práve raz

reprezentácia grafu polia $N[1..n, 1..s]$ a $S[1..n]$, kde n je počet vrcholov grafu a s je maximálny stupeň vrcholu v grafe. Hodnota $S[i]$ je stupeň vrcholu i a $N[i]$ je zoznam susedov vrcholu i.

potenciálne riešenia sú postupnosti vrcholov dĺžky n
riešenia reprezentujeme pomocou poľa $A[1..n]$, hodnoty uložené v A zodpovedajú cyklu
$A[n] \rightarrow A[n-1] \cdots \rightarrow A[1] \rightarrow A[n]$

neperspektívne riešenia postupnosti, v ktorých sa niektorý vrchol opakuje
pole $P[1..n]$; hodnota $P[i]$ je *false* ak aktuálna postupnosť už obsahuje vrchol i

Hamiltonovský cyklus – algoritmus

for $i = 1$ **to** $n-1$ **do** $P[i] \leftarrow \mathit{true}$ **od**
$P[n] \leftarrow \mathit{false}$; $A[n] \leftarrow n$
$\textsc{Hamilton}(n-1)$

procedure $\textsc{Hamilton}(m)$
if $m = 0$ **then** $\textsc{Kontrola}(A)$
    else for $j = 1$ **to** $S[A[m+1]]$ **do**
        $w \leftarrow N[A[m+1], j]$
        if $P[w]$ **then** $P[w] \leftarrow \mathit{false}$; $A[m] \leftarrow w$
            $\textsc{Hamilton}(m-1)$; $P[w] \leftarrow \mathit{true}$
        fi od fi

procedure $\textsc{Kontrola}(A)$
$ok \leftarrow \mathit{false}$
for $j = 1$ **to** $S[A[1]]$ **do if** $N[A[1], j] = A[n]$ **then** $ok \leftarrow \mathit{true}$ **fi od**
if ok **then** $\mathrm{print} A$

Hamiltonovský cyklus – algoritmus

- strom prehľadávame do hĺbky
- pole A napĺňame zprava
- po vygenerovaní Hamiltonovskej cesty dĺžky n testujeme (procedúra KONTROLA), či je možné cestu uzavrieť do cyklu
- neperspektívne riešenia odrezávame vždy, keď sa nich zopakuje niektorý vrchol druhý krát

Zložitosť $T(n)$ algoritmu je (b, c sú vhodné konštanty)

$$T(n) \leq \begin{cases} b \cdot s & \text{ak } n = 1 \\ s \cdot T(n-1) + c & \text{inak} \end{cases}$$

$$T(n) = \mathcal{O}(s^n)$$

Hamiltonovský cyklus - riešenie založené na permutáciach

- potenciálne riešenia sú permutácie postupnosti $(1, \ldots, n)$
- permutácie sú uložené v poli $A[1..n]$, iniciálne $A[i] = i$ pre $i = 1, \ldots, n$
- graf je zadaný maticou susednosti $M[1..n, 1..n]$

```
procedure PERMUTÁCIE(m)
if m = 1 then kontrola, či permutácia zodpovedá cyklu v grafe
          else PERMUTÁCIE(m − 1)
               for i = m − 1 downto 1 do
                   if M[A[m + 1], A[i]]
                      then vymeň A[m] a A[i]
                           PERMUTÁCIE(m − 1)
                           vymeň A[m] a A[i]
                   fi
               od
fi
```

Hamiltonovský cyklus - riešenie založené na permutáciach

označme $T(n)$ počet výmen, ktoré urobí počas výpočtu algoritmus Permutácie(n)

$$T(n) = \begin{cases} 0 & \text{pre } n = 1 \\ nT(n-1) + 2(n-1) & \text{pre } n \geq 2 \end{cases}$$

indukciou vzhľadom k n sa ukáže $T(n) \leq 2n! - 2$

porovnanie zložitostí dvoch algoritmov pre problém Hamiltonovského cyklu

- $\mathcal{O}(s^n)$ – algoritmus založený na reťazcoch
- $\mathcal{O}((n-1)!)$ – algoritmus založený na permutáciach

algoritmus založený na reťazcoch je lepší ak $s \leq n/e$

Grafové algoritmy

1. Zložitosť
2. Dátové štruktúry
3. Metódy návrhu algoritmov
4. Grafové algoritmy
   - Prieskum grafov a grafová súvislosť
   - Kostry
   - Najkratšie cesty
   - Toky v grafoch
5. Algoritmy pre prácu s reťazcami

Prieskum grafov a grafová súvislosť

Reprezentácia grafov

formy reprezentácie
- zoznam nasledníkov
- matica susednosti

(ne)výhody
- hustý vs. riedky graf
- dotaz na prítomnosť hrany v grafe
- generovanie zoznamu nasledníkov
- pomenovanie vrcholov
- maticové operácie, paralelizácia výpočtu

Prieskum grafu

vstup graf $G = (V, H)$ a vrchol $s \in V$

úloha nájsť všetky vrcholy grafu dosiahnuteľné z vrcholu s

algoritmy
- prieskum do šírky (BFS)
- prieskum do hĺbky (DFS)

prieskumy DFS a BFS

- sú aplikovateľné na neorientované aj na orientované grafy
- sú základom pre ďalšie algoritmy
  - topologické usporiadanie vrcholov grafu
  - rozklad grafu na silne súvislé komponenty
  - testovanie bipartity
  - ....

Topologické usporiadanie vrcholov grafu

- prieskumy DFS a BFS určujú usporiadanie vrcholov grafu (podľa poradia, v akom boli preskúmané)

orientovaný acyklický graf *dag*

- topologické usporiadanie vrcholov v *dag* $G = (V, H)$ je lineárne usporiadanie všetkých vrcholov grafu také, že ak G obsahuje hranu (u, v), tak v usporiadané vrchol u predchádza vrchol v
- grafická vizualizácia: *horizontálne usporiadanie vrcholov grafu také, že každá hrana grafu smeruje zľava doprava*
- aplikácie

Topologické usporiadanie - algoritmus

základom algoritmu je DFS prieskum grafu

1. vytvor prázdny zoznam
2. aplikuj DFS na G
3. vrchol vlož na začiatok zoznamu v okamihu, keď je ukončený jeho prieskum
4. zoznam určuje usporiadanie vrcholov

Silne súvislé komponenty grafu

- **silne súvislá komponenta** (SCC) orientovaného grafu $G = (V, H)$ je **maximálna** množina vrcholov $C \subseteq V$ taká, že pre každú dvojicu vrcholov $u, v \in C$ platí $u \rightsquigarrow v$ a $v \rightsquigarrow u$
- rôzne algoritmy pre výpočet SCC

Silne súvislé komponenty - DFS algoritmus

- $G^T = (V, H^T), \;\; H^T = \{(u,v) | (v,u) \in H\}$
- G a G^T majú rovnaké maximálne silne súvislé komponenty

1. aplikuj DFS na G a vrcholy grafu usporiadaj v poradí, v akom bol ukončený ich prieskum
2. vypočítaj G^T
3. aplikuj DFS na G^T na vrcholy grafu v zostupnom poradí (tj. začíname od vrchola, ktorého prieskum bol ukončený ako posledný)
4. vrcholy každého DFS stromu tvoria samostatnú SCC

Silne súvislé komponenty - BFS algoritmus

nájdi silne súvislú komponentu obsahujúcu daný vrchol s

1. aplikuj BFS na G z vrcholu s
2. aplikuj BFS na G^T z vrcholu s
3. vypočítaj prienik množín dosiahnuteľných vrcholov z oboch prieskumov

Bipartitné grafy

graf sa nazýva bipartitný práve ak jeho množinu vrcholov je možné rozdeliť na dve disjunktné množiny tak, že žiadne dva vrcholy z jednej množiny nie sú spojené hranou

grafická vizualizácia: vrcholy grafu je možné zafarbiť dvoma farbami tak, že každé dva vrcholy spojené hranou majú rôznu farbu

[3] liché délky

Bipartitné grafy

graf sa nazýva bipartitný práve ak jeho množinu vrcholov je možné rozdeliť na dve disjunktné množiny tak, že žiadne dva vrcholy z jednej množiny nie sú spojené hranou

grafická vizualizácia: vrcholy grafu je možné zafarbiť dvoma farbami tak, že každé dva vrcholy spojené hranou majú rôznu farbu

ak graf G je bipartitný, tak neobsahuje cyklus nepárnej dĺžky[3]

[3] liché délky

Testovanie bipartity - BFS algoritmus

BFS prieskum z vrcholu s definuje vrstvy $L_0, L_1, L_2, \dots$ s vlastnosťami

- $L_0 = \{s\}$
- vrstva L_1 je tvorená všetkými vrcholmi, ktoré susedia s vrcholom s
- predpokladajme, že sú definované vrstvy $L_0, \dots L_j$
- vrstva L_{j+1} je tvorená vrcholmi, ktoré nepatria do žiadnej z predchádzajúcich vrstiev a tvoria hranu s niektorým vrcholom z vrstvy L_j

[4] sudé
[5] liché

Testovanie bipartity - BFS algoritmus

BFS prieskum z vrcholu s definuje vrstvy $L_0, L_1, L_2, \dots$ s vlastnosťami

- $L_0 = \{s\}$
- vrstva L_1 je tvorená všetkými vrcholmi, ktoré susedia s vrcholom s
- predpokladajme, že sú definované vrstvy $L_0, \dots L_j$
- vrstva L_{j+1} je tvorená vrcholmi, ktoré nepatria do žiadnej z predchádzajúcich vrstiev a tvoria hranu s niektorým vrcholom z vrstvy L_j

využitie BFS vrstiev pre testovanie bipartitnosti

- keď vrchol v pridávame do množiny $L[i]$ pre i párne[4], priradíme mu modrú farbu
- keď vrchol v pridávame do množiny $L[i]$ pre i nepárne[5], priradíme mu oranžovú farbu
- graf je bipartitný práve ak sme žiadnemu vrcholu nepriradili dve rôzne farby

[4] sudé

[5] liché

Testovanie bipartity - BFS algoritmus, korektnosť

Lemma 18

Nech G je súvislý graf a nech $L_1, L_2, L_3 \ldots$ sú vrstvy určené BFS prieskumom grafu. Potom práve jedno z nasledovných tvrdení je pravdivé.

(i) *V grafe G neexistuje hrana spájajúca dva vrcholy z tej istej vrstvy. V takom prípade graf G je bipartitný a zafarbenie vrcholov je určené BFS vrstvami (vrcholy vrstvy s párnym indexom majú modrú farbu, vrcholy vrstvy s nepárnym indexom majú oranžovú farbu).*

(ii) *V grafe G existuje hrana, ktorá spája dva vrcholy z jednej vrstvy. V takom prípade G obsahuje cyklus nepárnej dĺžky a nie je bipartitný.*

Kostry

1. Zložitosť
2. Dátové štruktúry
3. Metódy návrhu algoritmov
4. Grafové algoritmy
   - Prieskum grafov a grafová súvislosť
   - Kostry
   - Najkratšie cesty
   - Toky v grafoch
5. Algoritmy pre prácu s reťazcami

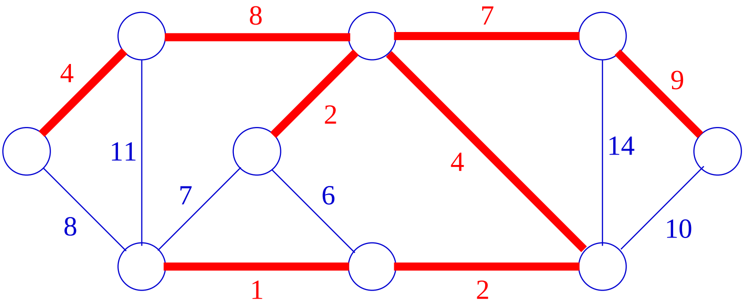
Minimálna kostra grafu

- je daný súvislý, neorientovaný graf $G = (V, H)$ spolu s ohodnotením (váhovou funkciou) $w : H \to \mathbb{R}^+$
- množinu $K \subseteq H$ nazveme kostrou grafu G práve ak graf $G' = (V, K)$ je súvislý a acyklický
- definujeme váhu kostry predpisom

$$w(K) \stackrel{\text{def}}{=} \sum_{h \in K} w(h)$$

- minimálnou kostrou rozumieme kostru s minimálnou váhou

Minimálna kostra grafu - príklad



minimálna kostra K má váhu

$$w(K) = 4 + 8 + 7 + 9 + 2 + 4 + 1 + 2 = 37$$

Budovanie minimálnej kostry

inicializácia: $K = \emptyset$
do K postupne (po jednej) pridávame hrany až kým nedosiahneme kostru

Invariant

K je podmnožinou nejakej minimálnej kostry

Budovanie minimálnej kostry

inicializácia: $K = \emptyset$
do K postupne (po jednej) pridávame hrany až kým nedosiahneme kostru

Invariant

K je podmnožinou nejakej minimálnej kostry

- nech K je množina hrán, ktorá je podmnožinou nejakej minimálnej kostry grafu G
- hranu $h \in H$ nazveme bezpečnou hranou pre množinu $K \subseteq H$ práve ak $h \notin K$ a $K \cup \{h\}$ je stále podmnožinou nejakej minimálnej kostry.

Obecný algoritmus pre nájdenie minimálnej kostry

$\textsc{Min-kostra}((V, H), w)$
$K \leftarrow \emptyset$
while K netvorí kostru **do**
    vyber hranu h, ktorá je bezpečná pre K
    $K \leftarrow K \cup \{h\}$
od
return K

- korektnosť algoritmu plynie z definície bezpečných hrán
- algoritmus vždy zastaví, pretože cyklus **while** prebehne vždy práve $|V| - 1$ krát
- najobtiažnejšou časťou algoritmu je nájdenie bezpečenej hrany pre aktuálne K

Hľadanie bezpečných hrán

rez $(S, V - S)$ v grafe je ľubovoľné rozdelenie množiny vrcholov do dvoch podmnožín

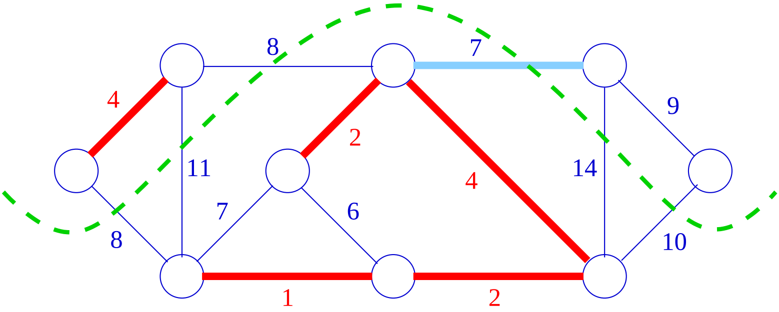
hrana h pretína rez práve ak jeden z jej koncových vrcholov patrí do S a druhý do $V - S$

rez $(S, V - S)$ rešpektuje množinu hrán K práve ak žiadna hrana z množiny K nepretína rez $(S, V - S)$

hrana h je ľahkou hranou vzhľadom k rezu $(S, V - S)$ práve ak tento rez pretína a má zo všetkých takých hrán minimálnu váhu

Príklad

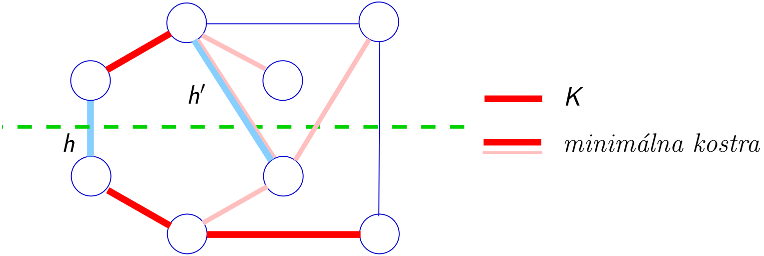
príklad rezu rešpektujúceho množinu červených hrán
ľahhká hrana rezu je zvýraznená modrou farbou



Hľadanie bezpečných hrán

Lemma 19

Nech $G = (V, H)$ je súvislý neorientovaný graf s váhovou funkciou $w : H \to \mathbb{R}^+$. Ďalej nech $K \subseteq H$ je množina hrán, ktorá je podmnožinou nejakej minimálnej kostry grafu G. Uvažujme ľubovoľný rez $(S, V - S)$, ktorý rešpektuje množinu u K a hranu h, ktorá je ľahkou hranou vzhľadom k rezu $(S, V - S)$. Potom h je bezpečná pre množinu K



Hľadanie bezpečných hrán

Dôkaz

- nech T je nejaká minimálna kostra, ktorá obsahuje K
- ak $h \in T$, Lemma platí
- ak $h \notin T$, tak skonštruujeme minimálnu kostru, ktorá obsahuje $K \cup \{h\}$ a tým dokážeme, že h je bezpečná
- množina $T \cup \{h\}$ určite obsahuje cyklus a preto musí existovať hrana $h' \neq h$ z tohoto cyklu, ktorá pretína rez $(S, V - S)$
- označme $X \stackrel{\text{def}}{=} T' \cup \{h\} - \{h'\}$
- zrejme (V, X) je súvislý graf a pretože $|T| = |X|$, tak X je kostra grafu G
- z minimality kostry T plynie $w(X) \geq w(T) \Rightarrow w(h) \geq w(h')$
- z ľahkosti hrany h plynie $w(h) \leq w(h')$
- spolu $w(h) = w(h')$ a preto aj X je minimálna kostra
- pretože $K \cup \{h\} \subseteq X$, tak je bezpečná pre K. ■

Kruskalov algoritmus

- množina K tvorí les
- v každom kroku Kruskalovho algoritmu vyberáme hranu h s minimálnou váhou zo všetkých hrán spájajúcich rôzne komponenty z K
- nech K_1, K_2 sú stromy, ktoré hrana h spája; je zrejmé, že hrana h je ľahkou hranou vzhľadom k rezu $(K_1, V - K_1)$, a preto je podľa lemmatu 19 bezpečnou hranou pre množinu K

- Kruskalov algoritmus je príkladom hladového algoritmu: v každom kroku vyberáme lokálne najlepšiu variantu
- implementácia algoritmu využíva štruktúru $\textsc{Union-Find}$: v jednotlivých množinách sú práve vrcholy z rovnakej komponenty vzhľadom k K

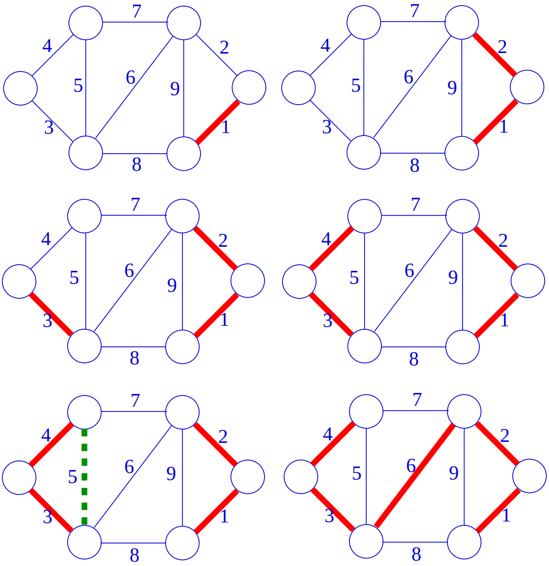
Kruskalov algoritmus

```
1 KRUSKAL((V, H), w)
2 K ← ∅
3 foreach v ∈ V do MAKE-SET(v) od
4 utrieď hrany podľa w do neklesajúcej postupnosti
5 foreach (u, v) ∈ H v tomto poradí do
6    if FIND-SET(u) ≠ FIND-SET(v)
7       then K ← K ∪ {u, v}
8              UNION(u, v) fi od
9 return K
```

Zložitosť

- inicializácia (riadky 2-4) má zložitosť $\Theta(|V|) + \Theta(|H| \log |H|)$
- cyklus 5-8 prebehne $|H|$ krát, celkove se tak prevedie $\Theta(|H|)$ operací FIND-SET, UNION, ktoré majú zložitosť $\mathcal{O}(|H|\alpha(|V|))$
- celková zložitosť Kruskalova algoritmu je $\mathcal{O}(|H| \log |H|)$

Kruskalov algoritmus



Primov algoritmus

- v prípade Primovho algoritmu množina K vždy tvorí jediný strom
- na začiatku zvolíme ľubovoľný vrchol r za koreň a postupne budeme pridávať ďalšie vrcholy, resp. hrany
- označme V_K množinu tých vrcholov, z ktorých vedie nejaká hrana $h \in K$ (v prípade $K = \emptyset$ definujeme $V_K = \{r\}$)
- v každom kroku vyberieme nejakú hranu h, ktorá je ľahká vzhľadom k rezu $(V_K, V - V_K)$ a pridáme ju do K
- Lemma 19 zaručuje korektnosť postupu
- Primov algoritmus je hladový
- efektívna implementácia Primovho algoritmu využíva dátovú štruktúru Q, nad ktorou sa dajú realizovať operácie Extract-Min a Decrease-Key

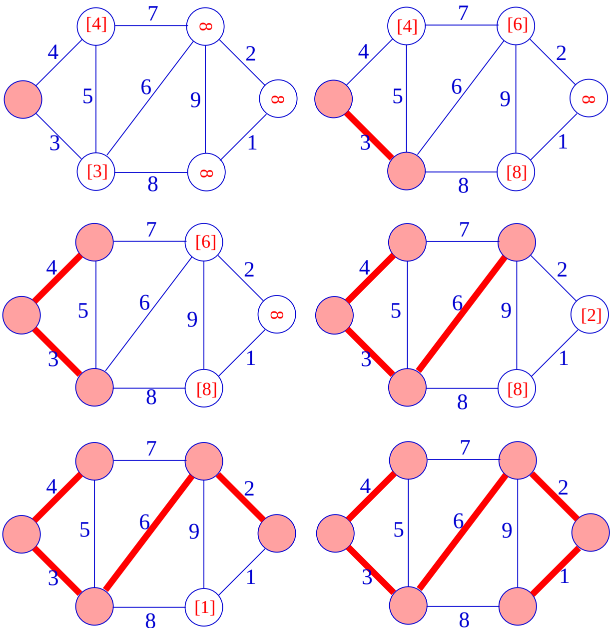
Primov algoritmus

$\textsc{Prim}((V,H),w,r)$
$Q \leftarrow V$
foreach $v \in Q$ **do** $key[v] \leftarrow \infty$ **od**
$key[r] \leftarrow 0$
$\pi[r] \leftarrow \textsc{Nil}$
while $Q \neq \emptyset$ **do**
    $u \leftarrow \textsc{Extract-Min}(Q)$
    foreach v takový, že $(u,v) \in H$ **do**
      if $v \in Q \wedge w(u,v) < key[v]$
        then $\pi[v] \leftarrow u$
            $\textsc{Decrease-Key}(Q,v,w(u,v))$ **fi od od**
return $\{(\pi[v],v) \mid v \in V, v \neq r\}$

$\pi[v]$ označuje otca vrcholu v v aktuálnom strome
po skončení algoritmu je tento strom minimálnou kostrou

Primov algoritmus – zložitosť

- najlepšie výsledky dosiahneme ak budeme implementovať dátovú štruktúru Q ako Fibonacciho haldu
- inicializácia (riadky 1-5) má zložitos´t $\Theta(|V|)$
- v priebehu **while** cyklu se prevedie $|V|$ operácií $\textsc{Extract-Min}$ a $|H|$ operácií $\textsc{Decrease-Key}$
- v prípade Fibonacciho haldy je amortizovaná zložitosť operácie $\textsc{Extract-Min}$ $\mathcal{O}(\log|V|)$ a amortizovaná zložitosť operácie $\textsc{Decrease-Key}$ jekonštantná
- celkom sa v algoritmu prevedie $|V|$ operácií $\textsc{Extract-Min}$ a nanajvýš $|H|$ operácií $\textsc{Decrease-Key}$
- celková zložitos algoritmu je preto $\mathcal{O}(|V|\log|V| + |H|)$


[4] 7 ∞ 4 2 5 6 9 ∞ 3 1 [3] 8 ∞
[4] 7 [6] 4 2 5 6 9 ∞ 3 1 8 [8]
7 [6] 4 2 5 6 9 ∞ 3 1 8 [8]
7 4 2 5 6 9 [2] 3 1 8 [8]
7 4 2 5 6 9 3 1 8 [1]
7 4 2 5 6 9 3 1 8

Najkratšie cesty

1. Zložitosť
2. Dátové štruktúry
3. Metódy návrhu algoritmov
4. Grafové algoritmy
   - Prieskum grafov a grafová súvislosť
   - Kostry
   - Najkratšie cesty
   - Toky v grafoch
5. Algoritmy pre prácu s reťazcami

Definícia pojmov

vstup orientovaný graf $G = (V, H)$ spolu s ohodnotením hrán $w : H \to \mathbb{R}$

cesta v G $p =< v_0, v_1, \ldots, v_k >$, označenie $v_0 \overset{p}{\leadsto} v_k$

dĺžka cesty $p =< v_0, v_1, \ldots, v_k >$ je súčet dĺžok jej hrán,

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

dĺžka najkratšej cesty z vrcholu u do vrcholu v je definovaná predpisom

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

najkratšia cesta z vrcholu u do vrcholu v je ľubovoľná cesta p (z u do v) pre ktorú $w(p) = \delta(u, v)$

Problém najkratšej cesty

najkratšie cesty z daného vrcholu do všetkých vrcholov, SSSP

- obrátením orientácie hrán sa redukuje na problém najkratších ciest zo všetkých vrcholov do daného vrchola

najkratšia cesta medzi danou dvojicou vrcholov

- obrátením orientácie hrán sa redukuje na problém SSSP
- pre problém nie je známy žiaden asymptoticky rýchlejší algoritmus než pre SSSP

najkratšie cesty medzi všetkými dvojicami vrcholov

- dá sa redukovať na SSSP
- efektívnejšie algoritmy

úlohou je vypočítať dĺžku najkratšej cesty
a nájsť (niektorú) najkratšiu cestu

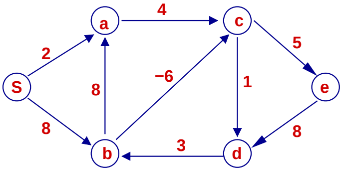
Optimálna subštruktúra najkratších ciest

Lemma 20

Nech $p = < v_1, \ldots, v_k >$ *je najkratšia cesta z vrcholu* v_1 *do vrcholu* v_k *a nech pre ľubovoľné* i, j *také, že* $1 \leq i \leq j \leq k$, *je* $p_{ij} = < v_i, \ldots, v_j >$ *podcesta cesty* p *z* v_i *do* v_j.
Potom p_{ij} *je najkratšia cesta z* v_i *do* v_j.

algoritmy pre hľadanie najkratších ciest sú založené na princípoch dynamického programovania a na princípe hladových algoritmov

Hrany so zápornou dĺžkou



- graf obsahuje cyklus zápornej dĺžky
- ak nejaká cesta z vrcholu u do vrcholu v obsahuje cyklus zápornej dĺžky, tak $\delta(u, v) = -\infty$ a neexistuje najkratšia cesta z u do v
- v prípade, že graf obsahuje hrany so zápornou dĺžkou, problém najkratšej cesty je formulovaný ako
  1. rozhodni, či graf obsahuje cyklus zápornej dĺžky
  2. ak graf neobsahuje cyklus zápornej dĺžky, tak nájdi najkratšie cesty
- niektoré algoritmy pre hľadanie najkratších ciest sú korektné len za predpokladu, že graf neobsahuje žiadnu hranu zápornej dĺžky (prípadne za predpokladu, že graf neobsaje cyklus zápornej dĺžky)

Cykly

môže najkratšia cesta obsahovať cyklus?

- nemôže obsahovať cyklus zápornej dĺžky
  viz predchádzajúci slajd
- nemôže obsahovať cyklus kladnej dĺžky
  spor s predpokladom, že je najkratšia
- existuje najkratšia cesta, ktorá neobsahuje cyklus dĺžky 0

Reprezentácia najkratších ciest

- najkratšie cesty z vrchola s
- pre daný graf $G = (V, H)$ a každý jeho vrchol $v \in V$ udržujeme atribút $p[v]$, ktorého hodnotou je buď vrchol grafu alebo *Nil*
- hodnoty $p[.]$ indukujú graf predchodcov $G_p = (V_p, H_p)$, kde

$$V_p = \{v \in V \mid p[v] \neq Nil\} \cup \{s\}$$

$$H_p = \{(p[v], v) \in H \mid v \in V_p \setminus \{s\}\}$$

- na konci výpočtu je G_p stromom najkratších ciest
- strom najkratších ciest s koreňom s je orientovaný podgraf $G' = (V', H')$ grafu G taký, že
  1. V' je množina vrcholov dosiahnuteľných z vrcholu s
  2. G' je strom s koreňom s
  3. pre každý vrchol $v \in V'$ je jediná cesta z s do v v G' najkratšou cestou z s do v v G

Reprezentácia dĺžky najkratších ciest

- najkratšie cesty z vrchola s
- pre daný graf $G = (V, H)$ a každý jeho vrchol $v \in V$ udržujeme hodnotu $d[v]$, ktorá je horným odhadom na dĺžku najkratšej cesty z koreňa s do v

$\textsc{Inicializácia}(G, s)$
for $v \in V$ **do**
  $d[v] \leftarrow \infty$
  $p[v] \leftarrow Nil$
od
$d[s] \leftarrow 0$

Najkratšie cesty z koreňa s do ostatných vrcholov

algoritmy

- Bellmanov - Fordov algoritmus
- algoritmus pre orientované acyklické grafy
- Dijkstrov algoritmus pre grafy s nezápornými dĺžkami hrán

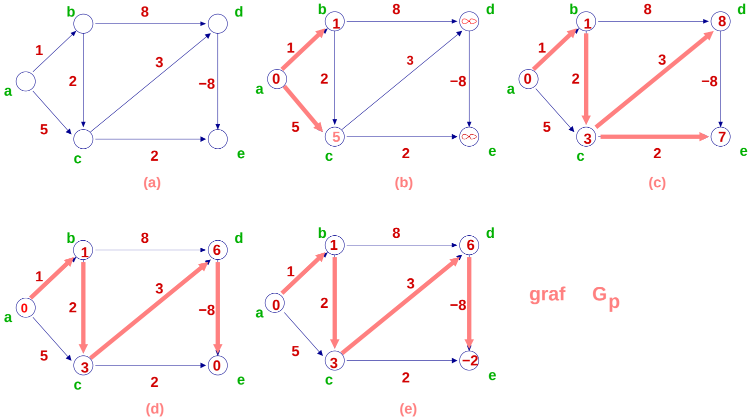
Bellmanov - Fordov algoritmus

- algoritmus je založený na postupnom zlepšovaní hodnôt $d[u]$
- ak vrcholu u zlepšíme hodnotu $d[u]$, musíme následne preskúmať všetky hrany $(u, v) \in H$ a ak je to možné, zlepšiť hodnotu $d[v]$ (procedúra RELAXÁCIA)
- algoritmus vráti hodnotu *true* práve ak graf neobsahuje cyklus zápornej dĺžky dosiahnuteľný z koreňa s

Bellmanov-Fordov algoritmus

```
BELLMAN-FORD(G, w, s)
INICIALIZÁCIA(G, s)
for i = 1 to |V| − 1 do
    for každú hranu (u, v) ∈ H do RELAXÁCIA(u, v, w) od od
for každú hranu (u, v) ∈ H do
    if d[v] > d[u] + w(u, v) then return false fi od
return true
```

```
RELAXÁCIA(u, v, w)
if d[v] > d[u] + w(u, v)
  then d[v] ← d[u] + w(u, v)
       p[v] ← u
fi
```



výpočet algoritmu Bellman-Ford pre graf s koreňom a
v každej iterácii cyklu 2-3 relaxujeme hrany v poradí (c, e), (d, e), (b, d), (c, d), (b, c), (a, c), (a, b)

Bellmanov-Fordov algoritmus, korektnosť

Lemma 21

Nech v grafe G neexistuje záporný cyklus dosiahnuteľný z koreňa s a pre G zavoláme procedúru $\textsc{Inicializácia}(G, s)$. *Potom G_p je strom s koreňom s a tento invariant zostáva v platnosti po vykonaní ľubovoľnej postupnosti relaxácií.*

Dôkaz Tvrdenie triviálne platí bezprostredne po inicializácii.
Uvažujme graf G_p, ktorý dostaneme po vykonaní postupnosti relaxácií.
Ukážeme, že je acyklický a že pre každý vrchol $v \in V_p$ existuje v G_p jediná cesta z s do v.
Predpokladajme, že v G_p existuje cyklus $c =< v_0, v_1, \ldots, v_k >$, kde $v_0 = v_k$. Potom $p[v_i] = v_{i-1}$ a búno môžeme predpokladať, že cyklus vznikol počas $\textsc{Relaxácia}(v_{k-1}, v_k, w)$. Ľahko overíme, že všetky vrcholy cyklu sú dosiahnuteľné z vrcholu s.

Tesne pred volaním $\text{Relaxácia}(v_{k-1}, v_k, w)$ platí pre každý vrchol v_i cyklu, $i = 1, \ldots, k-1$,

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

Pretože počas uvedenej relaxácie sa mení hodnota $p[v_k]$, tak bezprostredne pred jej vykonaním platí

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

Sčítaním všetkých nerovností dostávame

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

Pretože $\sum_{i=1}^{k} d[v_{i-1}] = \sum_{i=1}^{k} d[v_i]$, tak dostávame nerovnosť

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

čo je spor s predpokladom o neexistencii záporného cyklu v G.

Zostáva overiť, že pre každý vrchol $v \in V_p$ existuje v G_p práve jedna cesta z s do v. Existencia cesty je zrejmá. Pretože hodnota $p[v]$ je určená jednoznačne, nemôžu v G_p viesť do žiadneho vrcholu dve hrany. Preto v G_p nemôžu existovať ani dve cesty z s do v. $\square$

Lemma 22

Nech v grafe G neexistuje záporný cyklus dosiahnuteľný z koreňa s. Predpokladajme, že pre G voláme INICIALIZÁCIA(G, s) *a potom relaxujeme hrany v ľubovoľnom poradí tak, že na konci výpočtu pre každý vrchol v platí $d[v] = \delta[s, v]$. Potom G_p je strom najkratších ciest s koreňom s.*

Dôkaz

Zrejme V_p je množina vrcholov dosiahnuteľných z s. Podľa Lemy 21 je G_p strom. Zostáva ukázať, že pre každý vrchol $v \in V_p$ je jediná cesta $s \overset{p}{\rightsquigarrow} v$ v G_p najkratšou cestou z s do v. Nech $p =< v_0, v_1, \ldots, v_k >$, kde $v_0 = s$ a $v_k = v$. Pre každé $i = 1, 2, \ldots, k$ máme $d[v_i] = \delta(s, v_i)$ a $w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1})$. Sčítaním všetkých nerovností dostávame $w(p) = \delta(s, v_k)$. □

Lemma 23

Nech v grafe G neexistuje záporný cyklus dosiahnuteľný z koreňa s. Po tom po $|V|-1$ iteráciach cyklu 3-4 algoritmu $\text{BELLMAN-FORD}(G, w, s)$ *platí $d[v] = \delta[s, v]$ pre všetky vrcholy v dosiahnuteľné z vrcholu s.*

Dôkaz

Nech $p = < v_0, v_1, \ldots, v_k >$, kde $v_0 = s$ a $v_k = v$, je najkratšia acyklická cesta z s do v. Potom $k \leq |V| - 1$. Indukciou vzhľadom k i (s využitím optimálnej subštruktúry najkratších ciest) dokážeme, že po i-tej iterácii cyklu 3-4 platí $d[v_i] = \delta(s, v_i)$. Naviac platí, že ak premenná $d[v_i]$ nadobudne hodnotu $\delta(s, v_i)$, tak jej hodnota sa už v priebehu výpočtu nemení. □

Theorem 24

Ak v grafe G neexistuje žiaden záporný cyklus dosiahnuteľný z s, tak algoritmus $\textsc{Bellman-Ford}(G, w, s)$ *vráti hodnotu true, pre každý vrchol v platí $d[v] = \delta[s, v]$ a G_p je strom najkratších ciest s koreňom s.*
Ak v G existuje záporný cyklus dosiahnuteľný z s, tak algoritmus vráti hodnotu false.

Dôkaz

Nech v G neexistuje záporný cyklus dosiahnuteľný z s. Ak vrchol v je dosiahnuteľný z s, tak podľa Lemy 23 na konci výpočtu platí $d[v] = \delta[v]$. Ak v nie je dosiahnuteľný, tak z invariantu výpočtu $d[v] \geq \delta[v]$ plynie, že na konci je $\delta[v] = \infty$. Na konci výpočtu pre každú hranu $(u, v) \in H$ platí

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

a preto žiaden test na riadku 6 nevráti hodnotu *false*.

Naopak, nech v G existuje záporný cyklus $c = < v_0, v_1, \dots, v_k >$, kde $v_0 = v_k$, dosiahnuteľný z s. Predpokladajme, že algoritmus vráti *true*. Potom pre všetky $i = 1, 2, \dots, k$ platí
$d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$. Sčítame všetky nerovnosti

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

Pretože $\sum_{i=1}^{k} d[v_{i-1}] = \sum_{i=1}^{k} d[v_i]$ a pre všetky i je $d[v_i] < \infty$, tak dostávame nerovnosť

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

čo je spor s predpokladom o zápornej dĺžke cyklu c. $\square$

Zložitosť Bellmanovho-Fordovho algoritmu

- inicializácia grafu má zložitosť $\Theta(|V|)$
- cyklus 3–4 má zložitosť $\Theta(|H|)$; počet jeho opakovaní je $|V| - 1$
- celková zložitosť je $\mathcal{O}(nm)$, keď n je počet vrcholov a m je počet hrán grafu

Varianty Bellman-Fordovho algoritmu

- líšia sa poradím, v akom sa relaxujú hrany grafu a v spôsobe, akým sa zisťuje existencia dosiahnuteľného záporného cyklu v grafe
- pre špeciálne typy grafov existujú efektívnejšie algoritmy

Dijkstrov algoritmus

- rieši problém najkratších ciest z koreňa s do ostatných vrcholov grafu pre grafy s nezáporným ohodnotením hrán
- algoritmus udržuje množinu S vrcholov, pre ktoré sa už vypočítala dĺžka najkratšej cesty
- algoritmus opakovane vyberá vrchol $u \in V \setminus S$ s najkratšou cestou a relaxuje hrany vychádzajúce z u

Dijkstrov algoritmus

$\textsc{Dijkstra}(G, w, s)$
$\textsc{Inicializácia}(G, s)$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow (u \in Q \;\wedge\; d[u] = \min\{d[x] \mid x \in Q\})$
    $S \leftarrow S \cup \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for každý vrchol v taký, že $(u, v) \in H$ **do**
        if $d[v] > d[u] + w(u, v)$
            then $d[v] \leftarrow d[u] + w(u, v)$
                $p[v] \leftarrow u$ **fi**
    od od

Korektnosť Dijkstrovho algoritmu

Theorem 25

Pre orientovaný graf $G = (V, H)$ s nezáporným ohodnotením hrán w a koreňom s Dijkstrov algoritmus skončí a pre všetky $u \in V$ platí $d[u] = \delta(s, u)$.

Dôkaz Ukážeme, že invariantom **while** cyklu algoritmu je

pre každý vrchol $v \in S$ platí $d[v] = \delta(s, v)$

Platnosť invariantu po inicializácii je zrejmá.
Nech u je prvý vrchol, ktorý zaradíme do S a pritom $d[u] \neq \delta(s, u)$.
Zrejme $u \neq s$ a u je dosiahnuteľný z s (v opačnom prípade by platilo $\delta(s, u) = \infty = d[u]$). Nech p je najkratšia cesta z s do u. Cestu p môžeme dekomponovať na dve cesty ako $s \stackrel{p_1}{\rightsquigarrow} x \rightarrow y \stackrel{p_2}{\rightsquigarrow} u$ tak, že bezprostredne pred zaradením u do S všetky vrcholy cesty p_1 patria do S a $y \notin S$.
Potom $d[x] = \delta(s, x)$ a aj $d[y] = \delta(s, y)$ (pri zaradení x do S bola relaxovaná hrana (x, y)).

Cesta p_2 je najkratšou cestou z y do u čo spolu s nezáporným ohodnotením hrán implikuje $\delta(s,y) \leq \delta(s,u)$. Pretože ale vrchol u bol vybraný do S, tak zároveň bezprostredne pred zaradením u do S platí $d[u] \leq d[y] = \delta(s,y)$. Spojením dostávame
$\delta(s,u) \leq d[u] \leq \delta(s,y) \leq \delta(s,u)$ a teda $d[u] = \delta(s,u)$.
Výpočet končí keď $Q = \emptyset$, tj. $V = S$ a tvrdenie vety je dokázané. $\square$

Zložitosť Dijkstrovho algoritmu

- zložitosť závisí od spôsobu reprezentácie množiny Q, efektivity výberu prvku u s minimálnou hodnotou $d[u]$ (operácia $\textsc{Extract\_Min}$, $|V|$ krát) a aktualizácie hodnôt $d[v]$ pre vrcholy susediace s u (operácia $\textsc{Decrease\_Key}$, $|H|$ krát)
- ak hodnoty $d[v]$ sú uložené v poli, tak $\textsc{Extract\_Min}$ má zložitosť $\mathcal{O}(|V|)$ a $\textsc{Decrease\_Key}$ má konštantnú zložitosť; celkove

$$\mathcal{O}(|V|^2 + |H|) = \mathcal{O}(|V|^2)$$

- pri reprezentácii pomocou Fibonacciho haldy je amortizovaná cena každej $\textsc{Extract\_Min}$ operácie $\mathcal{O}(\log |V|)$ a amortizovaná cena $\textsc{Decrease\_Key}$ je konštantná; celkove

$$\mathcal{O}(|V| \log |V| + |H|)$$

Implementácia	INSERT	EXCTRACT_MIN	DECREASE_KEY
Pole	$\mathcal{O}(1)$	$\mathcal{O}(\|V\|)$	$\mathcal{O}(1)$
Binár. halda	$\mathcal{O}(\log\|V\|)$	$\mathcal{O}(\log\|V\|)$	$\mathcal{O}(\log\|V\|)$
d-árna halda	$\mathcal{O}(\frac{\log\|V\|}{\log d})$	$\mathcal{O}(\frac{d\log\|V\|}{\log d})$	$\mathcal{O}(\frac{\log\|V\|}{\log d})$
Fibon. halda	$\mathcal{O}(1)$	$\mathcal{O}(\log\|V\|)$	$\mathcal{O}(1)$

Implementácia	$\|V\|\times$ INSERT + $\|V\|\times$EXTRACT_MIN + $\|H\|\times$DECREASE_KEY
Pole	$\mathcal{O}(\|V\|^2)$
Binárna halda	$\mathcal{O}((\|V\|+\|H\|)\log\|V\|)$
d-árna halda	$\mathcal{O}((\|V\|\times d+\|H\|)\frac{\log\|V\|}{\log d})$
Fibon. halda	$\mathcal{O}(\|V\|\log\|V\|+\|H\|)$

Najkratšie cesty v orientovanom acyklickom grafe

- optimálne poradie relaxácie hrán v Bellmanovom-Fordovom algoritme je také, keď vždy relaxujeme hranu (u, v) pre ktorú $d[u] = \delta(s, u)$
- pre obecný graf určiť poradie relaxácií tak, aby bola dodržaná uvedená podmienka, môže byť rovnako náročné ako vypočítať najkratšie cesty
- špeciálne pre acyklické grafy sa toto poradie dá vypočítať jednoducho: požadovanú vlastnosť má topologické usporiadanie vrcholov grafu
- každá hrana grafu sa relaxuje iba raz, celková zložitosť algoritmu je $\Theta(n + m)$

Inicializácia$((V, H), s)$
for každý vrchol u v topologickom utriedení **do**
    for každý vrchol v taký, že $(u, v) \in H$ **do**
        Relaxácia(u, v, w)
    od
od

Najkratšie cesty medzi všekými dvojicami

algoritmy

- algoritmus násobenia matíc $\Theta(|V|^3 \log |V|)$
- Floydov-Warshallov algorimus $\Theta(|V|^3)$
- Johnsonov algoritmus $\mathcal{O}(|V|^2 \log |V| + |V| \cdot |H|)$

Algoritmus násobenia matíc

- predpokladáme, že graf je reprezentovaný $n \times n$ $(n = |V|)$ maticou susednosti $W = (w_{ij})$, kde

$$w_{ij} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } i = j \\ \text{váha hrany } (i,j) & \text{ak } i \neq j \text{ a } (i,j) \in H \\ \infty & \text{ak } i \neq j \text{ a } (i,j) \notin H \end{cases}$$

- predpokladáme, že graf neobsahuje žiadne záporné cykly
- algoritmus pracuje na princípoch dynamického programovania

Štruktúra optimálneho riešenia

- uvažujme najkratšiu cestu p z i do j
- cesta p je konečná a má m hrán
- ak $i = j$ tak p má dĺžku 0 a neobsahuje žiadnu hranu ($m = 0$)
- ak $i \neq j$, tak cestu p môžeme rozložiť na $i \overset{p'}{\rightsquigarrow} k \rightarrow j$, kde
  - p' má $m - 1$ hrán,
  - p' je najkratšou cestou z i do k a
  - $\delta(i,j) = \delta(i,k) + w_{kj}$

Hodnota optimálneho riešenia

- označme $l_{ij}^{(m)}$ minimálnu dĺžku cesty z i do j, ktorá má nanajvýš m hrán
- pre $m = 0$ platí

$$l_{ij}^{(0)} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{ak } i = j \\ \infty & \text{ak } i \neq j \end{cases}$$

- pre $m > 0$ platí

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

- pretože najkratšia cesta z i do j nemôže mať viac než $|V| - 1$ hrán, tak $\delta(i, j) = l_{ij}^{(n-1)}$

Výpočet hodnoty optimálneho riešenia

- počítame matice $L^{(1)}, \ldots L^{(n-1)}$, kde $L^{(m)} = (l_{ij}^{(m)})$
- základom výpočtu je procedúra $\textsc{Extend}(L, W)$, ktorá pre dané matice $L^{(m)}$ a W vypočíta maticu $L^{(m+1)}$

$\textsc{Extend}(L, W)$
nech $L' = (l'_{ij})$ je $n \times n$ matica
for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    $l'_{ij} \leftarrow \infty$
    for $k \leftarrow 1$ **to** n **do**
      $l'_{ij} \leftarrow \min\{l'_{ij}, l_{ik} + w_{kj}\}$ **od**
  od od
return L'

Zložitosť $\textsc{Extend}$ je $\Theta(n^3)$, zložitosť celého výpočtu je $\Theta(n^4)$.

Efektívnejšia varianta

Násobenie matíc použité v procedúre EXTEND je asociatívne a preto môžeme vypočítať $L^{(n-1)}$ efektívnejšie:

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

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

Zložitosť celého výpočtu je $\theta(n^3 \log n)$.

$\textsc{All\_Pairs\_Shortest\_Path}(W)$

$L^{(1)} \leftarrow W$
$m \leftarrow 1$
while $m < n - 1$ **do**
  $L^{(2m)} \leftarrow \textsc{Extend}(L^{(m)}, L^{(m)})$
  $m \leftarrow 2m$
od
return $L^{(m)}$

Floydov - Warshallov algoritmus

- daný je orientovaný graf $G = (V, H)$, $V = \{1, 2, \ldots, n\}$, s ohodnotením hrán $w : H \to \mathbf{R}$ a bez záporných cyklov a zadaný $n \times n$ maticou susednosti $W = (w_{ij})$
- existuje najkratšia cesta, ktorá neobsahuje cyklus
- najkratšia cesta p z vrcholu i do vrcholu j obsahuje ako svoje vnútorné vrcholy ľubovoľné vrcholy z V; označme ich $\{1, 2, \ldots k\}$
- vyberme vrchol k cesty p
- cesta p sa rozpadá na cestu z i do k a cestu z k do j; ani jedna z týchto ciest neobsahuje vrchol k
- Floydov - Warshallov algoritmus využíva vzťah medzi množinou ciest z vrcholu i do vrcholu j obsahujúcou vrcholy $\{1, 2, \ldots k\}$ a množinou ciest z i do j, ktorých vnútorné vrcholy sú z množiny $\{1, 2, \ldots k-1\}$

- ak k nie je vnútorným vrcholom cesty p, tak najkratšia cesta z i do j s vnútornými vrcholmi z $\{1, 2, \ldots, k-1\}$ je zároveň najkratšou cestou i do j s vnútornými vrcholmi z $\{1, 2, \ldots, k\}$
- ak k je vnútorným vrcholom cesty p, tak cestu p môžeme rozdeliť na $i \overset{p_1}{\leadsto} k \overset{p_2}{\leadsto} j$. Podľa Lemy 20 je p_1 najkratšou cestou z i do k s vnútornými vrcholmi z $\{1, 2, \ldots, k-1\}$. Podobne p_2 je najkratšou cestou z k do j s vnútornými vrcholmi z $\{1, 2, \ldots, k-1\}$.
- označme $d_{ij}^{(k)}$ dĺžku najkrašej cesty z i do j s vnútornými vrcholmi z množiny $\{1, 2, \ldots, k\}$
- platí

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

FLOYD-WARSHALL(W)

$D^{(0)} \leftarrow W$

for $k = 1$ **to** n

    do for $i = 1$ **to** n

        do for $j = 1$ **to** n

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

        od

    od

od

return $D^{(n)}$

časová zložitosť algoritmu je $\mathcal{O}(n^3)$

Konštrukcia najkratšej cesty

- spolu s maticou D dĺžok najkratších ciest počítame maticu Π predchodcov vrcholov na najkratšej ceste
- počítame postupnosť matíc $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)} = \Pi$, kde $\pi_{ij}^{(k)}$ je definované ako predchodca vrchola j na najkratšej ceste z i do j s vnútornými vrcholmi z $\{1, 2, \ldots, k\}$
- hodnotu $\pi_{ij}^{(k)}$ definujeme rekurzívne
- ak $k = 0$ tak najkratšia cesta z i do j neobsahuje žiaden vnútorný vrchol a preto

$$\pi_{ij}^{(0)} \stackrel{\text{def}}{=} \begin{cases} Nil & \text{ak } i = j \text{ alebo } w_{ij} = \infty \\ i & \text{ak } i \neq j \text{ a } w_{ij} < \infty \end{cases}$$

pre $k \geq 1$ rozlíšime dva prípady.

- ak najkratšia cesta obsahuje vnútorný vrchol k, tj. má tvar $i \rightsquigarrow k \rightsquigarrow j$, tak predchodca vrcholu j na tejto ceste je zhodný s predchodcom vrchola j na najkratšej ceste z k do j s vnútormými vrcholmi z množiny $\{1, \ldots, k-1\}$.
- v opačnom prípade je predchodca vrcholu j zhodný s predchodcom vrchola j na najkratšej ceste z k do j s vnútormými vrcholmi z $\{1, \ldots, k-1\}$.

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

Johnsonov algoritmus

- daný je orientovaný graf $G = (V, H)$, $V = \{1, 2, \ldots, n\}$, s ohodnotením hrán $w : H \rightarrow \mathbf{R}$ a zadaný $n \times n$ maticou susednosti $W = (w_{ij})$
- Johnsonov algoritmus je založený na technike transformácie ohodnotenia hrán a využíva efektivitu Dijkstrovho algoritmu
- ak ohodnotenie všetkých hrán je nezáporné, tak pre každý vrchol použijeme Dijkstrov algoritmus
- ak G má hrany so záporným ohodnotením, tak zistíme, či graf má záporné cykly
- ak G nemá záporné cykly, tak ohodnotenia hrán modifikujeme tak, aby ohodnotenia všetkých hrán boli nezáporné a pre každý vrchol použijeme Dijkstrov algoritmus

Transformácia ohodnotenia hrán

nové ohodnotenie $\widehat{w}$ musí spĺňať:

1. pre každé $u, v \in V$ platí:
   p je najkratšia cesta z u do v pri ohodnotení w
   práve vtedy ak
   p je najkratšia cesta z u do v pri ohodnotení $\widehat{w}$
2. pre každú hranu $(u, v) \in H$ platí $\widehat{w}(u, v) \geq 0$.

Transformácia ohodnotenia hrán

Lemma 26

Nech $G = (V, H)$ *je orientovaný graf s ohodnotením* $w : H \to \mathbb{R}$. *Nech* $h : V \to \mathbb{R}$ *je ľubovoľná funkcia. Pre každú hranu* $(u, v) \in V$ *definujeme*

$$\widehat{w}(u, v) \stackrel{def}{=} w(u, v) + h(u) - h(v).$$

Potom pre každú cestu p *z* v_0 *do* v_k *platí:*

$$w(p) = \delta(v_0, v_k) \text{ vtedy a len vtedy ak } \widehat{w}(p) = \widehat{\delta}(v_0, v_k)^6.$$

Naviac, G *pri ohodnotení* w *má záporný cyklus vtedy a len vtedy ak* G *pri ohodnotení* $\widehat{w}$ *má záporný cyklus*

[6] $\widehat{\delta}(v_0, v_k)$ je dĺžka najkratšej cesty z v_0 do v_k pri ohodnotení $\widehat{w}$.

Dôkaz. Nech $p =< v_0, v_1, \dots, v_k >$. Potom

$$\begin{aligned}
\widehat{w}(p) &= \sum_{i=1}^{k} \widehat{w}(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\
&= w(p) + h(v_0) - h(v_k)
\end{aligned}$$

Preto ak p je najkratšia cesta z v_0 do v_k pri ohodnotení w, tak je najkratšou cestou aj pri ohodnotení $\widehat{w}$ a naopak.

Nech $c =< v_0, v_1, \dots, v_k >$, $v_0 = v_k$, je cyklus. Analogicky odvodíme

$$\widehat{w}(c) = w(c) + h(v_0) - h(v_k) = w(c).$$

Z toho plynie druhé tvrdenie Lemy. $\square$

Detekcia záporného cyklu a voľba funkcie h

cieľom je rozhodnúť, či graf má záporný cyklus a ak nie, tak zvoliť funkciu h tak, aby hodnota $\widehat{w}(u,v)$ bola pre každú hranu nezáporná

skonštruujeme $\overline{G} = (\overline{V}, \overline{H})$ s ohodnotením $\overline{w} : \overline{H} \to \mathbb{R}$

- $\overline{V} = V \cup \{s\}$, $s \notin V$
- $\overline{H} = H \cup \{(s,v) \mid v \in V\}$
- $\overline{w}(u,v) = w(u,v)$ pre $(u,v) \in H$ a $\overline{w}(s,v) = 0$ pre $v \in V$

platí

- c je záporný cyklus v $G \Leftrightarrow c$ je záporný cyklus v $\overline{G}$
- každý cyklus v $\overline{G}$ je dosiahnuteľný z koreňa s
- žiadna najkratšia cesta z u do v $(u, v \neq s)$ neobsahuje vrchol s

pre každé $v \in \overline{V}$ a $(u, v) \in \overline{H}$ definujeme

$$h(v) \stackrel{\text{def}}{=} \delta(s, v)$$

$$\widehat{w}(u, v) \stackrel{\text{def}}{=} w(u, v) + h(u) - h(v)$$

pre každé $(u, v) \in \overline{H}$ platí

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

$$\widehat{w}(u, v) \geq 0$$

$\textsc{Johnson}(G)$

```
skonštruuj G̅ = (V̅, H̅)
if BELLMAN-FORD(G̅, w̅, s) = false
  then graf má záporný cyklus
   else foreach vrchol v ∈ V
          do h(v) ← δ(s, v) (vypočítané alg. BELL.-FORD) od
        foreach hranu (u, v) ∈ H
          do ŵ(u, v) ← w(u, v) + h(u) − h(v) od
        foreach vrchol u ∈ V
          do DIJKSTRA(G, ŵ, u)
          foreach vrchol v ∈ V
            do d_uv ← δ̂(u, v) + h(v) − h(u)
          od od
fi
return D = (d_uv)
```

Zložitosť Johnsonovho algoritmu

zložitosť algoritmu Bellman-Ford $\approx$ $(|V| \cdot |H|)$

$+$

$|V|$-krát zložitosť Dijkstrovho alg. $\approx$ $|V| \cdot (|V| \cdot \log |V| + |H|)$

$=$

$$(|V|^2 \log |V| + |V||H|)$$

pre riedke grafy je Johnsonov algoritmus efektívnejší než Floyd-Warshallov algoritmus

Toky v grafoch

Toky v sieťach

- sieť $G = (V, H)$ je orientovaný graf, ktorého každá hrana $(u, v) \in H$ má nezápornú kapacitu (priepustnosť) $c(u, v) \geq 0$
- predpokladáme, že ak H obsahuje hranu (u, v), tak neobsahuje hranu (v, u)
- ak $(u, v) \notin H$, tak (z technických dôvodov) definujeme $c(u, v) = 0$
- predpokladáme, že H neobsahuje slučky (hrany tvaru (u, u))
- predpokladáme, že v sieti sú vyznačené dva vrcholy: zdroj s a cieľ t
- predpokladáme, že všetky vrcholy grafu ležia na nejakej ceste z s do t

Maximálny tok v sieti

- nech $G = (V, H)$ je sieť s kapacitnou funkciou c, zdrojom s a cieľom t
- tok v sieti G je funkcia $f : V \times V \to \mathbb{R}$ spĺňajúca nasledujúce dve podmienky

  kapacitné ohraničenie pre všetky $u, v \in V$

  $$0 \leq f(u, v) \leq c(u, v)$$

  podmienka kontinuity pre všetky $u \in V \setminus \{s, t\}$

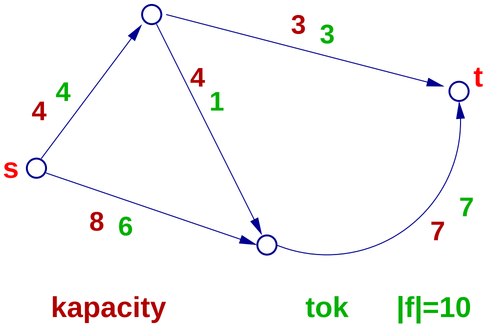
  $$\sum_{v \in V} f(u, v) = \sum_{v \in V} f(v, u)$$

- hodnota $|f|$ toku f je definovaná ako

  $$|f| \stackrel{\text{def}}{=} \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

- problém maximálneho toku je definovaný ako úloha nájsť v sieti tok s maximálnou hodnotou

Toky v sietiach - príklad



Algoritmy pre problém maximálneho toku

Fordova-Fulkersonova metóda

- algoritmus Edmonds-Karp / Dinitz
- algoritmus najširších ciest
- algoritmus zjemňovania stupnice

metóda Push-Relabel

- algoritmus Relabel-to-front
- ......

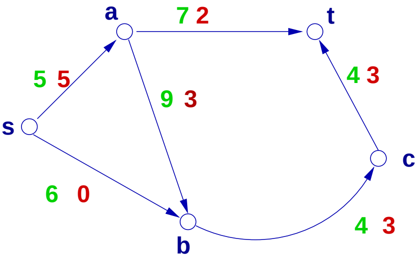
Fordova - Fulkersonova metóda

- iteratívna metóda
- začína s tokom nulovej hodnoty a postupne zvyšuje hodnotu toku
- v každej iterácii hľadá tzv. zlepšujúcu cestu z s do t, po ktorej môže zvýšiť hodnotu toku
- zlepšujúca cesta sa hľadá v asociovanej tzv. reziduálnej sieti
- v každej iterácii sa tok na hrane môže zvýšiť alebo znížiť
- konkrétne algoritmy podľa spôsobu hľadania zlepšujúcej cesty

Ford_Fulkersonova_metóda

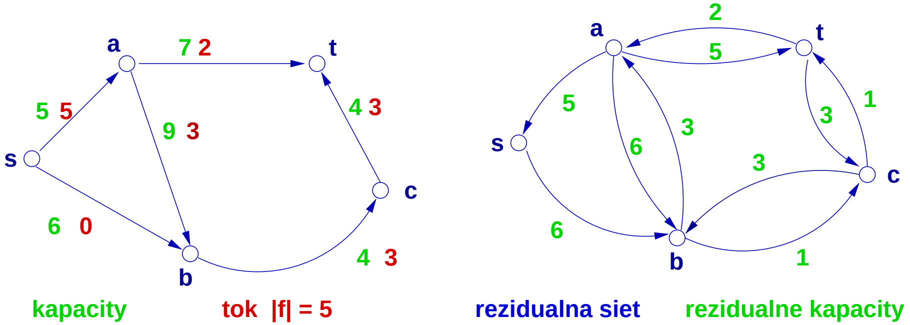
inicializuj tok f na 0
while existuje zlepšujúca cesta p **do**
    zlepši hodnotu toku na ceste p **od**
return f

Zlepšujúca cesta a reziduálna sieť - príklad



kapacity tok |f| = 5

Zlepšujúca cesta a reziduálna sieť - príklad



Reziduálna sieť

- reziduálna sieť G_f asociovaná sieti G a toku f obsahuje hrany, ktorých kapacity reprezentujú hodnotu, o ktorú sa môže zmeniť tok na hrane

- ak $f(u,v) < c(u,v)$ tak hodnotu toku na hrane (u,v) môžeme zvýšiť až o rozdiel $c(u,v) - f(u,v)$ a preto do reziduálnej siete dáme hranu (u,v) s kapacitou $c(u,v) - f(u,v)$

- ak $f(u,v) > 0$, tak hodnotu toku na hrane (u,v) môžeme znížiť až o $f(u,v)$ a preto do reziduálnej siete dáme hranu (v,u) s kapacitou $f(u,v)$

Reziduálna sieť

nech G je sieť so zdrojom s a cieľom t a nech f je tok v G

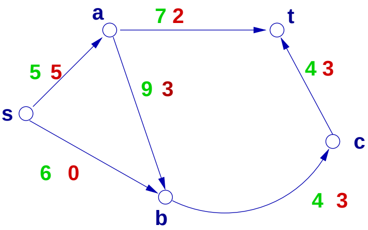
- pre vrcholy $u, v \in V$ definujeme reziduálnu kapacitu hrany (u, v) predpisom

$$c_f(u,v) \stackrel{\text{def}}{=} \begin{cases} c(u,v) - f(u,v) & \text{ak } (u,v) \in H \\ f(v,u) & \text{ak } (v,u) \in H \\ 0 & \text{inak} \end{cases}$$

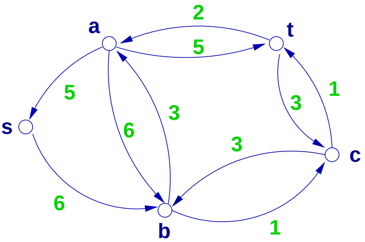
- reziduálna sieť asociovaná so sieťou G a tokom f je $G_f = (V, H_f)$,

$$H_f \stackrel{\text{def}}{=} \{(u,v) \in V \times V \mid c_f(u,v) > 0\}$$

Reziduálna sieť a zvýšenie toku - príklad

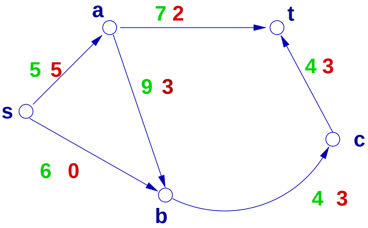


kapacity **tok |f| = 5**

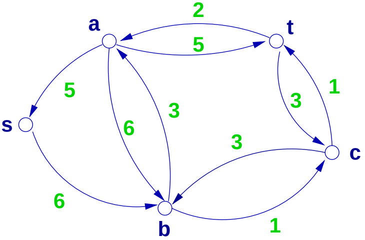


rezidualna siet **rezidualne kapacity**

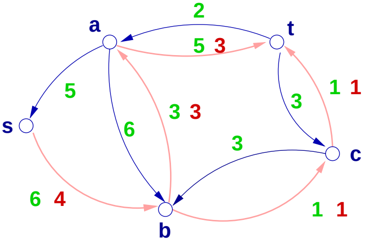
Reziduálna sieť a zvýšenie toku - príklad



kapacity tok |f| = 5

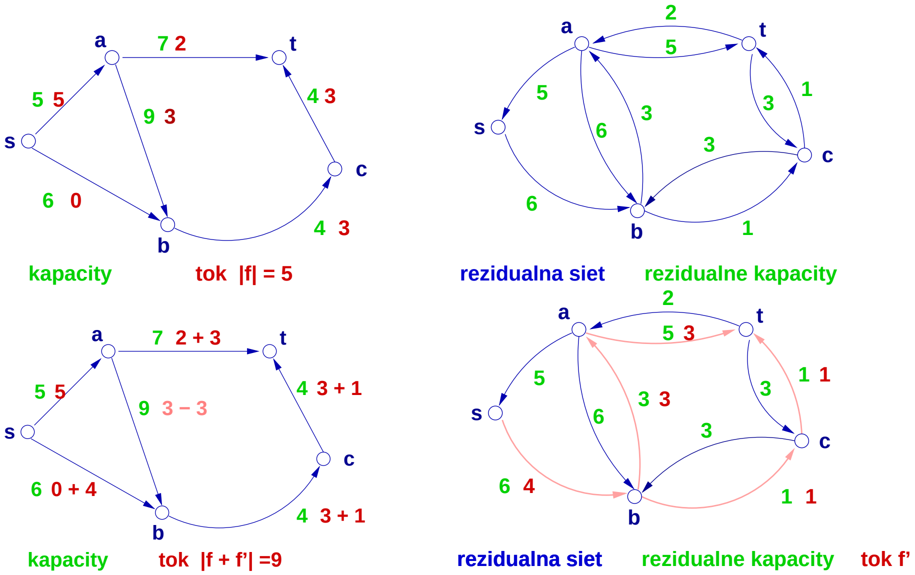


rezidualna siet rezidualne kapacity



rezidualna siet rezidualne kapacity tok f’

Reziduálna sieť a zvýšenie toku - príklad



Reziduálna sieť a zvýšenie toku

- tok a reziduálna sieť poskytujú návod pre nájdenie toku s vyššou hodnotou
- ak f' je tok v reziduálnej sieti, tak definujeme zvýšenie toku f o hodnotu f' ako funkciu $f \uparrow f' : V \times V \to \mathbb{R}$ definovanú predpisom

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{ak}(u,v) \in H \\ 0 & \text{inak} \end{cases}$$

Reziduálna sieť a zvýšenie toku - korektnosť

Lemma 27

Nech $G = (V, H)$ je sieť so zdrojom s a cieľom t, a nech f je tok v G. Nech G_f je reziduálna sieť asociovaná s G a f, a nech f' je tok v G_f. Potom funkcia $f \uparrow f'$ je tokom v G s hodnotou $|f| + |f'|$.

Dôkaz

- $f \uparrow f'$ spĺňa kapacitné ohraničenia

$$\begin{aligned}
(f \uparrow f')(u,v) &= f(u,v) + f'(u,v) - f'(v,u) \\
&\geq f(u,v) + f'(u,v) - f(u,v) \\
&= f'(u,v) \\
&\geq 0
\end{aligned}$$

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

využili sme $f'(v,u) \leq f(u,v)$ a nezápornosť $f'(v,u)$

Dôkaz

- $f \uparrow f'$ spĺňa podmienky kontinuity pre $u \in V \setminus \{s, t\}$

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

využili sme podmienku kontinuity pre f'

Dôkaz

- hodnota toku $f \uparrow f'$

$$
\begin{aligned}
|f \uparrow f'| &= \sum_{v \in V} (f \uparrow f')(s,v) - \sum_{v \in V} (f \uparrow f')(v,s) \\
&= \sum_{v \in V_1} (f \uparrow f')(s,v) - \sum_{v \in V_2} (f \uparrow f')(v,s) \\
&= \sum_{v \in V_1} (f(s,v) + f'(s,v) - f'(v,s)) - \sum_{v \in V_2} (f(v,s) + f'(v,s) - f'(s,v)) \\
&= \sum_{v \in V_1} f(s,v) - \sum_{v \in V_2} f(v,s) \\
&\quad + \sum_{v \in V_1} f'(s,v) + \sum_{v \in V_2} f'(v,s) - \sum_{v \in V_1} f'(v,s) - \sum_{v \in V_2} f'(v,s) \\
&= \sum_{v \in V_1} f(s,v) - \sum_{v \in V_2} f(v,s) + \sum_{v \in V_1 \cup V_2} f'(s,v) - \sum_{v \in V_1 \cup V_2} f'(v,s) \\
&= |f| + |f'|
\end{aligned}
$$

$V_1 = \{v \mid (s,v) \in H\}$, $V_2 = \{v \mid (v,s) \in H\}$

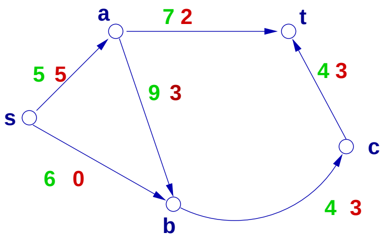
Tok v reziduálnej sieti

problém

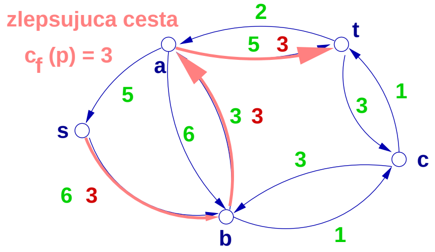
ako nájsť tok v reziduálnej sieti ???

riešenie

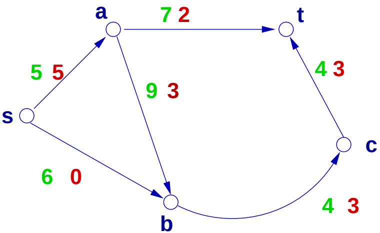
cesta v reziduálnej sieti



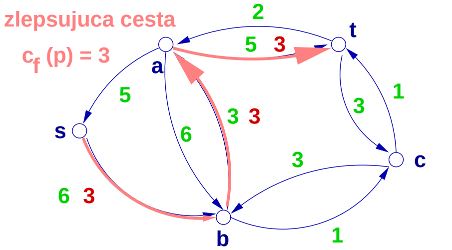
kapacity tok |f| = 5



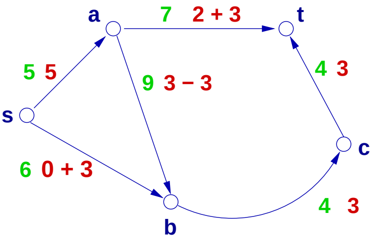
rezidualna siet rezidualne kapacity tok



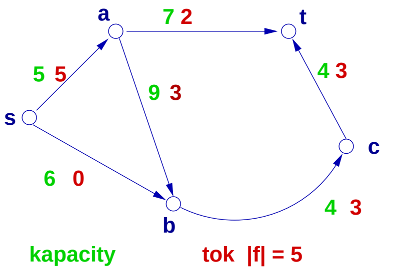
kapacity tok |f| = 5



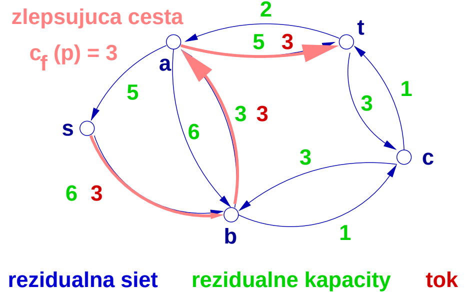
rezidualna siet rezidualne kapacity tok



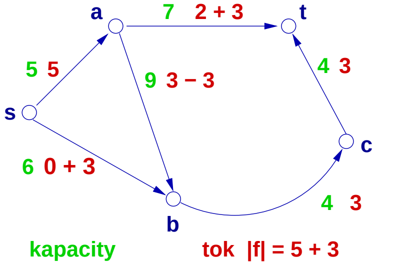
kapacity tok |f| = 5 + 3



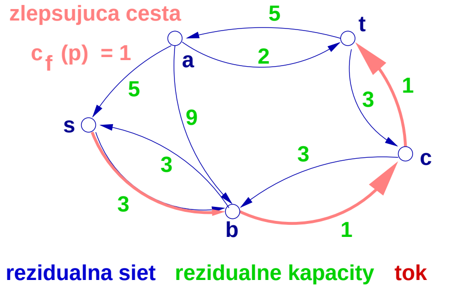
a
t
s
b
c
7 2
5 5
9 3
4 3
6 0
4 3
kapacity
tok |f| = 5



zlepsujuca cesta
c_f (p) = 3
2
5 3
5
3 3
6
1
3
6 3
3
1
a
t
s
b
c
rezidualna siet
rezidualne kapacity
tok



a
t
s
b
c
7 2 + 3
5 5
9 3 − 3
4 3
6 0 + 3
4 3
kapacity
tok |f| = 5 + 3



zlepsujuca cesta
c_f (p) = 1
5
2
5
9
3
1
3
3
3
1
a
t
s
b
c
rezidualna siet
rezidualne kapacity
tok

Zlepšujúce cesty

- zlepšujúca cesta p je jednoduchá cesta cesta zo zdroja s do cieľa t v reziduálnej sieti G_f
- z definície reziduálnej sieta vyplýva, že môžeme zvýšiť tok na hrane (u, v) zlepšujúcej cesty a to až na hodnotu $c_f(u, v)$ bez porušenia kapacitného ohraničenia hrany (u, v) resp. (v, u) v sieti G
- pri hľadaní toku musíme vziať do úvahy všetky hrany zlepšujúcej cesty
- definujeme reziduálnu kapacitu cesty p ako

$$c_f(p) \stackrel{\text{def}}{=} \min\{c_f(u, v) \mid (u, v) \text{ leží na } p\}$$

Zlepšujúce cesty - korektnosť

Lemma 28

Nech G je sieť, f tok v G a p zlepšujúca cesta v G_f.
Definujme funkciu $f_p : V \times V \to \mathbb{R}$ predpisom

$$f_p(u,v) = \begin{cases} c_f(p) & \textit{ak } (u,v) \textit{ leží na ceste } p \\ 0 & \textit{inak} \end{cases}$$

Potom f_p je tok v G_f a jeho hodnota je $|f_p| = c_f(p) > 0$.

Dôkaz

Overíme, že f_p v G_f spĺňa kapacitné ohraničenia a podmienky kontinuity.$\square$

Poznámka Funkcia $f \uparrow f_p$ je tokom v G a má vyššiu hodnotu než f.

Korektnosť Ford-Fulkersonovej metódy

rez (S, T) v sieti G je rozdelenie V na S a T také, že $s \in S$ a $t \in T$

tok rezu je

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

kapacita rezu je

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

minimálny rez je rez, ktorého kapacita je spomedzi všetkých rezov grafu najmenšia

Korektnosť Ford-Fulkersonovej metódy

Lema 29

Nech f je tok v sieti G a nech (S,T) je rez v G. Potom

$$|f| = f(S,T) \leq c(S,T).$$

Dôkaz

$$
\begin{aligned}
|f| &= \sum_{v\in V} f(s,v) - \sum_{v\in V} f(v,s) + \sum_{u\in S\setminus\{s\}} \Big(\sum_{v\in V} f(u,v) - \sum_{v\in V} f(v,u)\Big) \\
&\quad \textit{preskupením sumandov} \\
&= \sum_{v\in V}\sum_{u\in S} f(u,v) - \sum_{v\in V}\sum_{u\in S} f(v,u) \\
&\quad \textit{využijeme disjunktnosť množín } S \textit{ a } T \\
&= \sum_{v\in S}\sum_{u\in S} f(u,v) + \sum_{v\in T}\sum_{u\in S} f(u,v) - \sum_{v\in S}\sum_{u\in S} f(v,u) - \sum_{v\in T}\sum_{u\in S} f(v,u) \\
&= \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u) \\
&= f(S,T)
\end{aligned}
$$

Nerovnosť $f(S,T) \le c(S,T)$ plynie z kapacitného ohraničenia $f(u,v) \le c(u,v)$. $\square$

Korektnosť Ford-Fulkersonovej metódy

Veta 30

Ak f je rez v sieti G, tak nasledujúce podmienky sú ekvivalentné:

1. f **je maximálny tok v sieti** G
2. **reziduálna sieť** G_f **neobsahuje žiadnu zlepšujúcu cestu**
3. $|f| = c(S, T)$ **pre nejaký rez** (S, T) **siete** G**.**

Dôkaz (1) $\Rightarrow$ (2) Plynie z Lemy 28.

(2) $\Rightarrow$ (3) Nech G_f nemá žiadnu zlepšujúcu cestu. Definujme $S = \{v \in V \mid \text{ existuje cesta z } s \text{ do } v \text{ v } G_f\}$ a $T = V \setminus S$. Rozdelenie (S, T) je rezom v G_f. Pre každú dvojicu vrcholov u, v takú, že $u \in S$ a $v \in T$ platí $f(u, v) = c(u, v)$ (v opačnom prípade by $(u, v) \in H_f$) a teda $f(S, T) = c(S, T)$. Podľa Lemy 29 platí $|f| = f(S, T)$.

(3) $\Rightarrow$ (1) Podľa Lemy 29 pre každý rez platí $|f| \leq c(S, T)$. Rovnosť $|f| = c(S, T)$ preto implikuje maximalitu toku. $\square$

Fordova Fulkersonova Metóda

FORDOVA-FULKERSONOVA-METÓDA

for každú hranu $(u, v) \in H$ **do**
    $f[u, v] \leftarrow 0$
od
while existuje zlepšujúca cesta p **do**
        $c_f(p) \leftarrow \min\{c_f(u, v) \mid (u, v)$ leží na $p\}$
        for každú hranu (u, v) na p **do**
            if $(u, v) \in H$ **then** $f[u, v] \leftarrow f[u, v] + c_f(p)$
                                        else $f[v, u] \leftarrow f[u, v] - c_f(p)$ **fi**
        od
od
return f

Zložitosť Fordovej-Fulkersonovej metódy

sieť celočíselnými kapacitami

Lema 31

Po každej iterácii Fordovej-Fulkersonovej metódy sú hodnoty $f[u, v]$ celočíselné a reziduálny graf má celočíselné kapacity.

Lema 32

V každej iterácii Fordovej-Fulkersonovej metódy sa hodnota toku zvýši aspoň o 1.

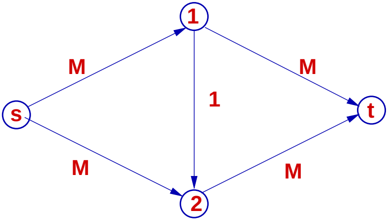
Lema 33

Existuje maximálny tok f^* taký, že tok $f^*(e)$ na každej hrane e je celočíselný.

- zložitosť jednej iterácie je rovná zložitosti nájdenia zlepšujúcej cesty z s do t a je $\mathcal{O}(|H|)$
- celková zložitosť je $\mathcal{O}(|f^*| \cdot |H|)$

Zložitosť Fordovej-Fulkersonovej metódy

príklad grafu, pre ktorý je zložitosť $|f^*| \cdot \mathcal{O}(|H|)$



- počiatočný tok má hodnotu $|f| = 0$
- zlepšujúca cesta $s, 1, 2, t \;\Rightarrow\;$ tok $|f| = 1$
- zlepšujúca cesta $s, 2, 1, t \;\Rightarrow\;$ tok $|f| = 2$
- zlepšujúca cesta $s, 1, 2, t \;\Rightarrow\;$ tok $|f| = 3$
- ...

Zložitosť Fordovej-Fulkersonovej metódy

- sieť s racionálnymi kapacitami dokážeme previesť na sieť s celočíselnými kapacitami

- v prípade, že kapacity hrán sú iracionálne čísla, konečnosť Fordovej-Fulkersonovej metódy nie je zaručená
- *existenciu maximálneho toku zaručuje Veta 30*

Výber zlepšujúcej cesty

algoritmus Edmonds, Karp / Dinic

- vyberá zlepšujúcu cestu najkratšej dĺžky
(dĺžka cesty je rovná počtu hrán cesty)
- k výberu zlepšujúcej cesty používa BFS
- počet iterácii je $\mathcal{O}(|V| \cdot |H|)$
- zložitosť algoritmu je $\mathcal{O}(|V| \cdot |H|^2)$ pre grafy s ľubovoľnými kapacitami

algoritmus najširších ciest

- vyberá zlepšujúcu cestu s maximálnou reziduálnou kapacitou
- počet iterácii je $\mathcal{O}(|V|^2)$
- zložitosť algoritmu je $\mathcal{O}(|V|^2 \cdot |H| \ln |f^*|)$ pre grafy s celočíslenými kapacitami

algoritmus zjemňovania stupnice

- zložitosť je $\mathcal{O}(|H|^2 \log_2 |f^*|)$ pre grafy s celočíslenými kapacitami

Algoritmus zjemňovania stupnice

- upredností́uje zlepšujúce cesty, ktorých reziduálna kapacita je čo najväčšia
- pre daný škálovací parameter Δ hľadá zlepšujúcu cestu, ktorej reziduálna kapacita je aspoň Δ
- z reziduálnej siete odstránime všetky hrany, ktorých kapacita je menšia než Δ
- postupne znižujeme parameter Δ; ako škálovaciu stupnicu volíme mocniny 2
- korektnosť algoritmu je daná tým, že škálovanie sa používa len na rozhodnutie o výbere zlepšujúcej cesty; princíp samotného algoritmu sa nemení

Algoritmus zjemňovania stupnice

ALGORITMUS_ZJEMŇOVANIA_STUPNICE

```
for každú hranu (u, v) ∈ H do
    f[u, v] ← 0
od
Δ ← max{2^i | 2^i ≤ max_{v∈V} c(s, v)}
while Δ ≥ 1 do
        while existuje zlepšujúca cesta p v G_f(Δ) do
                update f
        od
        Δ ← Δ/2
od
return f
```

$G_f(\Delta)$ vznikne z G_f odstránením všetkých hrán, ktorých kapacita je menšia než Δ

Algoritmus zjemňovania stupnice - zložitosť

- ak kapacity hrán sú celočíselné, tak počas celého výpočtu zostáva tok a reziduálne kapacity celočíselné; to garantuje, že pre $\Delta = 1$ je $G_f(\Delta) = G_f$ a po skončení výpočtu algoritmu má f maximálnu možnú hodnotu
- počet iterácií vonkajšieho **while** cyklu je $1 + \lceil \log_2 C \rceil$, kde $C = \sum_{v \in V} c(s, v)$
- každá zvolená zlepšujúca cesta zvýši hodnotu toku aspoň o aktuálnu hodnotu parametra Δ
- nech f je tok po ukončení vnútorného **while** cyklu pre hodnotu Δ; potom existuje rez (S, T) taký, že $c(S, T) \leq |f| + |H|\Delta$ a následne hodnota maximálneho toku je zhora ohraničená výrazom $|f| + |H|\Delta$
- pre fixovaný parameter Δ algoritmus použije nanajvýš $|H|$ zlepšujúcich ciest, nájdenie zlepšujúcej cesty si vyžaduje čas $\mathcal{O}(|H|)$
- algoritmus zjemňovania stupnice má zložitosť $\mathcal{O}(|H|^2 \log_2 C)$

Goldbergova metóda (Push-relabel)

- základ asysmptoticky najrýchlejších algoritmov pre problém maximálneho toku
- generická metóda
- jednoduchá implementácia zložitosti $\mathcal{O}(|V|^2 \cdot |H|)$
  (*viz algoritmus Edmonds-Karp zložitosti* $\mathcal{O}(|V| \cdot |H|^2)$)
- efektívna implementácia zložitosti $\mathcal{O}(|V|^3)$
- ďalšie efektívne implementácie

Goldbergova metóda (Push-relabel)

porovnanie s metódou Fork-Fulkerson

- lokálny charakter
- namiesto budovania zlepšujúcej cesty skúma a upravuje tok len na hranách vychádzajúcich z jedného vrchola
- nezachováva vlastnosť kontinuity
- metóda pracuje s tzv. pseudotokom (*preflow*), ktorý spĺňa kapacitné ohraničenie a do istej miery môže porušiť podmienky kontinuity
- je prípustné, aby tok na hranách vstupujúcich do vrcholu bol väčší než tok na hranách vystupujúcich z vrcholu

Goldbergova metóda (Push-relabel) - neformálny popis

- do vrcholov priteká tok po hranách vstupujúcich do vrcholu; tok sa vo vrchole prerozdeľuje a odteká po hranách odchádzajúcich z vrcholu
- vrchol má rezervnú nádrž, v ktorej môže byť uchované neobmedzené množstvo toku (*pre prípad, že prichádzajúci tok je väčší než odchádzajúci tok*)
- vrchol má svoju výšku; výška vrcholu v priebehu výpočtu vzrastá
- výška vrcholu určuje, akým spôsobom sa mení tok na hranách: tok z nižšieho vrcholu do vyššieho vrcholu môže byť kladný, ale *pretláčať tok* (zvyšovať tok na hrane) môžeme len z vyššieho do nižšieho vrcholu
- zdroj s má fixovanú výšku $|V|$, cieľ t má fixovanú výšku 0; všetky ostané vrcholy majú na začiatku výpočtu výšku 0

Goldbergova metóda (Push-relabel) - neformálny popis

- na začiatku výpočtu sa zo zdroja s pretlačí maximálne možné množstvo toku (tak, aby sa naplnila kapacita všetkých hrán vychádzajúcich z s)
- tok, ktorý pritečie do vnútorného vrcholu siete, sa uloží do rezervnej nádrže a odtiaľ sa postupne pretláča do vrcholov s menšou výškou
- nakoniec nastane situácia, že rezervná nádrž vrchola nie je prázdna, ale všetky hrany, ktoré odchádzajú z vrchola a majú voľnú kapacitu, vedú do vrcholov, ktoré majú rovnakú alebo väčšiu výšku
- v takej situácii zvýšime výšku vrchola tak, aby o 1 prevyšovala výšku najnižšieho vrchola do ktorého vedie hrana s voľnou kapacitou
- výpočet končí, keď už nie je možné pretlačiť viac toku do cieľového vrchola
- pseudotok sa upraví na tok: obsah rezerných nádrží, ktorý nebolo možné pretlačiť smerom k cieľu, sa pošle naspäť k zdroju
- v okamihu, keď sa vyprázdnia všetky rezervné nádrže, sa pseudotok stáva tokom a je zaručené, že je maximálnym tokom

Pseudotok

- pseudotok je funkcia $f : V \times V \to \mathbb{R}$ spĺňajúca
  kapacitné ohraničenie pre všetky $v, w \in V$
  $$0 \leq f(v, w) \leq c(v, w)$$
  modifikovanú podmienku kontinuity pre všetky $v \in V \setminus \{s\}$
  $$\sum_{w \in V} f(w, v) - \sum_{w \in V} f(v, w) \geq 0$$
- vrchol v nazveme aktívny vrchol práve ak
  $$e_f(v) = \sum_{w \in V} f(w, v) - \sum_{w \in V} f(v, w) > 0$$
  hodnotu $e_f(v)$ nazývame aktivitou vrchola v
- ak jediné dva aktívne vrcholy sú s a t, tak pseudotok je tokom
- pseudotoku f môžeme priradiť reziduálnu sieť $G_f = (V, H_f)$

Výška vrchola

- výšku vrchola určuje funkcia $h : V \rightarrow \mathbb{N}_0$
- výška h a pseudotok f sú kompatibilné práve ak

zdroj	$h(s) = \|V\| = n$
cieľ	$h(t) = 0$
výškové rozdiely	pre všetky hrany (v, w) reziduálnej siete $G_f = (V, H_f)$ platí $h(v) \leq h(w) + 1$

Pseudotok a maximálny tok

Lema 34

Ak pseudotok f je kompatibilný s výškou h, tak v reziduálnom grafe G_f neexistuje žiadna $s - t$ cesta.

Dôkaz

- nech existuje jednoduchá cesta $s, v_1, \ldots, v_k = t$
- z kompatibility plynie $h(s) = n$
- hrana (s, v_1) je hranou reziduálnej siete a preto $h(v_1) \geq h(s) - 1 = n - 1$
- indukciou k i overíme, že $h(v_i) \geq n - i$ (špeciálne $h(t) \geq n - k$)
- spor s $h(t) = 0$, pretože $k < n$ (cesta je jednoduchá)

Lema 35

Ak tok f je kompatibilný s výškou h, tak f je maximálny tok.

Dôkaz Lema 34 + Fordova - Fulkersonova metóda

Iniciálny pseudotok a výška

- $h(v) = 0$ pre všetky $v \in V$, $v \neq s$
- $h(s) = n$
- žiadna hrana (s, v) nesmie byť v reziduálnej sieti, pretože by nespĺňala výškový rozdiel
- $f(s, v) = c(s, v)$ pre každú hranu $(s, v) \in H$
- $f(u, v) = 0$ pre ostatné hrany siete

Fakt 36

Iniciálny pseudotok a výška sú kompatibilné

Generický algoritmus

INITIALIZE-PREFLOW(G, s)

for $v \in V$ **do** $\quad h(v) \leftarrow 0; \quad e_f(v) \leftarrow 0$ **od**

$h(s) \leftarrow |V|$

for $(u, v) \in H$ **do** $\quad f(u, v) \leftarrow 0$ **od**

for $(s, v) \in H$ **do**

$\quad f(s, v) \leftarrow c(s, v); \quad e_f(v) \leftarrow c(s, v); \quad e_f(s) \leftarrow e_f(s) - c(s, v)$ **od**

GENERIC-PUSH-RELABEL(G)

INITIALIZE-PREFLOW(G, s)

while existuje aktívny vrchol $v \neq t$ **do**

nech v je aktívny vrchol

if existuje (v, w) spĺňajúca predpoklady operácie PUSH

then PUSH(f, h, v, w)

else RELABEL(f, h, v) **fi**

od

Operácia **Push**

- nech v je aktívny vrchol, $e_f(v) > 0$
- ak v reziduálnom grafe existuje hrana (v, w), tak môžeme pseudotok modifikovať tak, že *pretlačíme* časť toku z v do w

PUSH(f, h, v, w)

1 **predpoklad** $e_f(v) > 0$, $c_f(v, w) > 0$, $h(w) < h(v)$
2 **efekt** operácia pretlačí $\Delta_f(v, w)$ jednotiek toku z v do w
3 $\Delta_f(v, w) \leftarrow \min(e_f(v), c_f(v, w))$
4 **if** $(v, w) \in H$ **then** $f(v, w) \leftarrow f(v, w) + \Delta_f(v, w)$
5 **else** $f(v, w) \leftarrow f(v, w) - \Delta_f(v, w)$ **fi**
6 $e_f(v) \leftarrow e_f(v) - \Delta_f(v, w)$
7 $e_f(w) \leftarrow e_f(w) + \Delta_f(v, w)$
8 **return** (f, h)

- hodnota $\Delta_f(v, w)$ predstavuje bezpečnú zmenu toku
- ak reziduálna hrana je aj hranou siete, tak tok na hrane sa zvýši (riadok 4)
- v opačnom prípade sa tok na hrane zníži (riadok 5)

Operácia **Relabel**

- ak nemôžeme modifikovať pseudotok na žiadnej hrane vychádzajúcej z aktívneho vrchola, musíme zvýšiť výšku vrchola

RELABEL(f, h, v)
predpoklad v je aktívny vrchol a
pre všetky $w \in V$ také, že $(v, w) \in H_f$, platí $h(w) \geq h(v)$
efekt operácia zvýši výšku vrchola v
$h(v) \leftarrow h(v) + 1$
return (f, h)

Analýza algoritmu - korektnosť

Lema 37

V priebehu celého výpočtu algoritmu platí

1. *výška vrcholu je nezáporné celé číslo*
2. *f je pseudotok a ak kapacity hrán siete sú celočíselné, tak f je celočíselný*
3. *pseudotok f a výška v sú kompatibilné*

Ak algoritmus vráti pseudotok f, tak f je maximálny tok.

Dôkaz

- fakt 36 implikuje platnosť pre iniciálne nastavenie
- Push rešpektuje kapacitné ohraničenie hrany a do reziduálneho grafu pridá hranu, ktorá spĺňa výškový rozdiel
- Relabel zvyšuje výšku vrchola v práve ak v reziduálnej sieti nevedie z v hrana do žiadneho vrchola s menšou výškou
- vlastnosť výsledného pseudotoku vyplýva z Lemy 35

Analýza algoritmu - zložitosť Relabel

počet operácií Relabel

- na začiatku má každý vrchol (okrem zdroja) výšku 0
- pri každej zmene sa výška zvýši o 1
- potrebujeme odhadnúť maximálnu možnú výšku vrchola

Analýza algoritmu - zložitosť

Lema 38

Nech f je pseudotok. Ak vrchol v je aktívny, tak v reziduálnej sieti G_f existuje cesta z v do zdroja s.

Dôkaz

- označme A množinu všetkých vrcholov reziduálnej siete, z ktorých vedie cesta do s; $B = V \setminus A$; $s \in A$
- potrebujeme ukázať, že všetky aktívne vrcholy patria do A
- tok na hrane (x, y) pre $x \in A, y \notin A$ je nulový (inak by v rez. sieti bola hrana (y, x) a $y \in A$
- uvážme sumu aktivít vrcholov z B

$$0 \leq \sum_{v \in B} e_f(v) = \sum_{v \in B} (f^{in}(v) - f^{out}(v))$$

Analýza algoritmu - zložitosť RELABEL

- uvážme sumu aktivít vrcholov z B

$$0 \leq \sum_{v \in B} e_f(v) = \sum_{v \in B} (f^{in}(v) - f^{out}(v))$$

- ak hrana má obidva koncové vrcholy v B, prispeje do sumy nulovou hodnotou
- ak hrana má v B len koncový vrchol, tak jej počiatočný vrchol je v A a tok na hrane je nulový
- ak hrana má v B len počiatočný vrchol, tak jej tok sa v sume vyskytuje len so záporným znamienkom
- $$0 \leq \sum_{v \in B} e_f(v) = -f^{out}(B)$$
- pretože toky na hranách sú nezáporné, má každý vrchol z B nulovú aktivitu (nie je aktívny)

Analýza algoritmu - zložitosť Relabel

Lema 39

Počas celého výpočtu platí pre každý vrchol $h(v) \leq 2n - 1$.

Dôkaz

- iniciálne hodnoty $h(s) = n$ a $h(t) = 0$ sa počas výpočta nemenia
- operácia Relabel mení výšku aktívneho vrchola v
- podľa Lemy 38 existuje z v cesta P do s, $|P| \leq n - 1$
- hrany cesty spĺňajú výškové rozdiely, tj. na každej hrane klesne výška nanajvýš o 1
- $h(v) - h(s) \leq |P|$

Dôsledok 40

Celkový počet operácií Relabel *je nanajvýš* $2n^2$.

Analýza algoritmu - zložitosť PUSH

počet operácií PUSH

- rozlíšime dva typy PUSH operácií
- operácia PUSH(f, h, v, w) sa nazýva saturujúca práve ak po jej prevedení je hrana (v, w) v reziduálnej sieti nasýtená, t.j. nezostane v reziduálnej sieti
- ostatné operácie nazývame nesaturujúce

Analýza algoritmu - zložitosť Push

Lema 41

Celkový počet saturujúcich Push *operácií je nanajvýš $2nm$.*

Dôkaz

- uvážme hranu (v, w) z reziduálneho grafu
- po aplikácie saturujúcej $\textsc{Push}(f, h, v, w)$ operácie je $h(v) = h(w) + 1$ a (v, w) nie je reziduálna
- aby sme mohli znovu aplikovať $\textsc{Push}(f, h, v, w)$, musím zvýšiť tok na (w, v)
- k tomu potrebujeme, aby sa výška w zvýšila aspoň o 2 (musí byť vyššia než výška v)
- výšku w môžeme zvýšiť o hodnotu 2 maximálne $n - 1$ krát
- to nám dáva horný odhan na počet saturujúcich $\textsc{Push}(f, h, v, w)$ operácií

Analýza algoritmu - zložitosť Push

Lema 42

Celkový počet nesaturujúcich Push *operácií je nanajvýš* $4n^2m$.

- pre kompatibilný pseudotok f a výšku h definujeme ich potenciál

$$\Phi(f,h) = \sum_{v.e_f(v)>0} h(v)$$

- iniciálny potenciál je nulový a počas celého výpočtu je nezáporný
- nesaturujúca Push operácia zníži potenciál aspoň o 1
- potenciál zvyšujú operácie Relabel a saturujúca Push
- Relabel zvýši potenciál o 1, celkove maximálne o $2n^2$
- saturujúca Push(f,h,v,w) môže zaktívniť vrchol w a tým sa zvýši potenciál o $h(w) \leq 2n-1$
- celkový počet saturujúcich Push operácií je nanajvýš $2nm$
- saturujúce Push operácie môžu zvýšiť potenciál nanajvýš o hodnotu $2nm(2n-1)$
- počet nesaturujúcich Push je nanajvýš $2n^2 + 2nm(2n-1) \leq 4n^2m$

Analýza algoritmu - zložitosť

výsledná zložitosť závisí od poradia, v akom sa vykonávajú operácie PUSH a RELABEL a od zložitosti ich implementácie

Lema 43

Ak v každom kroku algoritmu vyberáme aktívny vrchol s maximálnou výškou, tak celkový počet nesaturujúcich PUSH *operácií je nanajvýš* $4n^3$.

Lema 44

Oprácie PUSH *a* RELABEL *sa dajú realizovať s konštantnou časovou zložitosťou*

Časová zložitosť algoritmu PUSH-RELABEL je $\mathcal{O}(n^2 m)$ resp. $\mathcal{O}(n^3)$ pri výbere aktívnych vrcholov s maximálnou výškou.

Toky v sietiach - modifikácie

antiparalelné hrany

- v praktických situáciach potrebujeme modelovať hranu z u do v ale aj hranu z v do u
- prevod na štandartnú úlohu pridaním nového vrcholu

viaczdrojové toky

- sieť obsahuje niekoľko zdrojov a niekoľko cieľov
- prevod na štandartnú úlohu pridaním nového zdrojového a nového cieľového vrcholu
- hrany vychádzajúce z nového zdroja (vstupujúce do nového cieľa) majú neobmedzenú kapacitu

toky s kapacitným ohraničením vrcholov

- prevod na štandartnú úlohu nahradením vrcholov novou dvojicou vrcholov
- hrana medzi novou dvojicou vrcholov nesie kapacitné ohraničenie pôvodného vrchola

Toky v sietiach - modifikácie

- siete s násobnými zdrojmi a cieľmi
- najlacnejší maximálny tok
- viacproduktové toky
- ...

Bipartitné grafy

- daný neorientovaný graf $G = 9V, H)$
- párovaním v grafe G je každá množina $M \subseteq H$ taká, že s každým vrcholom $v \in V$ inciduje maximálne jedna hrana z M
- problém maximálneho párovania je úloha nájsť pre daný graf párovanie maximálnej možnej kardinality
- budeme skúmať len bipartitné grafy, tj. grafy, ktorých množinu vrcholov je možné rozdeliť na dve množiny L a R tak, že $V = L \cup R$, $L \cap R = \emptyset$ a $H \subseteq L \times R$

problém maximálneho párovania v bipartitných grafoch prevedieme na problém maximálneho toku

Redukcia problému maximálneho párovania

- k danému bipartitnému grafu $G = (V, H)$ s rozdelením množiny vrcholov na L a R skonštruujeme sieť $G' = (V', H')$
- $V' = V \cup \{s, t\}$, pričom $s, t \notin V$
- $H' = \{(s, u) \mid u \in L\} \cup \{(u, v) \mid (u, v) \in H\} \cup \{(v, t) \mid v \in R\}$
- kapacita každej hrany v H' je 1

Redukcia problému maximálneho párovania

Lemma 45

Ak M je párovanie v G, tak existuje celočíselný tok f v G' s hodnotou $|f| = |M|$.
Naopak, ak f je celočíselný tok v G', tak v G existuje párovanie M kardinality $|M| = |f|$.

Dôsledok

Kardinalita maximálneho párovania M v bipartitnom grafe G je rovná hodnote maximálneho toku f v sieti G'

Fakt

Ak všetky kapacitné ohraničenia v sieti sú celočíselné, tak metóda Forda-Fulkersona nájde maximálny tok, ktorého hodnota je celočíselná a naviac aj tok na každej hrane je celočíselný.

Toky s požiadavkami

- s každým vrcholom v v sieti je asociovaná hodnota $d(v)$, tzv. požiadavka vrchola
- hodnota $d(v) > 0$ indikuje cieľový vrchol, do ktorého má byť prepravený tok veľkosti $d(v)$
- hodnota $d(v) < 0$ indikuje zdrojových vrchol, z ktorého má byť prepravený tok veľkosti $d(v)$
- hodnota hodnota $d(v) = 0$ indikuje vrchol, cez ktorý prechádza tok
- predpokladáme, že kapacity hrán ako aj požiadavky vrcholov sú celočíselné

Toky s požiadavkami

- tok v sieti G s požiadavkami $d : V \to \mathbb{Z}$ je funkcia $f : V \times V \to \mathbb{R}$ spĺňajúca nasledujúce dve podmienky

  kapacitné ohraničenie pre všetky $u, v \in V$

  $$0 \leq f(u,v) \leq c(u,v)$$

  podmienka kontinuity pre všetky $u \in V$ platí

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

  tok s uvedenými vlastnosťami sa nazýva prípustný tok
- problém prípustného toku je definovaný ako úloha nájsť prípustný tok v sieti s požiadavkami

Toky s požiadavkami - redukcia

- ak existuje prípustný tok v sieti s požiadavkami, tak $\sum_{v \in V} d(v) = 0$ a naviac $\sum_{v.d(v)>0} d(v) = \sum_{v.d(v)<0} d(v)$
- úlohu prevedieme na problém maximálneho toku
- sieť rozšírime o vrchol s^* (*super-zdroj*) a t^* (*super-cieľ*)
- pre každý vrchol $v \in V$ s $d(v) < 0$ pridáme do siete hranu (s^*, v) s kapacitou $-d(v)$
- pre každý vrchol $v \in V$ s $d(v) > 0$ pridáme do siete hranu (v, t^*) s kapacitou $d(v)$
- sieť s požiadavkami má prípustný tok práve ak maximálny tok v rozšírenej sieti má hodnotu

$$D = \sum_{v.d(v)>0} d(v)$$

Toky s požiadavkami a dolnými hranicami

- s každým vrcholom v v sieti je asociovaná požiadavka $d(v)$
- s každou hranou (u, v) v sieti je asociovaná kapacita $0 \leq c(u, v)$ a dolná hranica $l(u, v)$ taká, že $0 \leq l(u, v) \leq c(u, v)$
- tok v sieti G s požiadavkami d a dolnými hranicami l je funkcia $f : V \times V \rightarrow \mathbb{R}$ spĺňajúca nasledujúce dve podmienky

  kapacitné ohraničenie pre všetky $u, v \in V$

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

  podmienka kontinuity pre všetky $u \in V$ platí

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

  tok s uvedenými vlastnosťami sa nazýva prípustný tok

Toky s požiadavkami a dolnými hranicami - redukcia

- daná je sieť G s požiadavkami d a dolnými hranicami l
- predpokladajme, že iniciálny tok f_0 na hrane (u, v) siete G je $l(u, v)$
- iniciálny tok spĺňa kapacitné ohraničenia, ale môže porušovať podmienky kontinuity
- označme $L(u) = \sum_{v \in V} f_0(v, u) - \sum_{v \in V} f_0(u, v)$
- k vyrovnaniu disbalancie $L(u)$ má každá hrana incidujúca s vrcholom u rezervu $c(u, v) - l(u, v)$
- skonštruujeme sieť G', ktorá sa od siete G líši kapacitami hrán (kapacita hrany (u, v) v sieti G' je $c(u, v) - l(u, v)$) a požiadavkami vrcholov (požiadavok vrcholu je $L(u)$)
- hľadáme prípustný tok v sieti G' s požiadavkami d
- v sieti G existuje prípustný tok práve vtedy keď existuje prípustný tok v sieti G'

Algoritmy pre prácu s reťazcami

Algoritmy pre vyhľadávanie, porovnávanie a editáciu reťazcov

- vyhľadávanie vzorky v texte
- vzdialenosti reťazcov a transformácia reťazcov
- spoločná podpostupnosť
- aproximácia reťazcov
- opakujúce sa podreťazce

Vyhľadávanie vzorky v texte

- daný je text T a vzorka P (*pattern*) – reťazce nad abecedou Σ
- úlohou je vyhľadať všetky výskyty vzorky v texte
- text je daný ako pole $T[1..n]$, vzorka ako $P[1..m]$

- vzorka P sa vyskytuje v texte T s posunom s ak $0 \leq s \leq n - m$ a $T[s+1..s+m] = P[1..m]$
- číslo s uvedených vlastností sa nazýva platným posunom pre text T a vzorku P
- problém vyhľadávania vzorky je formulovaný ako úloha nájsť pre dané T a P všetky platné posuny

Algoritmy

Algoritmus	*Predspracovanie*	*Vyhľadávanie*
Úplné prehľadávanie	0	$\mathcal{O}((n-m+1)m)$
Karp-Rabin	$\Theta(m)$	$\mathcal{O}((n-m+1)m)$
Konečné automaty	$\mathcal{O}(m\|\Sigma\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$
Boyer-Moore	$\Theta(m+\|\Sigma\|)$	$\mathcal{O}((n-m+1)m)$

priemerná zložitosť algoritmov Karp-Rabin a Boyer-Moore je výrazne lepšia než uvedená zložitosť v najhoršom prípade

Úplné prehľadávanie

ÚPLNÉ_PREHĽADÁVANIE(T, P)
for $s = 0$ **to** $n - m$ **do**
    if $P[1..m] = T[s+1..s+m]$
        then print "s je platný posun **fi**
od

zložitosť
cyklus sa vykoná $n - m + 1$ krát
v každom cykle sa vykoná m porovnaní znakov
spolu $\mathcal{O}((n - m + 1)m)$

Karpov - Rabinov algoritmus

- predpokladajme, že $\Sigma = \{0, 1, \ldots, 9\}$
  (*zobecnenie pre ľubovoľnú abecedu je priamočiare*)
- každý reťazec nad abecedou Σ môžeme chápať ako číslo zapísané v desiatkovej sústave
- označme p číslo zodpovedjaúce reťazcu $P[1..n]$ a
  t_s čísla zodpovedajúce reťazcom $T[s+1..s+m]$
- problém overiť, či s je platným posunom sa redukuje na problém overiť, či $t_s = p$

predspracovanie

výpočet čísla p Hornerovou schémou (čas $\Theta(m)$)
výpočet čísla t_0 Hornerovou schémou (čas $\Theta(m)$)
$p = P[m] + 10(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1]) \ldots))$

výpočet čísel $t_1, \ldots, t_{n-m}$ (čas $\Theta(n-m)$)
$t_{s+1} = 10(t_s - 10^{m-1} T[s+1]) + T[s+m+1]$

Algoritmus Karp-Rabin so zvyškami

- algoritmus Karp-Rabin sa nedá použiť ak čísla p a t_s sú príliš veľké
- v takom prípade sa používa výpočet modulo q, kde typicky q je prvočíslo také, že $10q \approx$ počítačové slovo
- test $t_s = p$ sa nahradí testom $t_s \equiv p \pmod{q}$
- číslo s, pre ktoré platí rovnosť $t_s \equiv p \pmod{q}$ je len potenciálnym posunom, jeho platnosť sa musí overiť porovnaním príslušných reťazcov

- zložitosť predspracovania je nezmenená ($\Theta(m)$)
- zložitosť výpočtu je v najhoršom prípade *(tj. ak pre všetky s platí skúmaná rovnosť)* $\mathcal{O}((n-m+1)m)$
- zložitosť konkrétneho výpočtu je daná počtom platných posunov
- ak očakávaný počet platných posunov je c, tak očakávaná zložitosť algoritmu je $\mathcal{O}((n-m+1)+cm)$

Konečné automaty

- pre danú vzorku $P[1..m]$ skonštruujeme konečný automat $A = (\{0, \ldots, m\}, \Sigma, \delta, \{0\}, \{m\})$
- text $T[1..n]$ spracujeme automatom A

FINITE_AUTOMATON_MATCHER(T, A)
$q \leftarrow 0$
for $i = 1$ **to** n **do**
    $q \leftarrow \delta(q, T[i])$
    if $q = m$ **then** **print** "$i - m$ je platný posun" **fi**
od

zložitosť spracovania textu je $\Theta(n)$

Konštrukcia automatu pre danú vzorku P

- označenie $P_q = P[1] \cdots P[q]$, $T_q = T[1] \cdots [q]$
- sufixová funkcia $\sigma : \Sigma^* \rightarrow \{0, 1, \ldots, m\}$ kde

  $\sigma(x)$ je dĺžka najdlhšieho prefixu vzorky P, ktorý je sufixom slova x
- *príklad*: pre $P = popapopa$ je
  $\sigma(\varepsilon) = 0$, $\sigma(papap) = 1$, $\sigma(papop) = 3$, $\sigma(popapo) = 6$
- konečný automat pre vzorku P je

$$A = (\{0, \ldots, m\}, \Sigma, \delta, \{0\}, \{m\})$$

pričom prechodová funkcia δ je definovaná predpisom

$$\delta(q, a) \stackrel{\text{def}}{=} \sigma(P[1] \ldots P[q]\ a)$$

Konštrukcia automatu pre danú vzorku P

$\textsc{Automat}(P, \Sigma)$
for $q = 0$ **to** m **do**
    for každé $a \in \Sigma$ **do**
        $k \leftarrow \min(m+1, q+2)$
        repeat $k \leftarrow k - 1$
            until $P[1] \cdots P[k]$ je sufixom $P[1] \cdots P[q]$
        $\delta(q, a) \leftarrow k$
    od od
return δ

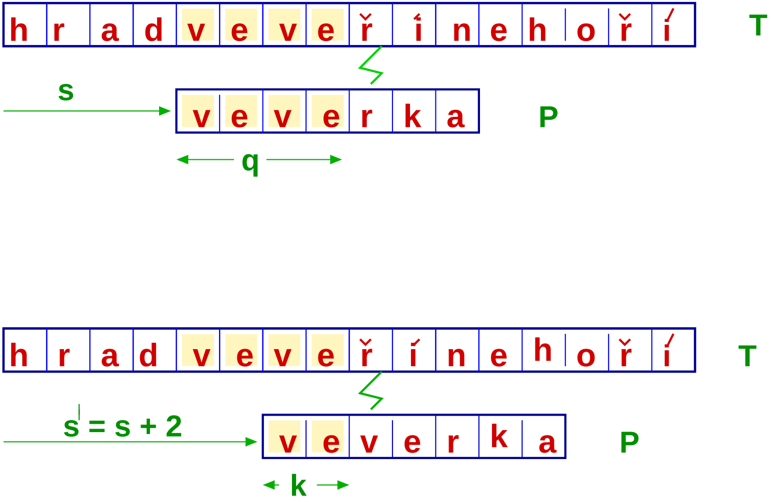
zložitosť konštrukcie automatu je $\mathcal{O}(m^3|\Sigma|)$

existuje efektívnejšia procedúra zložitosti $\mathcal{O}(m|\Sigma|)$

Algoritmus Knuth-Morris-Pratt

- vychádza z podobného princípu ako konečné automaty, ale nekonštruuje celý konečný automat
- namiesto konečného automatu sa pred samotným vyhľadávaním vypočíta v čase $\Theta(m)$ zo vzorky tzv. prefixová funkcia
- samotné vyhľadávanie vzorky sa realizuje v čase $\Theta(n)$

Príklad


h r a d v e v e ř í n e h o ř í
T
s
v e v e r k a
P
q
h r a d v e v e ř í n e h o ř í
T
s' = s + 2
v e v e r k a
P
k

Algoritmus Knuth-Morris-Pratt

- pre prefix $P[1 \dots q]$ vzorky zhodný s textom $T[s+1 \dots s+q]$ testujeme, aké je najväčšie $s' > s$ pre ktoré

$$P[1 \dots k] = T[s'+1 \dots s'+k],$$

  kde $s' + k = s + q$
- k výpočtu s' nepotrebujeme poznať text, pretože $T[s'+1 \dots s'+k]$ je prefixom vzorky
- pre vzorku $P[1 \dots m]$ definujeme prefixovú funkciu $\pi : \{1, 2, \dots, m\} \rightarrow \{0, 1, \dots m-1\}$ predpisom

$$\pi[q] \stackrel{\text{def}}{=} \max\{k \mid k < q \text{ a } P_k \text{ je sufixom } P_q\}$$

Algoritmus Knuth-Morris-Pratt

- počítame $\pi[q]$ za predpokladu, že poznáme $\pi[1], \ldots, \pi[q-1]$
- nech $\pi[q-1] = k$, tj. $P[1 \ldots k] = P[\star \ldots q-1]$
- ak $P[k+1] = P[q]$ tak $P[1 \ldots k+1] = P[\star \ldots q]$ a $\pi[q] = k+1$
- ak $P[k+1] \neq P[q]$ tak hľadáme vlastný prefix $\triangle$ reťazca P_k taký, že $\triangle$ je sufixom P_k a teda aj sufixom P_{q-1}
- kandidátom je prefix reťazca P_k dĺžky $\pi[k]$
  - $P[1 \ldots \pi[k]] = P[\star \ldots k]$ (definícia $\pi[k]$)
  - $P[\star \ldots k] = P[\star \ldots q-1]$ (pretože $\pi[q-1] = k$)
- ak $P[\pi[k]+1] = P[q]$, tak $\pi[q] = \pi[k]+1$
- v opačnom prípade je ďalším kandidátom vlastný prefix $\triangle$, ktorý je zároveň sufixom P_k a teda aj sufixom P_{q-1}
- hľadaným kandidátom je prefix reťazca P_k dĺžky $\pi[\pi[k]]$
- ďalej postupujeme analogicky

Algoritmus Knuth-Morris-Pratt

KMP(T, P)
$\pi \leftarrow$ PREFIXOVÁ_FUNKCIA(P); $q \leftarrow 0$
for $i = 1$ **to** n **do**
    while $q > 0 \wedge P[q+1] \neq T[i]$ **do** $q \leftarrow \pi[q]$ **od**
    if $P[q+1] = T[i]$ **then** $q \leftarrow q + 1$ **fi**
    if $q = m$ **then** $i - m$ *je platný posun* ; $q \leftarrow \pi[q]$ **fi**
od

PREFIXOVÁ_FUNKCIA(P)
$\pi[1] \leftarrow 0$; $k \leftarrow 0$
for $q \leftarrow 2$ **to** m **do**
    while $k > 0 \wedge P[k+1] \neq P[q]$ **do**
        $k \leftarrow \pi[k]$ **od**
    if $P[k+1] = P[q]$ **then** $k \leftarrow k + 1$ **fi**
    $\pi[q] \leftarrow k$
od
return π

Časová zložitosť algoritmu Knuth-Morris-Pratt

- pre analýzu zložitosti výpočtu prefixovej funkcie použijeme metódu účtov
- každý **for** cyklus (riadky 3 až 8) dostane 3 kredity
- 2 kredity sa použijú na zaplatenie priraďovacích príkazov z riadk. 6 a 8
- 1 kredit sa uloží na účet
- priraďovací príkaz v tele **while** cyklu sa zaplatí z účtu

Invariant: počet kreditov na účte neklesne pod 0

- každý ukončený **for** cyklus uloží na účet 1 kredit a prípadne zvýši hodotu premennej k o 1
- priraďovací príkaz v tele **while** cyklu postupne znižuje hodnotu k; s každým znížením sa zníži počet kreditov na účte o 1
- hodnota premennej k neprevyšuje počet kreditov na účte

zložitosť výpočtu prefixovej funkcie je $\Theta(m)$

Časová zložitosť algoritmu Knuth-Morris-Pratt

- pre analýzu zložitosti výpočtu algoritmu KMP použijeme metódu účtov
- analýza je analogická analýze výpočtu prefixovej funkcie
- každý **for** cyklus dostane 3 kredity
- 1 kredit sa uloží na účet
- priraďovací príkaz v tele **while** cyklu sa zaplatí z účtu
- 2 kredity sa použijú na zaplatenie priraďovacích príkazov
- počas celého výpočtu počet kreditov na účte neklesne pod 0
- platnosť invariantu vyplýva z totožných argumentov; úlohu premennej k má premenná q

zložitosť výpočtu algoritmu KMP je $\Theta(n)$