Návrh algoritmů II

Ivana Černá
FI MU Brno
cerna@fi.muni.cz

September 4, 2006

Obsah prednášky

- Zložitosť, analýza zložitosti problémov a algoritmov
- Dátové štruktúry
  - Haldy: binomiálna, Fibonacciho
  - Reprezentácia množín
- Techniky návrhu efektívnych algoritmov
  - rozdeľ a panuj
  - dynamické programovanie
  - hladové heuristiky
  - branch and bound
- Konkrétne algoritmy
  - Grafové algoritmy
  - Algoritmy pre prácu s reťazcami

Analýza zložitosti

- Výpočtový model, zložitostné kritérium. Zložitosť ako funkcia dĺžky vstupu.
- Asymptotická zložitosť algoritmu
- Typy zložitosti
  - zložitosť v najlepšom prípade
  - zložitosť v najhoršom prípade
  - priemerná zložitosť, vážený priemer
- Zložitosť problému vs. zložitosť algoritmu.

Analýza zložitosti problému

- Dolný odhad zložitosti problému
- Horný odhad zložitosti problému — zložitosť konkrétneho algoritmu pre problém
- Zložitosť problému

Dolný odhad zložitosti problému - techniky

Informačná metóda riešenie problému v sebe obsahuje isté množstvo informácie a v každom kroku výpočtu sme schopní určiť len časť tejto informácie (*násobenie matíc, cesta v grafe, triedenie*)

Redukcia

Metóda sporu *Varianta A:* za predpokladu, že algoritmus má zložitosť menšiu než uvažovanú hranicu, vieme skonštruovať vstup, na ktorom nedá 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 vzájomne rôznych celých čísel $x = (x_1, \ldots, x_n)$ a číslo $i \in \mathbb{N}$

Úloha nájsť i-ty najmenší prvok postupnosti, tj. číslo k t.ž.

$$\left|\{x_j \mid x_j \leq x_k\}\right| = i$$

Ak $i = \lceil \frac{n}{2} \rceil$ hovoríme o *mediáne*.

Dolný odhad

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

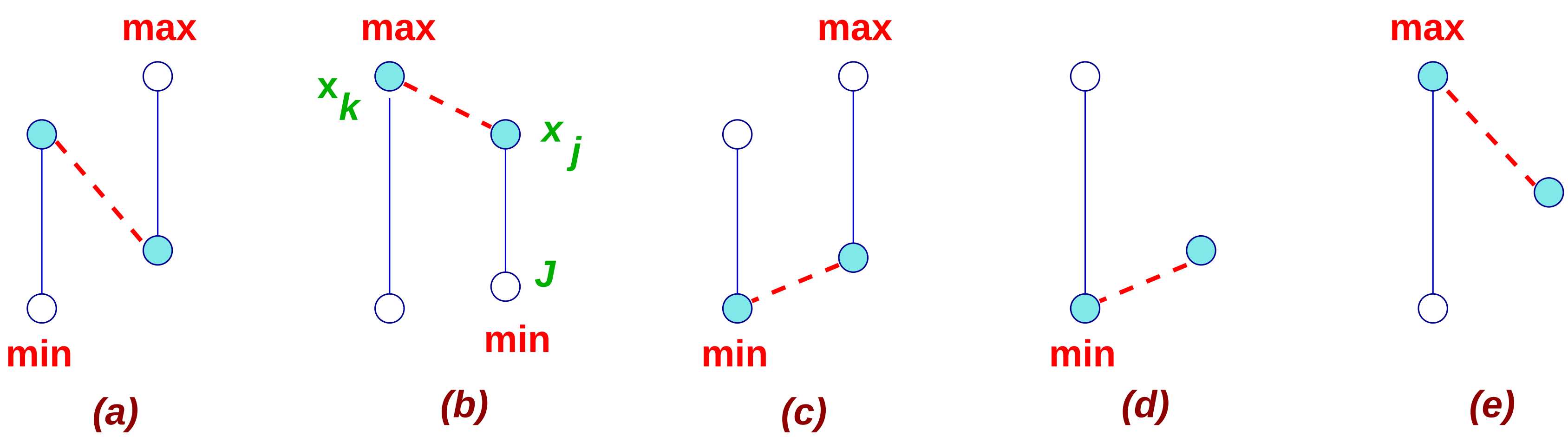
Dôkaz. (Varianta A)
Nech $x = (x_1, \ldots, x_n)$ je vstup dĺžky n, na ktorom $\mathcal{A}$ vykoná menej než $n-1$ porovnaní a nech x_r je maximálny prvok v x.
Potom v x musí existovať prvok x_p taký, že $p \neq r$ a v priebehu výpočtu x_p nebol porovnávaný so žiadnym prvkom väčším než on sám. Existencia 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.
$\square$

Veta. *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,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 `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.

$$M(n,i) \quad \geq 3 + M(n-2, i-1) \qquad (1)$$

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 $i - 1 \leq \lceil n/2 \rceil - 1 \leq \lceil (n-1)/2 \rceil$

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

□

Algoritmus Select pre problém i-teho prvku

Vstupom pre algoritmus $\textsc{Select}$ je postupnosť $x = (x_1, \ldots, x_n)$ vzájomne rôznych čísel a číslo $i \in \mathbb{N}$. Algoritmus nájde i-ty prvok postupnosti x.

Zložitosť algoritmu je *lineárna* voči dĺžke postupnosti.

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 $\textsc{Select}$ 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)

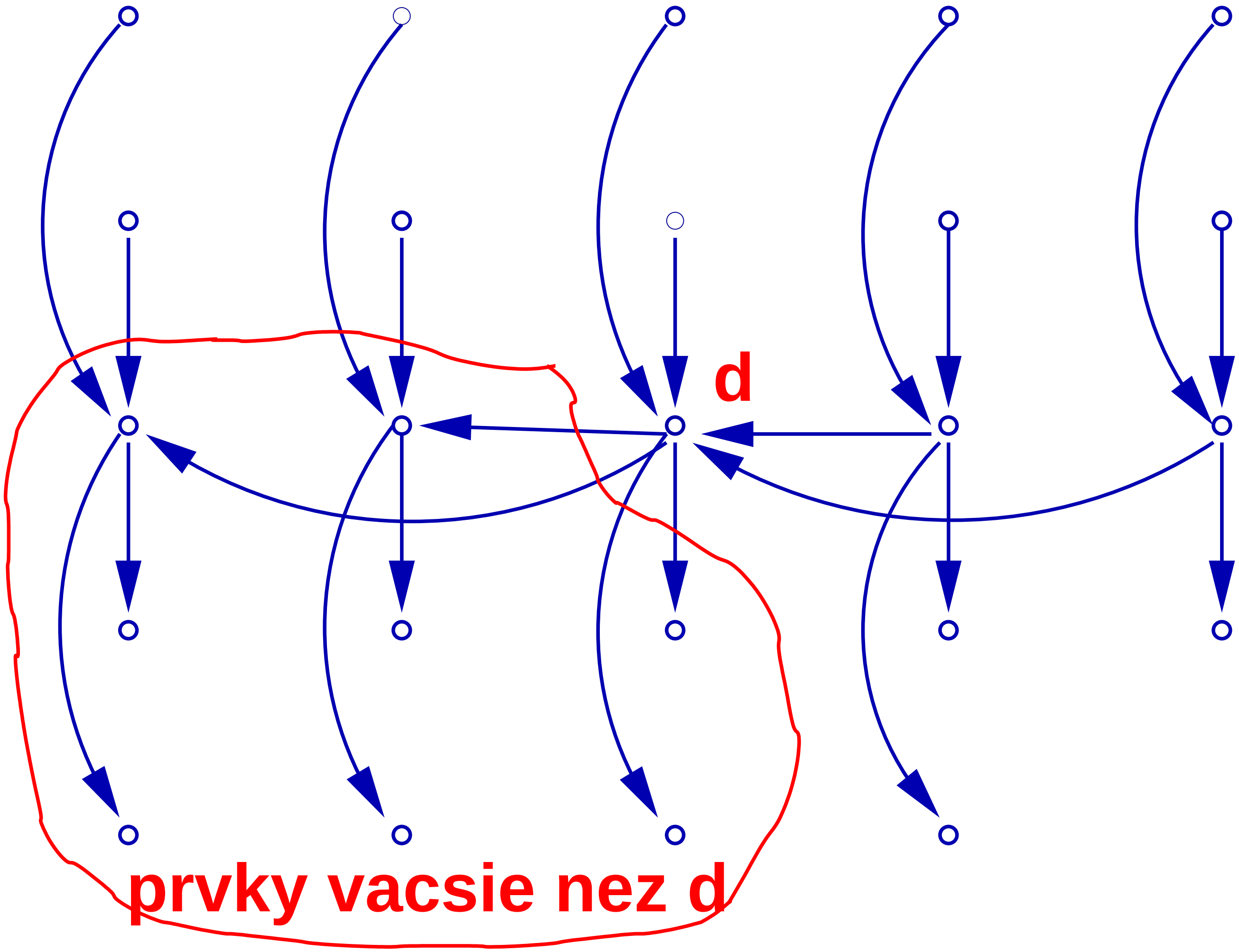
4. prvky postupnosti x (okrem d) rozdeľ do 2 skupín
   skupina 1: prvky menšie než d (označme ich počet m)
   skupina 2: prvky väčšie než d (ich počet je $n-m-1$)

5. ak $m+1=i$ tak d je hľadaný i-ty prvok postupnosti x
   ak $i \leq m$ tak rekurzívne volaj $\textsc{Select}$ pre skupinu 1 a číslo i
   ak $i > m+1$ tak rekurzívne volaj $\textsc{Select}$ pre skupinu 2 a číslo $i+m-1$

Analýza zložitosti

1. $\mathcal{O}(n)$
2. $\mathcal{O}(n)$ (medián 5-tich prvkov nájdeme v konštantnom čase)
3. $T(\lceil\frac{n}{5}\rceil)$ ($T(k)$ je zložitosť $\textsc{Select}$ pre postupnosť k prvkov)
4. $\mathcal{O}(n)$
5. závisí od počtu prvkov menších resp. vắčších než d
   buď $T(m)$ alebo $T(n-m-1)$.

Odhad počtu prvkov menších a väčších než medián d



Počet prvkov väčších než d — 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

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

Následne **počet prvkov menších než d je maximálne** $\frac{7n}{10} + 6$.

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.

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

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 $T(n) \leq cn$.

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

Príklady

Zásobník

Operácie $\textsc{Push}(S,x)$, $\textsc{Pop}(S)$, $\textsc{Multipop}(S,k)$ (vyberie k prvkov z S resp. vyprázdni zásobník ak $\mid S \mid< k$).

Uvažujme postupnosť n operácií, každá operácia je $\textsc{Pop}$, $\textsc{Push}$ alebo $\textsc{Multipop}$.

$\textsc{Push}$ a $\textsc{Pop}$ majú zložitosť 1. V postupnosti n operácií má $\textsc{Multipop}$ v najhoršom prípade zložitosť n. Preto celá postupnosť má v najhoršom prípade zložitosť n^2.

Binárne počítadlo

k-bitové počítadlo implementované ako pole $A[0 \ldots k-1]$. Hodnota počítadla je $x = \sum_{i=0}^{k-1} A[i]2^i$. Iniciálna hodnota počítadla je 0.

$\textsc{Inc}(A)$
$i \leftarrow 0$
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 ($n < 2^{k+1}$) operácií $\textsc{Inc}$ je $n \cdot k$.

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 má v najhoršom prípade zložitosť $2n$.

Binárne počítadlo

Skupina 1 — preklopenie $A[0]$, pri každom volaní $\textsc{Inc}$. Celková zložitosť je n.

Skupina 2 — preklopenie $A[1]$, pri každom druhom volaní $\textsc{Inc}$. Celková zložitosť je $\lfloor \frac{n}{2} \rfloor$.

Skupina 3 — preklopenie $A[2]$, pri každom štvrtom volaní $\textsc{Inc}$. Celková zložitosť je $\lfloor \frac{n}{4} \rfloor$.

. . .

Počet skupín je $k - 1$

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

Metóda účtov

Každej operácii priradíme kredit (číslo), ktoré môže byť rôzne od jej skutočnej ceny (zložitosti). Pri realizácii operácie zaplatíme jej cenu kreditmi podľa nasledovných pravidiel:

– ak *cena operácie* $\leq$ *kredit operácie*, tak za operáciu zaplatíme toľko kreditov, aká je jej cena, a zvyšné kredity uložíme na účet,

– ak *cena operácie* $>$ *kredit operácie*, tak kredity potrebné na zaplatenie operácie vezmeme z účtu.

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.

Zásobník

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

v okamihu, keď objekt vkladáme do zásobníka, 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 postunosti n operácií je $\leq$ súčet kreditov vykonaných operácií $\leq 2n$.

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, 1 kredit dáme na účet premennej $A[i]$. Nastavenie 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 $\textsc{Inc}$ sa mení hodnota premennej na 1 maximálne raz. Preto amortizovaná cena operácie $\textsc{Inc}$ je 2 a cena zložitosti postupnosti n operácií $\textsc{Inc}$ 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 je

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

Za predpokladu $\Phi(D_n) \geq \Phi(D_0)$ platí $\boxed{\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i}$ 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 každdú hodnotu D dátovej štruktúry platilo, že jej potenciál $\Phi(D)$ je aspoň tak veľký, ako potenciál počiatočnej hodnoty D_0 dátovej štruktúry.

Zásobník

Dátová štruktúre je zásobník.

Potenciálová funkcia = počet prvkov v zásobníku.

Amortizovaná cena i-tej operácie - rozlíšime podľa typu operácie

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

Pre všetky $1 \le i \le n$ platí $\Phi(D_i) \ge \Phi(D_0)$.

Preto zložitosť celej postupnosti je $\sum_{i=1}^n c_i \le \sum_{i=1}^n \hat{c}_i \le 2n$

Binárne počítadlo

Dátová štruktúra je pole $A[0 \dots k-1]$, potenciálová funkcia je počet 1 v poli. Označme b_i počet 1 v počítadle $A[0 \dots k-1]$ po vykonaní i-tej operácie $\textsc{Inc}$.

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

Do tabuľky ukladáme položky, 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. Podobne, ak po odobraní položiek je tabuľka nenaplnená, alokujeme novú, menšiu tabuľku a položky do nej presunieme.

Podporované operácie sú $\textsc{Table Insert}$, $\textsc{Table Delete}$. 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.

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 tabuľku do nej presunieme a vložíme novú položku. Naopak, z tabuľky, ktorej faktor je $> 1/2$ môžeme položky odoberať. Ak je faktor 1/2, tak alokujeme novú tabuľku polovičnej veľkosti, prvky starej tabuľky do nej presunieme a položku odoberieme.

Cena jednej operácie $\textsc{Table Insert}$ alebo $\textsc{Table Delete}$ je v najhoršom prípade rovná počtu prvkov, ktoré presúvame do novej tabuľky. Cena postupnosti n operácií typu $\textsc{Table Insert}$ a $\textsc{Table Delete}$ je preto v najhoršom prípade $\mathcal{O}(n^2)$.

Analyzujme amortizovanú zložitosť postupnosti n operácií $\textsc{Table Insert}$.

Metódou 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. Druhú skupinu tvoria operácie, pri ktorých sa alokuje nová tabuľka.

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

Metódou kreditov

Každej operácii priradíme 3 kredity. Jeden kredit zaplatí vloženie položky do tabuľky, 1 kredit dostane na svoj účet položka, ktorú sme práve vložili do tabuľky a 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.

Metódou potenciálovej funkcie

Označme $num[T]$ počet položiek tabuľky a $size[T]$ veľkosť tabuľky. Definujeme 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

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

Zložitosť postupnosti n operácií typu TABLE INSERT a TABLE DELETE je $\Theta(n^2)$. Vysvetlite prečo.

Navrhnite lepšiu stratégiu tak, aby amortizovaná cena postupnosti n operácií typu TABLE INSERT a TABLE DELETE bola lineárna.

Dátové štruktúry

Reprezentácia dynamickej množiny objektov spolu s operáciami nad touto množinou objektov.

- 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 stromy: AVL, bielo-čierne stromy, B-stromy, 2-3 stromy

- haldy (Insert, Delete, Minimum)
  - (binárna) halda
  - binomiálna halda
  - Fibonacciho halda
- reprezentácia dijunktných množín (Insert, Find, UNION)
  - spájané zoznamy
  - UNION-FIND štruktúra

Haldy

Dátové štruktúry pre reprezentáciu množiny prvkov, nad ktorými je definované usporiadanie. 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

$\textsc{Decrease Key}(H, x, k)$ – zníži hodnotu vrcholu x na k

Halda – strom, každý vrchol stromu obsahuje kľúč = prvok reprezentovanej množiny. Pre každý podstrom haldy platí, že kľúč koreňa je menší alebo rovný než kľúče všetkých jeho následníkov.

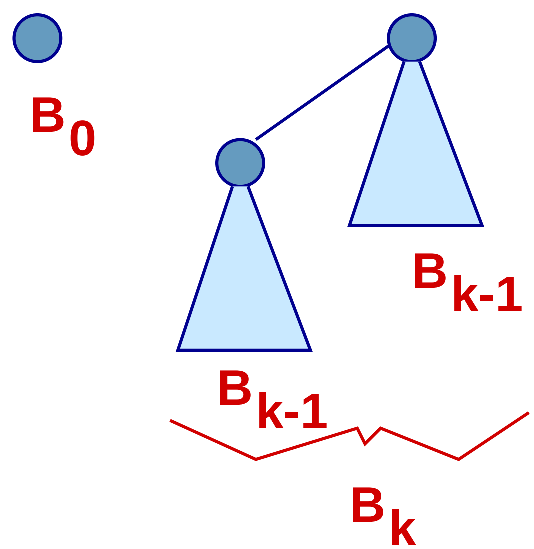
Operácia	Binárna halda	Binomiálna halda	Fibonacciho halda
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$
INSERT	$\Theta(\log n)$	$\Theta(1)$ *	$\Theta(1)$
UNION	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log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

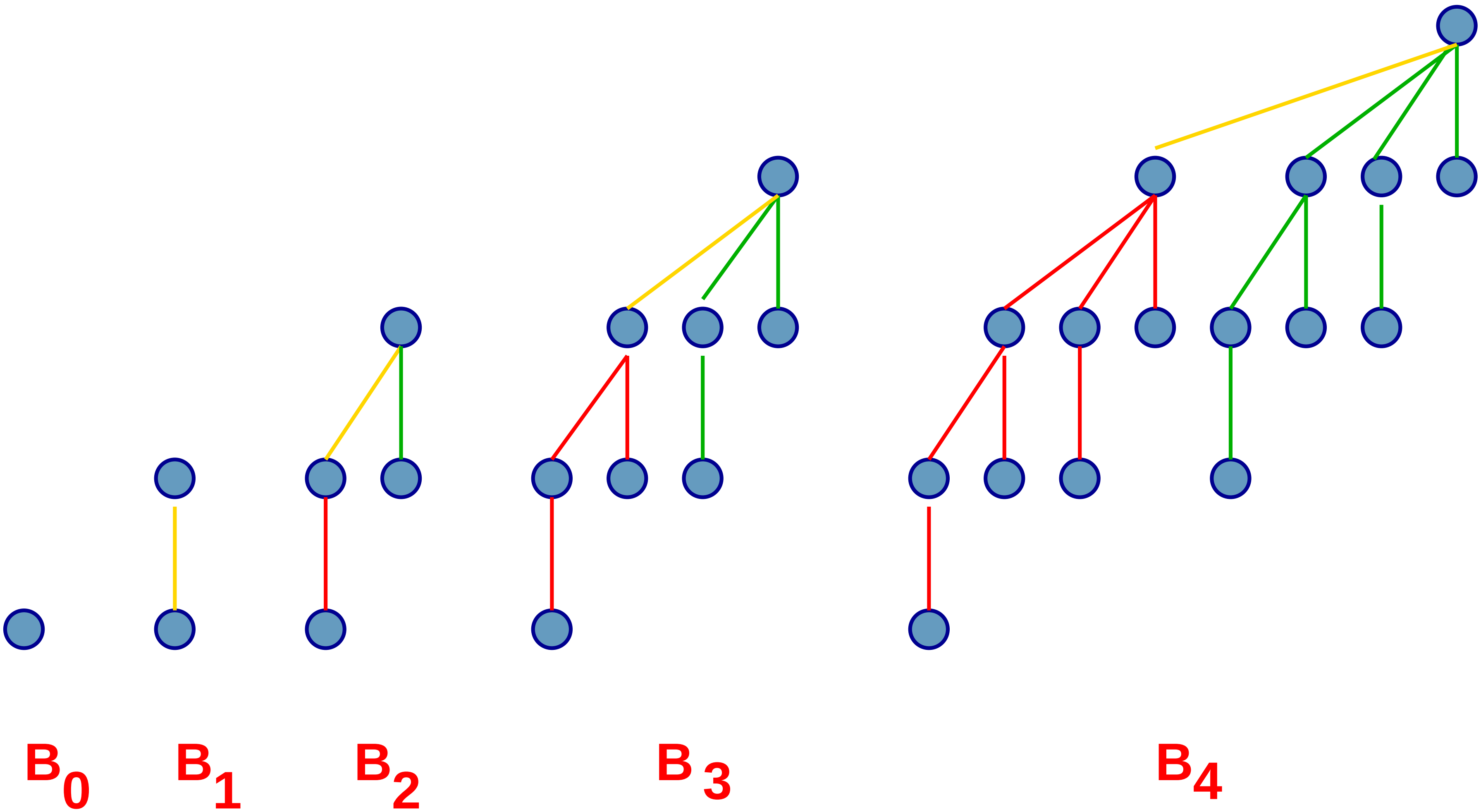
* amortizovaná zložitosť

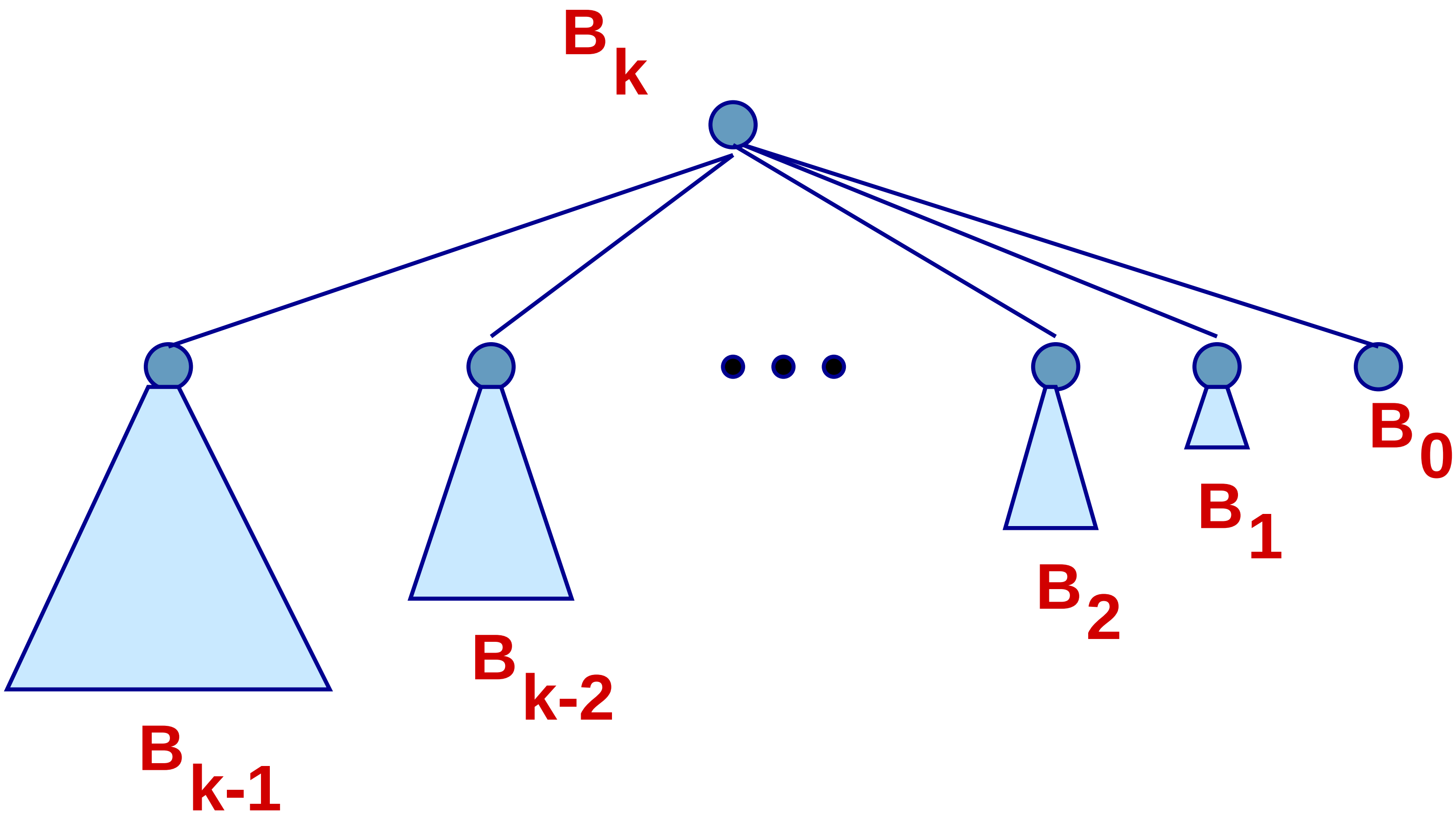
Binomiálna halda

Binomiálny strom B_k – je definovaný rekurzívne:

- binomiálny strom B_0 obsahuje jediný vrchol,
- binomiálny strom B_k pozostáva z dvoch binomiálnych stromov B_{k-1}, ktoré sú spojené tak, že koreň jedného z nich je najľavejším synom koreňa druhého z nich.




B_0
B_1
B_2
B_3
B_4


Bk
Bk-1
Bk-2
B2
B1
B0

Vlastnosti binomiálneho stromu B_k

- B_k má 2^k vrcholov, jeho hĺbka je k
- počet vrcholov hĺbky i je $\binom{k}{i}$
- koreň stromu B_k má stupeň k
- synovia koreňa zľava doprava sú korene stromov $B_{k-1}, \ldots B_0$

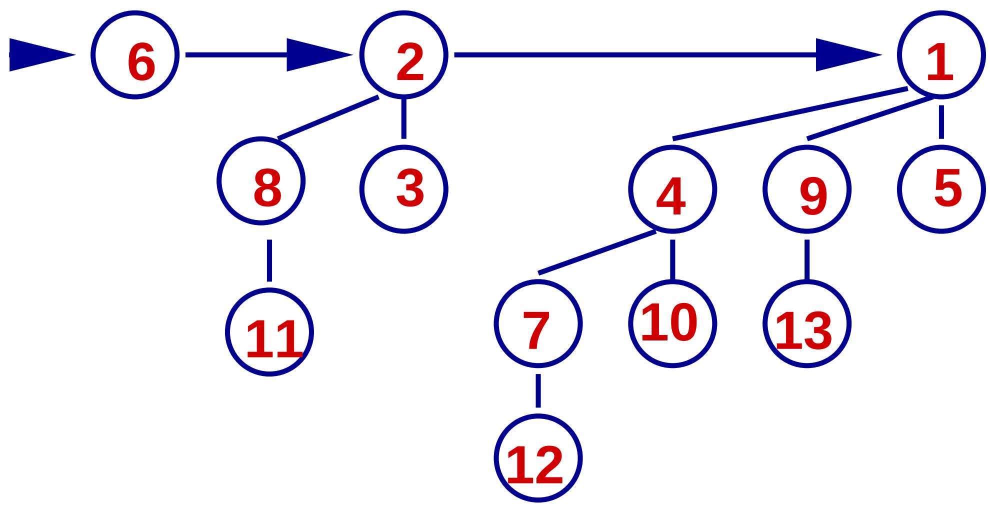
Dôkaz: indukciou vzhľadom k hodnote k.

Stupeň vrchola je počet jeho synov. Stupeň stromu je stupeň jeho koreňa.

Binomiálna halda

Binomiálna halda je les H binomiálnych stromov, kde

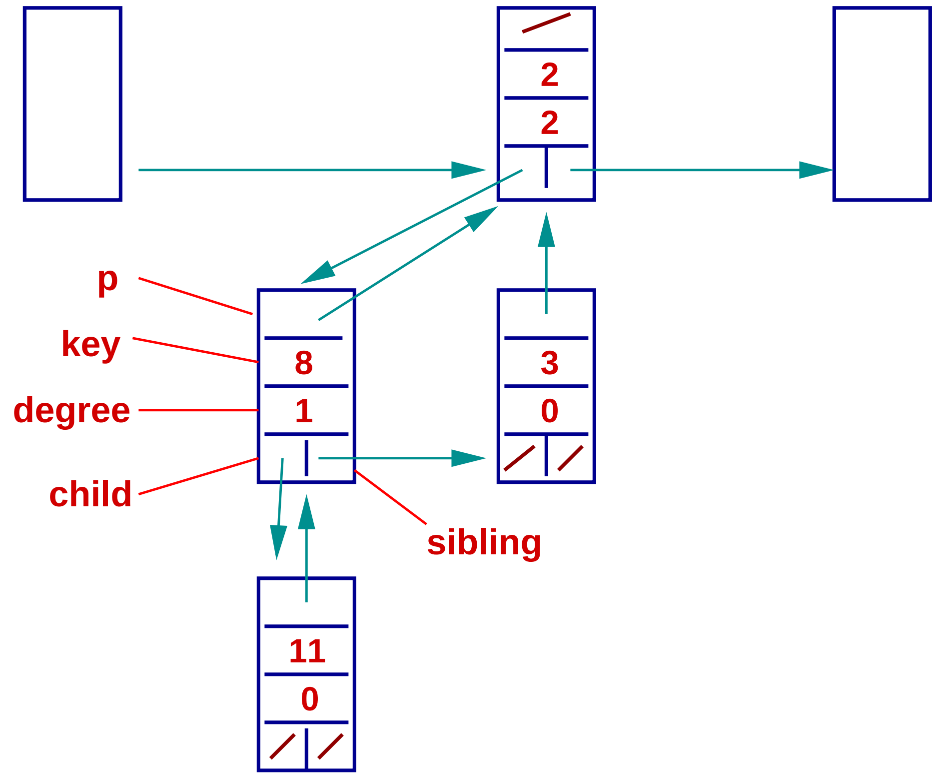
- každý binomiálny strom v H má vlastnosť haldy tj. prvok uložený vo vrchole je väčší alebo rovný prvku, ktorý je uložený v jeho otcovi,
- pre ľubovoľné k existuje v H maximálne jeden strom stupňa k
- korene stromov tvoria zoznam usporiadaný vzostupne podľa stupňa koreňov



Vlastnosti binomiálnej haldy

- halda s n vrcholmi má maximálne $\lceil \log n \rceil$ stromov
- schematicky môžeme haldu o n prvkoch vyjadriť pomocou binárneho zápisu čísla n

Operácie nad binomiálnou haldou



Vrchol stromu

je záznam s údajmi

p – ukazateľ na otca

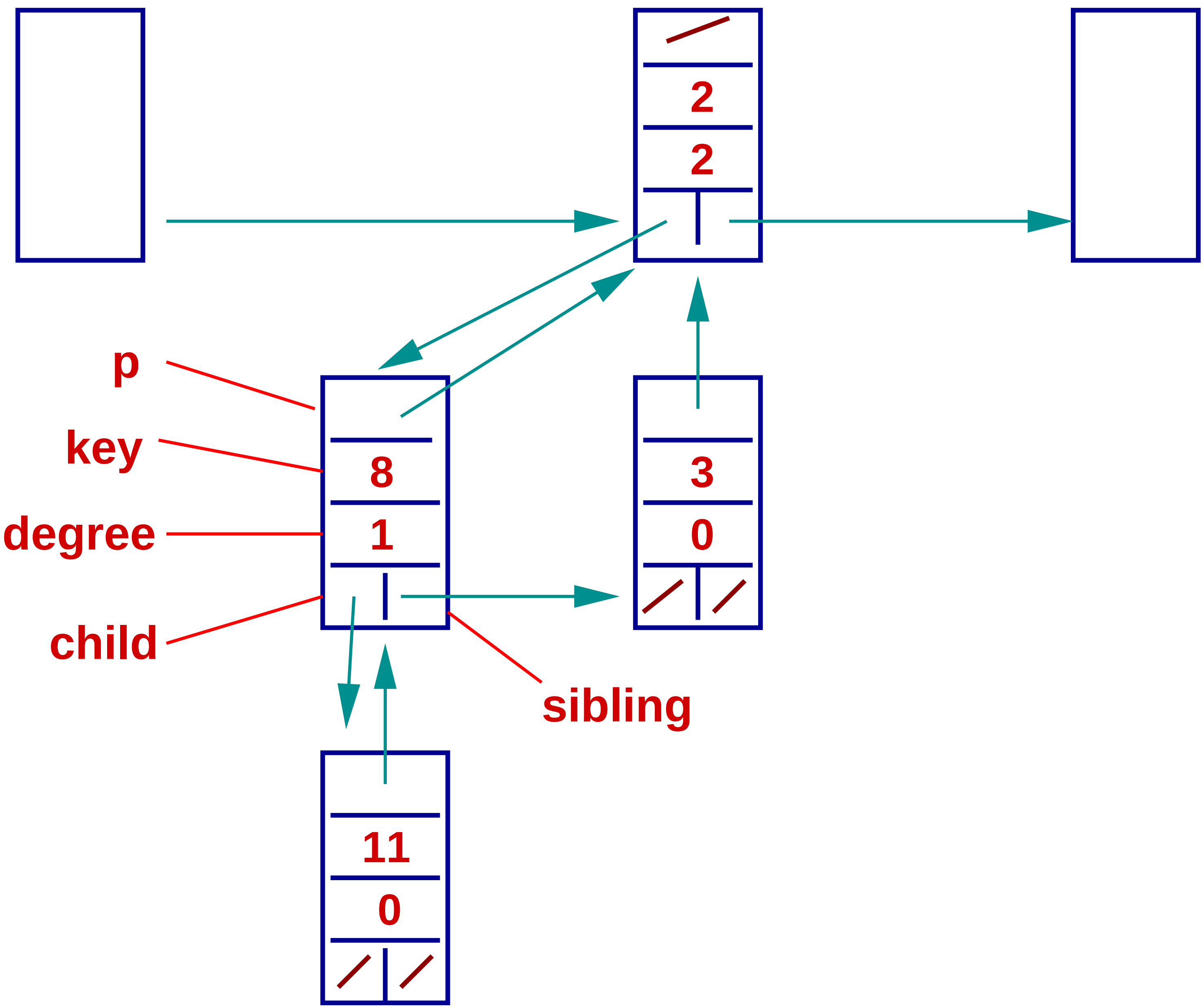
key – kľúč

$degree$ – počet synov

$sibling$ –
ukazateľ na pravého brata

$child$ –
ukazateľ na najľavejšieho syna

Premenná $head[H]$ – ukazateľ na strom s minimálnym stupňom


2
2
p
key
8
degree
1
child
3
0
sibling
11
0

Nájdenie minimálneho prvku haldy

Minimálny prvok každého stromu je uložený v koreni. Preto stačí prejsť koreňový zoznam. Zložitosť $\mathcal{O}(\log n)$

$\text{BH\_}\textsc{Minimum}(H)$

$y \leftarrow Nil$

$x \leftarrow head[H]$

$min \leftarrow \infty$

while $x \neq Nil$ **do**

  if $key[x] < min$ **then** $min \leftarrow key[x];\ y \leftarrow x$ **fi**

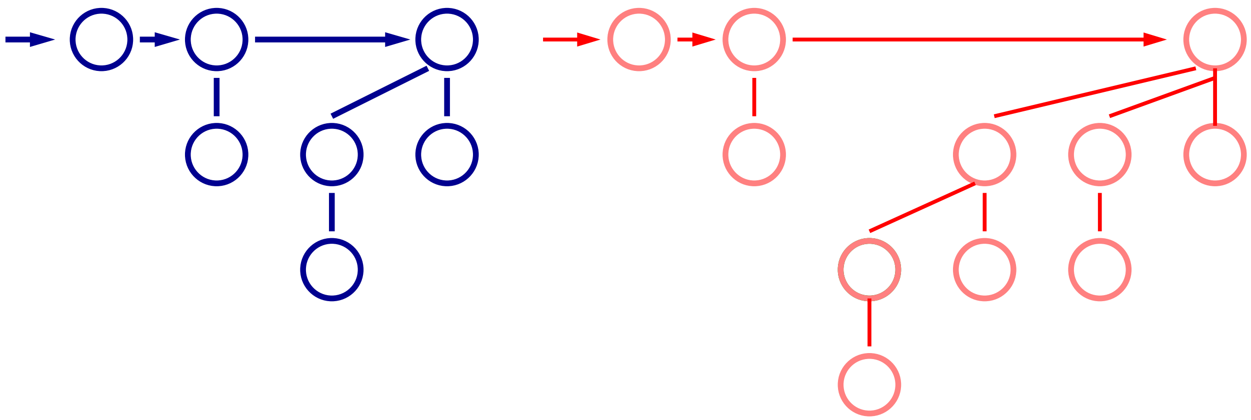
  $x \leftarrow sibling[x]$

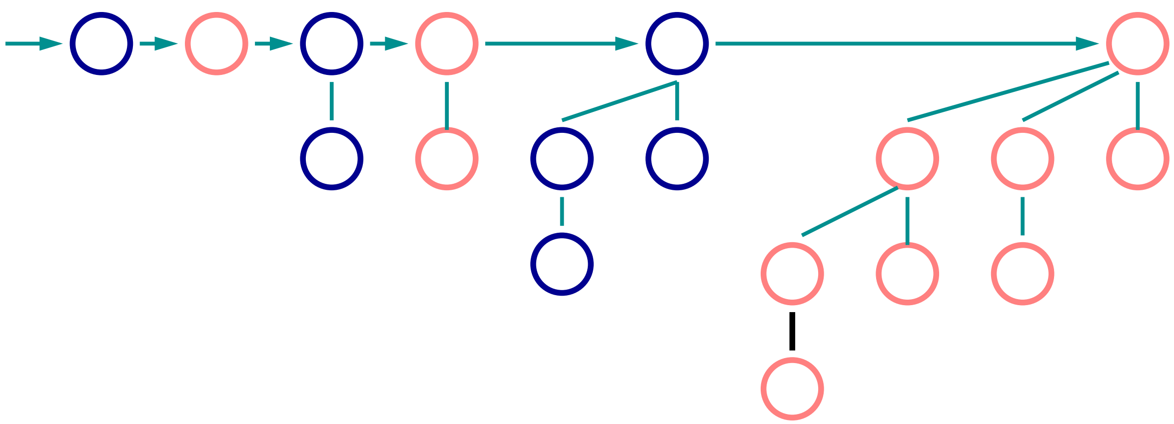
od

return y

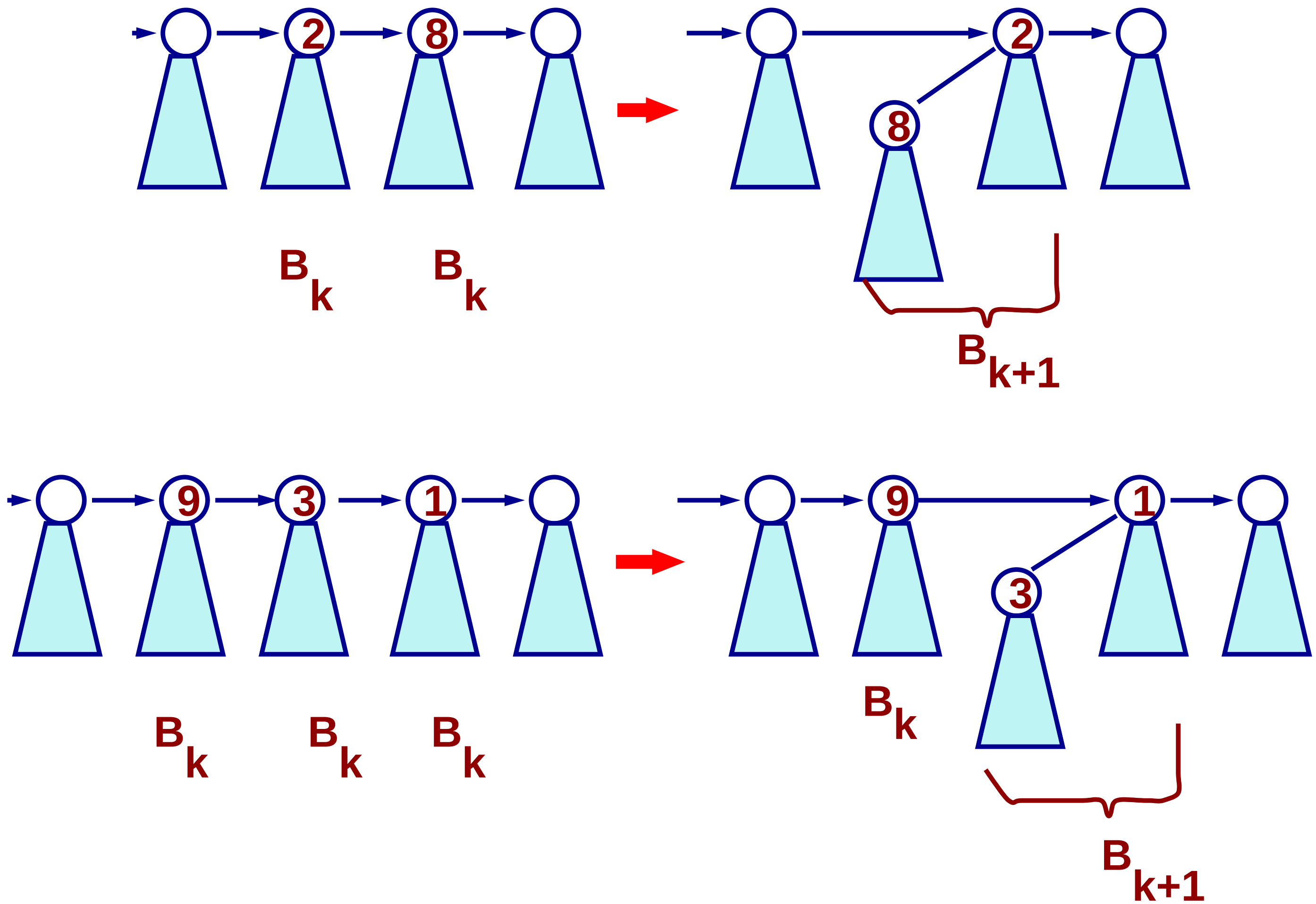
Zjednotenie dvoch háld

1. Koreňové zoznamy háld H_1 a H_2 spojíme tak, že stupne koreňov tvoria neklesajúcu postupnosť. Zložitosť $\mathcal{O}(\log n_1 + \log n_2)$





2. Prechádzame vytvorený koreňový zoznam. Ak narazíme na dva stromy s rovnakým stupňom koreňa, spojíme ich (*analógia sčitovania binárnych čísel*). Zložitosť $\mathcal{O}(\log n)$



$\textsc{BH\_Union}(H_1, H_2)$

$H \leftarrow \textsc{Make\_BH\_Heap}()$
$head[H] \leftarrow \textsc{BH\_Heap\_Merge}(H_1, H_2)$
if $head[H] = Nil$ **then return** H **fi**
$prev\_x \leftarrow Nil$
$x \leftarrow head[H]$
$next\_x \leftarrow sibling[x]$
while $next\_x \neq Nil$ **do**
    if $(degree[x] \neq degree[next\_x]) \lor$
        $(sibling[next\_x] \neq Nil \land$
            $degree[sibling[next\_x]] = degree[x])$
        then $prev\_x \leftarrow x;\ x \leftarrow next\_x$
        else if $key[x] \leq key[next\_x]$
            then $sibling[x] \leftarrow sibling[next\_x]$
                $\textsc{BH\_Link}(next\_x, x)$

```
                else if prev_x = Nil
                     then head[H] ← next_x
                     else sibling[prev_x] ← next_x fi
                BH_Link(x, next_x)
                x ← next_x fi
    fi
    next_x ← sibling[x]
od
return H
```

$\text{BH\_L}\textsc{ink}(y, z)$

```
p[y] ← z
sibling[y] ← child[z]
child[z] ← y
degree[z] ← degree[z] + 1
```

$\text{BH\_HEAP\_MERGE}(H_1, H_2)$

spojí zoznam koreňov haldy H_1 a haldy H_2 do jeného zoznamu tak, že korene sú usporiadané podľa stupňa (ako v algoritme triedenia spájaním, Mergesort)

Vloženie nového prvku do haldy

Operácia BH_INSERT pripojí k halde H nový vrchol x. Predpokladáme, že $key[x]$ je vkladaný prvok a že pre ostatné parametre platí $p[x] = Nil$, $child[x] = Nil$, $sibling[x] = Nil$, a $degree[x] = 0$.
Zložitosť $\mathcal{O}(1) + \mathcal{O}(\log n)$

$\text{BH\_INSERT}(H, x)$

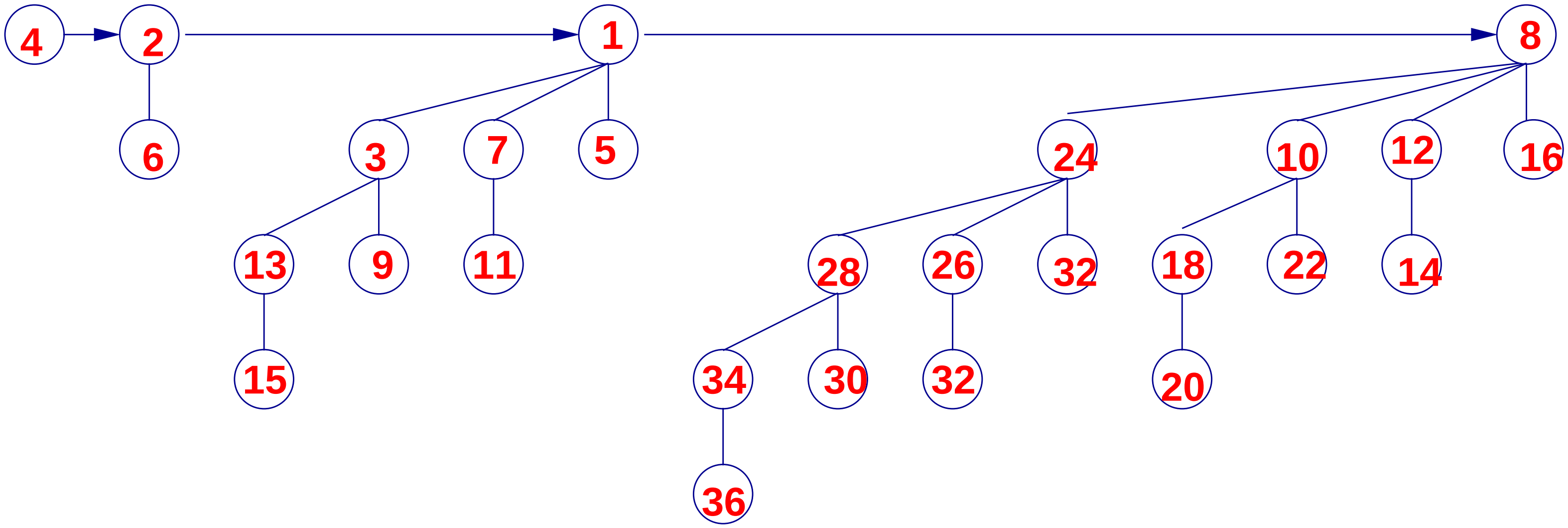
$H' \leftarrow \text{MAKE\_BH\_HEAP}()$
$head[H'] \leftarrow x$
$H \leftarrow \text{BH\_UNION}(H, H')$

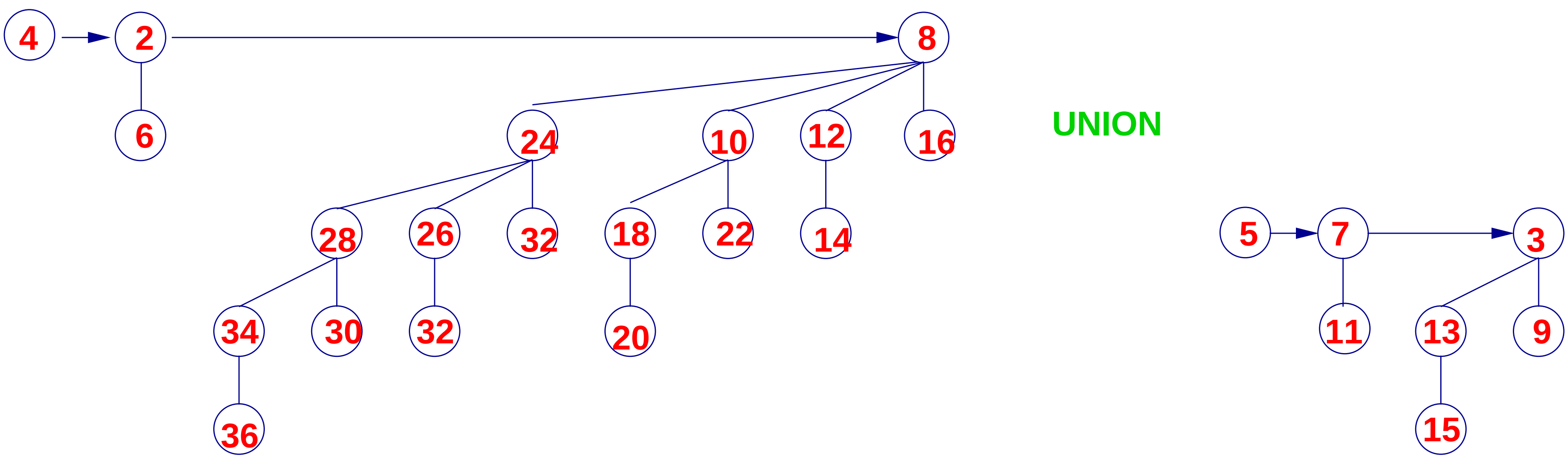
Odstránenie minimálneho prvku z haldy

Nájdeme koreň k s minimálnym prvkom. Odstránime k z koreňového zoznamu. Z detí koreňa k vytvoríme novú haldu a pripojíme ju k pôvodnej halde. Zložitosť $\mathcal{O}(\log n)$

BH_Extract_Min(H)

1 nech x je koreň s minimálnym kľúčom; odstráň x z H
2 $H' \leftarrow$ Make_BH_Heap()
3 obrát́ poradie, v ktorom sú spojené deti vrchola x
4 nech $head[H']$ukazuje na tento zoznam
5 $H \leftarrow$ BH_Union(H, H')




UNION

Zníženie hodnoty kľúča

Parametrom operácie je vrchol x a číslo k. Operácia zmení hodnotu kľúča uloženého v x na k za predpokladu, že k je menšie než pôvodný kľúč uložený v x. BH_DECREASE_KEY zníži hodnotu kľúča a vrchol presúva smerom nahor (ako v binárnej halde). Zložitosť $\mathcal{O}(\log n)$.

BH_DECREASE_KEY(H, x, k)

if $k > key[x]$ **then** chyba, nový kľúč je väčší než pôvodný **fi**
$key[x] \leftarrow k;\ y \leftarrow x;\ z \leftarrow p[y]$
while $z \neq Nil \wedge (key[y] < key[z])$
    do $exchange\ key[y] \leftrightarrow key[z]$
        $y \leftarrow z$
        $z \leftarrow p[y]$ **od**

Odstránenie prvku z haldy

Vrcholu, ktorý chceme odstrániť, znížime pomocou $\textsc{BH\_Decrease\_Key}$ hodnotu kľúča na $-\infty$ a operáciou $\textsc{BH\_Extract\_Min}$ ho odstránime z haldy. Zložitosť $\mathcal{O}(\log n)$.

$\textsc{BH\_Delete}(H, x)$

1 $\textsc{BH\_Decrease\_Key}(H, x, -\infty)$

2 $\textsc{BH\_Extract\_Min}(H)$

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 cena operácií nad binomiálnou hladou

Invariant: každý strom v halde má na svojom účte 1 kredit.

Amortizovaná cena operácie BH_INSERT je $\mathcal{O}(1)$. 1 kredit zaplatí vytvorenie nového vrcholu, 1 kredit dáme na účet novovytvorenému stromu. Spájanie stromov zaplatíme kreditmi stromov, ktoré spájame.

Amortizovaná cena operácie BH_UNION je $\mathcal{O}(\log n)$. Kredity použijeme na zaplatenie spájania koreňových zoznamov háld. Spájanie stromov zaplatíme kreditmi stromov, ktoré spájame.

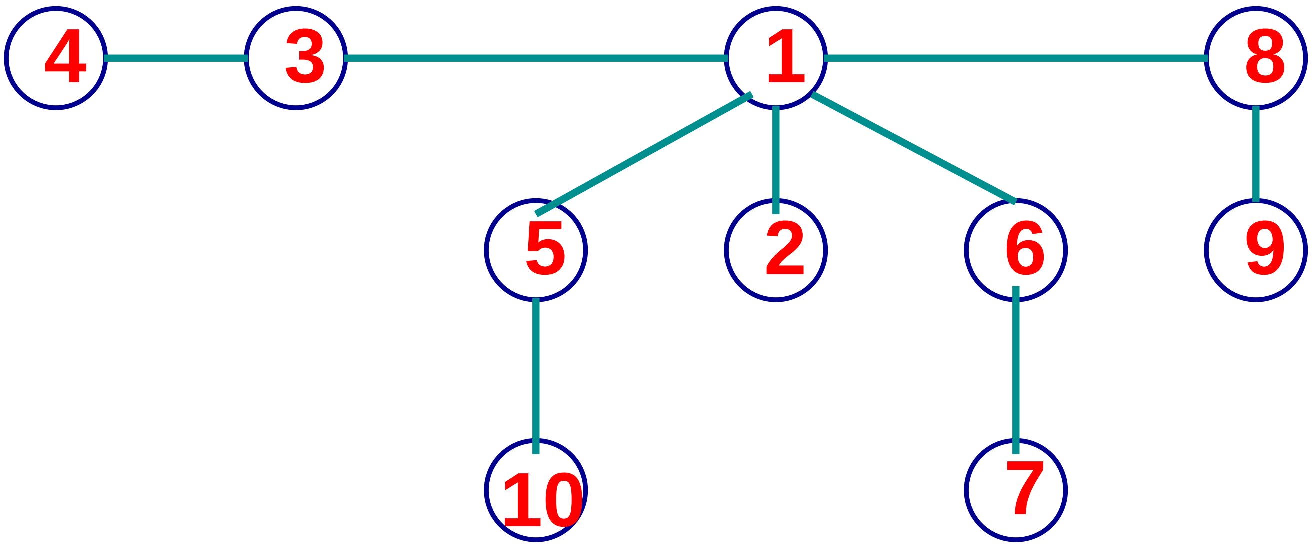
Amortizovaná cena operácie BH_EXTRACT_MIN je $\mathcal{O}(\log n)$. Kredity vložíme na účet stromov, ktoré vznikli odstránením koreňa s minimálnym prvkom. Spájanie stromov zaplatíme kreditmi stromov, ktoré spájame.

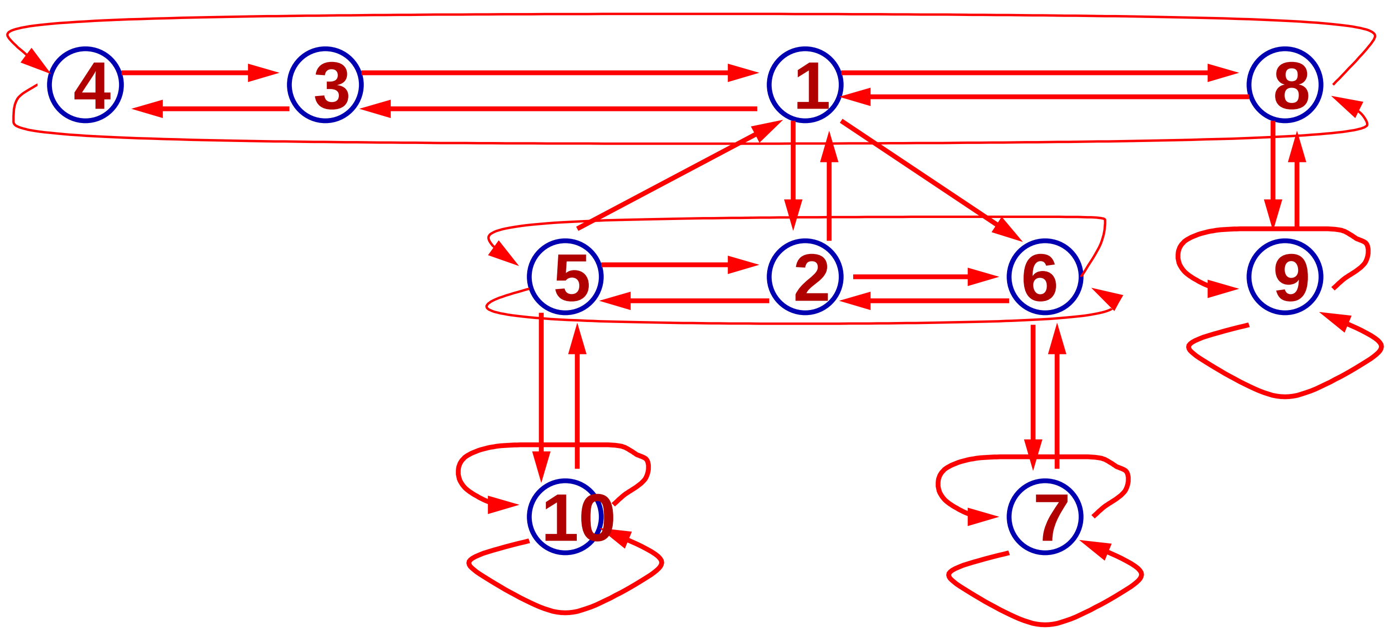
Fibonacciho halda

Je modifikáciou binomiálnej haldy. Dovoľuje efektívnejšiu realizáciu operácií Union, Minimum a Decrease_Key; zároveň nezhoršuje amortizovanú zložitosť ostatných operácií.

Základné princípy

- zavedenie ukazateľa na minimálny prvok
- štruktúra môže obsahovať viac stromov rovnakého stupňa
- stromy nie sú binomiálne (pri súčasnom zachovaní ich vyváženosti)
- 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.


4
3
1
8
5
2
6
9
10
7


4
3
1
8
5
2
6
9
10
7

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

Premenné

$min[H]$ – ukazateľ na minimálny prvok Fibonacciho haldy

$n[H]$ – aktuálny počet prvkov v halde H

Označenie

$D(n)$ – maximálny možný stupeň vrchola vo Fibonacciho halde s n prvkami

$m(H)$ – počet stromov v halde H

Operácie

Vytvorenie prázdnej haldy

$\textsc{Make\_Fib\_Heap}()$

vytvorí prázdnu haldu H, $head[H] = \textsc{nil}$

skutočná cena $= 1$

amortizovaná cena $= \mathcal{O}(1)$ (2 kredity na účet vytvoreného stromu)

Invariant výpočtu: každý strom má na svojom účte 2 kredity

Nájdenie minimálneho prvku

$\textsc{Fib\_Minimum}$(H)

udržujeme ukazateľ $min[H]$ na koreň obsahujúci minimálny prvok

skutočná $=$ amortizovaná cena $= 1$

Vloženie nového prvu do haldy

Operácia $\textsc{Fib\_Insert}$ pripojí k halde H nový vrchol x. Predpokladáme, že $key[x]$ je vkladaný prvok a že pre ostatné parametre platí $p[x] = Nil$, $child[x] = Nil$, $left[x] = right[x] = x$, $mark[x] = \textsc{false}$ a $degree[x] = 0$. Aktualizujeme hodnoty $min[H]$ a $n[H]$.
skutočná cena $= \mathcal{O}(1)$
amortizovaná cena $= \mathcal{O}(1)$ (2 kredity na účet vytvoreného stromu)

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

spoj zoznam x so zoznamom H
if $min[H] = \textsc{nil} \lor key[x] < key[min[H]]$
  then $min[H] \leftarrow x$ **fi**
$n[H] \leftarrow n[H] + 1$

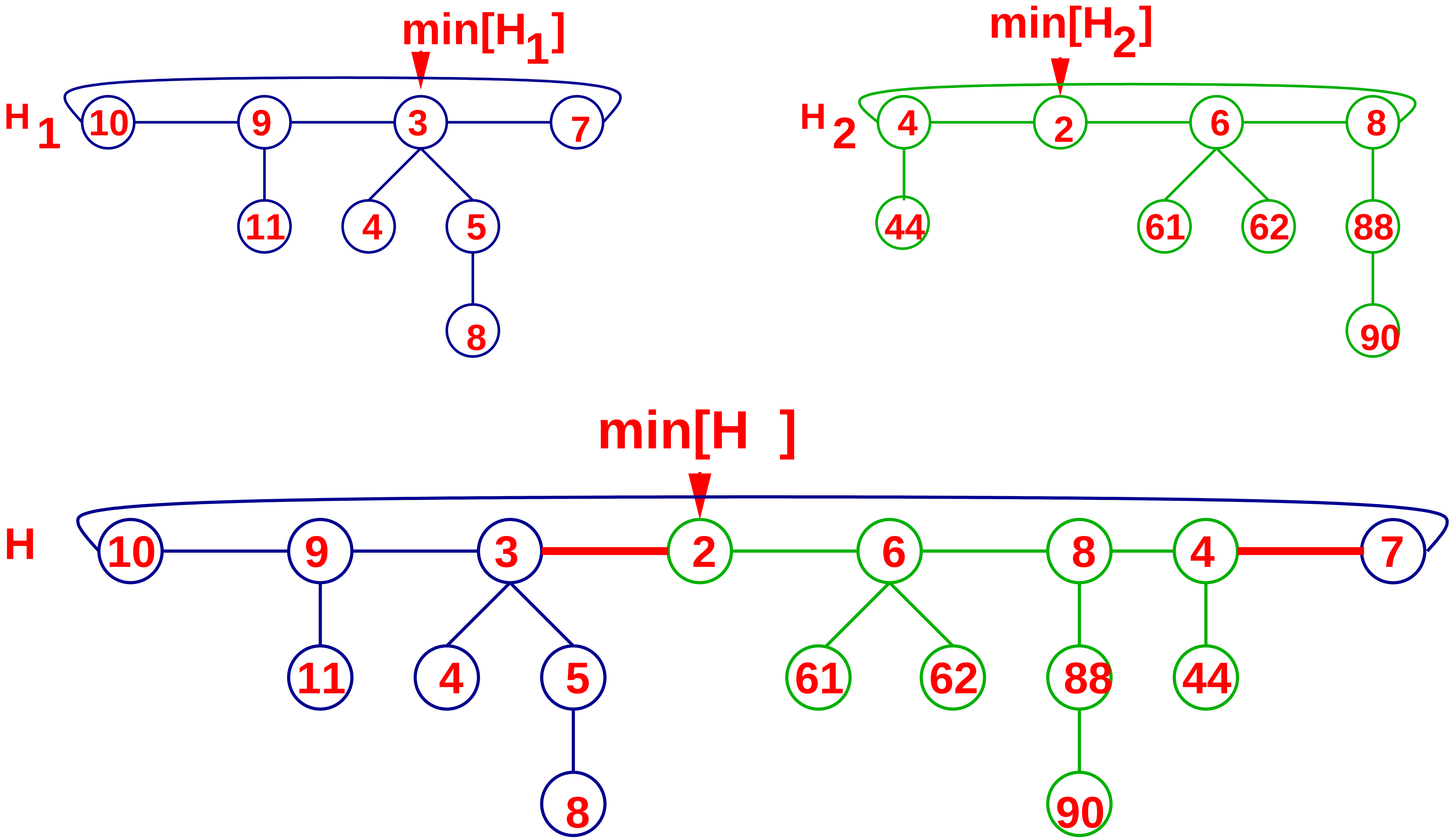
Zjednotenie dvoch háld

– zoznamy H_1 a H_2 spojíme do jedného
– aktualizujeme hodnoty $min[H]$ a $n[H]$.
skutočná $=$ amortizovaná cena $= \mathcal{O}(1)$

$\textsc{Fib\_Union}(H_1, H_2)$

$H \leftarrow \textsc{Make\_Fib\_Heap}()$
spoj zoznamy H_1 a H_2 do zoznamu H
if $(min[H_1] = \textsc{nil}) \vee (min[H_2] \neq \textsc{nil} \wedge min[H_2] < min[H_1])$
  then $min[H] \leftarrow min[H_2]$
  else $min[H] \leftarrow min[H_1]$ **fi**
$n[H] \leftarrow n[H_1] + n[H_2]$
return H

Union



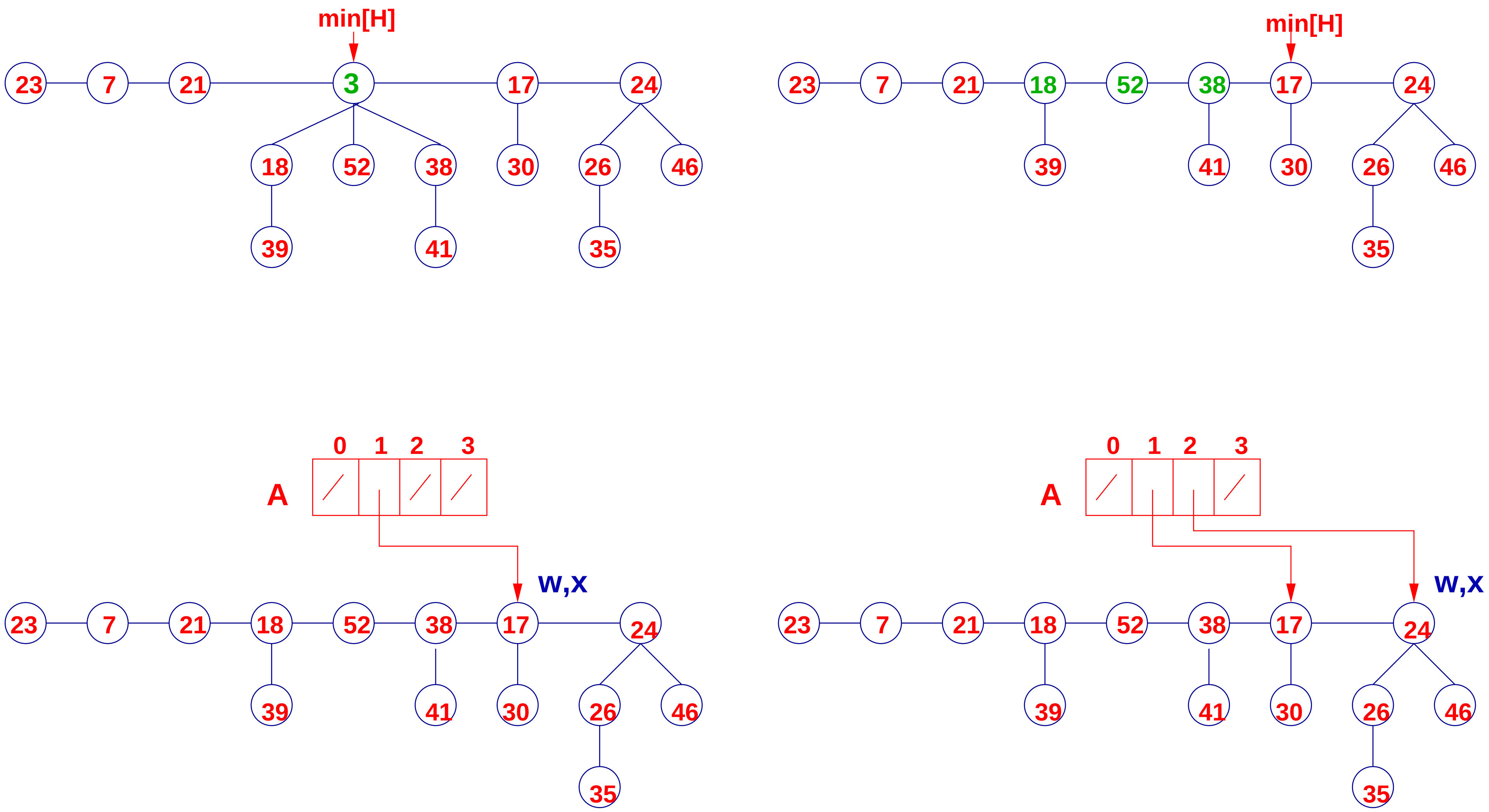
min[H1]
H 1
10
9
3
7
11
4
5
8
min[H2]
H 2
4
2
6
8
44
61
62
88
90
min[H]
H
10
9
3
2
6
8
4
7
11
4
5
61
62
88
44
8
90

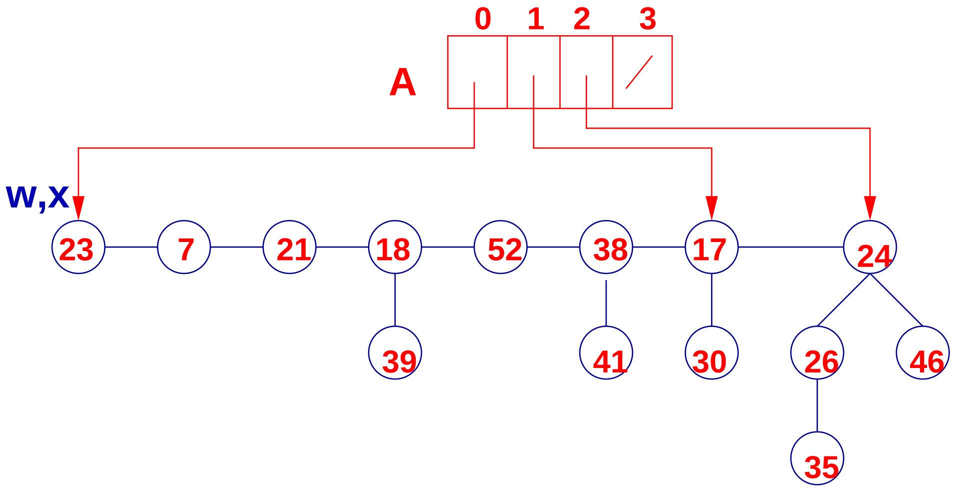
Odstránenie minimálneho prvku z haldy

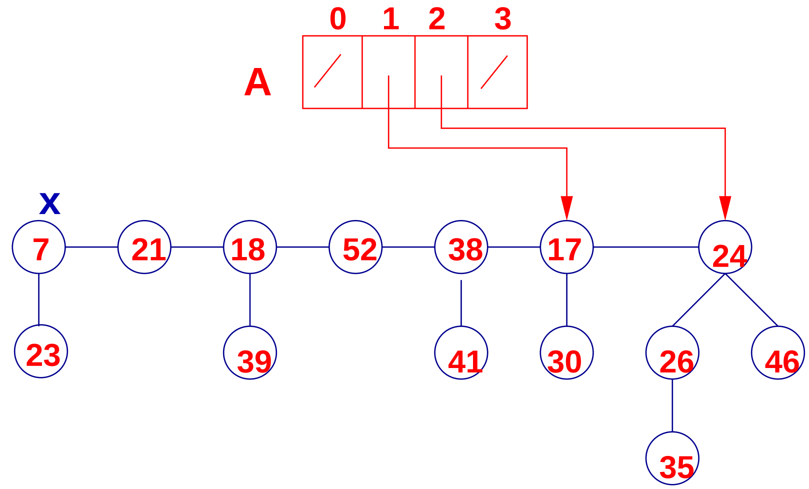
- všetkých synov koreňa z obsahujúceho minimálny prvok pripojíme do koreňového zoznamu H
- odstránime z
- voláme $\textsc{Consolidate}$ a upravíme haldu tak, aby neobsahovala dva stromy rovnakého stupňa
- v upravenej halde prechádzame koreňový zoznam a hľadáme nový minimálny prvok

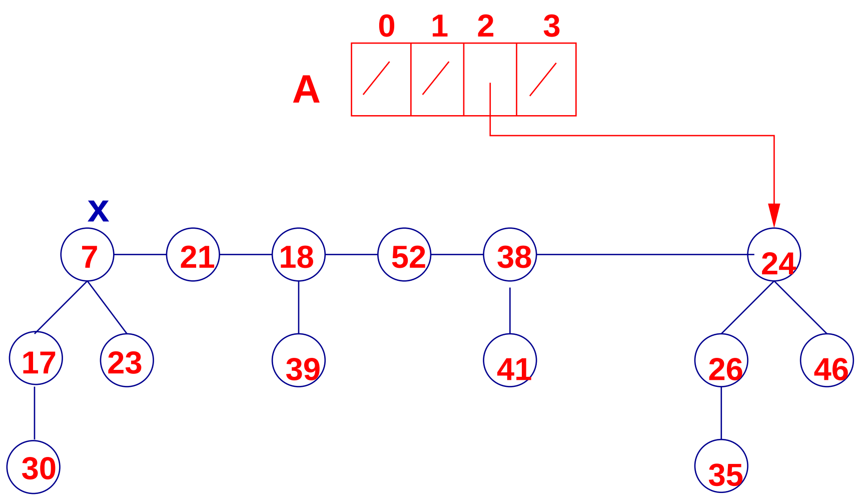
Úprava haldy (CONSOLIDATE)

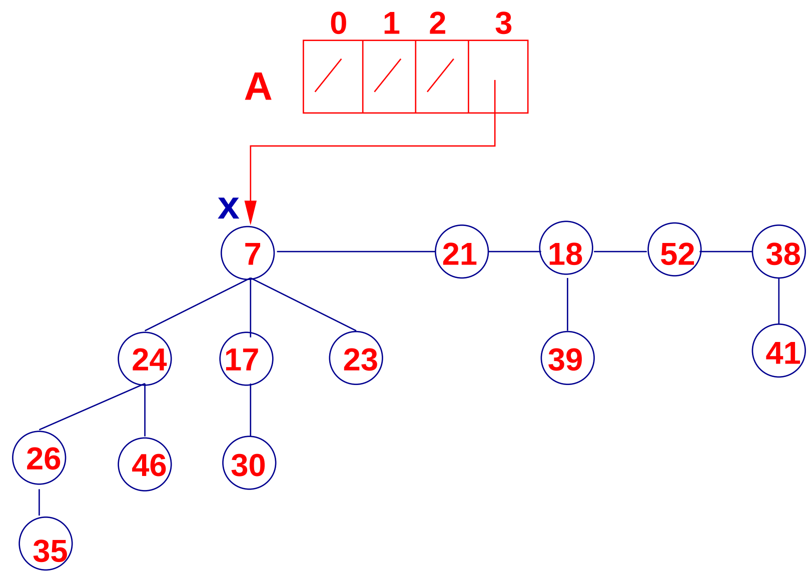
- využíva pomocné pole rozmerov $D(n[H])$
- prechádzame koreňový zoznam H, ukazateľ na každý koreň x uložíme do $A[degree(x)]$. Ak je hodnota $A[degree(x)] \neq Nil$, tak prevedieme spojenie (LINK) príslušných stromov a ukazateľ na výsledný strom uložíme do $A[degree(x) + 1]$. V prípade potreby spájanie opakujeme.
- na základe informácii uložených v A vytvoríme nový koreňový zoznam.

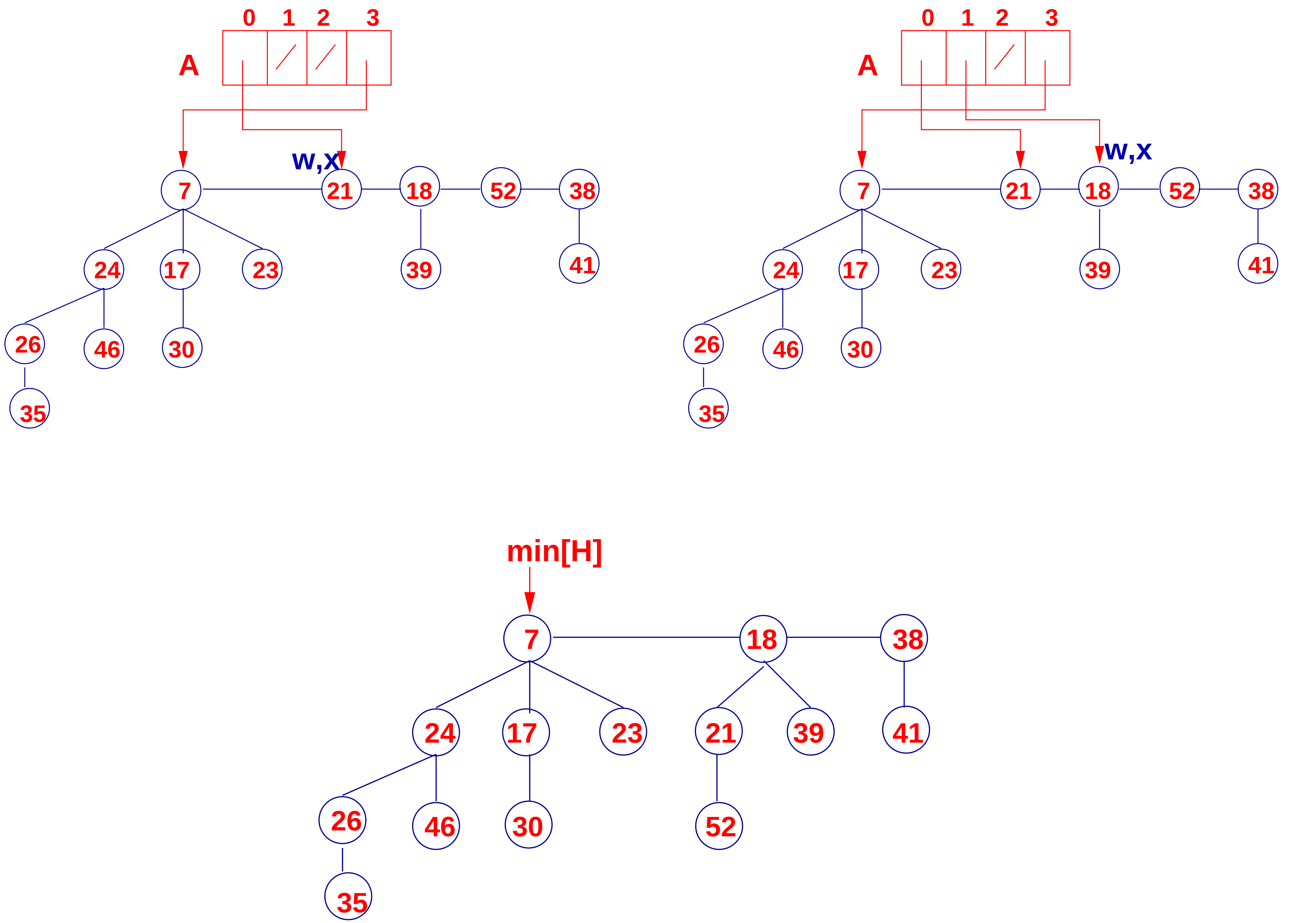

min[H]
23 7 21 3 17 24
18 52 38 30 26 46
39 41 35
min[H]
23 7 21 18 52 38 17 24
39 41 30 26 46
35
A
0 1 2 3
w,x
23 7 21 18 52 38 17 24
39 41 30 26 46
35
A
0 1 2 3
w,x
23 7 21 18 52 38 17 24
39 41 30 26 46
35


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


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


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


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


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

FIB_EXCTRACT_MIN(H)

$z \leftarrow min[H]$
if $z \neq$ NIL
    then for pre každého syna x vrcholu z **do**
            pridaj x do koreňového zoznamu haldy H
            $p[x] \leftarrow$ NIL **od**
        odstráň z zo zoznamu H
        if $z = right[z]$
          then $min[H] \leftarrow$ NIL z bol jediný vrchol haldy
          else $min[H] \leftarrow right[z]$
              CONSOLIDATE(H) **fi**
        $n[H] \leftarrow n[H] - 1$
fi
return z

CONSOLIDATE(H)

for $i \leftarrow 0$ **to** $D(n[H])$ **do**
    $A[i] \leftarrow$ NIL **od**
for pre každý vrchol w v koreňovom zozname haldy H **do**
    $x \leftarrow w$
    $d \leftarrow degree[x]$
    while $A[d] \neq$ NIL **do**
        $y \leftarrow A[d]$
        if $key[x] > key[y]$
          then $exchange\ x \leftrightarrow y$ **fi**
        FIB_LINK(H, y, x)
        $A[d] \leftarrow$ NIL
        $d \leftarrow d + 1$ **od**
    $A[d] \leftarrow x$
od

```
min[H] ← NIL
for i ← 0 to D(n[H]) do
    if A[i] ≠ NIL
      then pridaj A[i] do koreňového zoznamu haldy H
           if min[H] = NIL ∨ key[A[i]] < key[min[H]]
             then min[H] ← A[i]
           fi
    fi
od
```

FIB_LINK(H, y, x)

```
odstráň y z koreňového zoznamu haldy H
sprav y synom x, zvýš degree[x]
mark[y] ← FALSE
```

Amortizovaná zložitosť operácie Fib_Exctract_Min(H)

Operácii priradíme $\mathcal{O}(D(n[H]))$ kreditov.

$D(n)$ – maximálny možný stupeň vrchola vo Fibonacciho halde s n prvkami
$m(H)$ – počet stromov v halde H

FIB_EXCTRACT_MIN

- pripojenie synov vrcholu z do koreňového zoznamu $\approx$ konštantná zložitosť, tj. $\mathcal{O}(1)$ kreditov
- každý nový strom haldy dostane na svoj účet 2 kredity, tj. spolu maximálne $D(n[H])$ kreditov

Consolidate

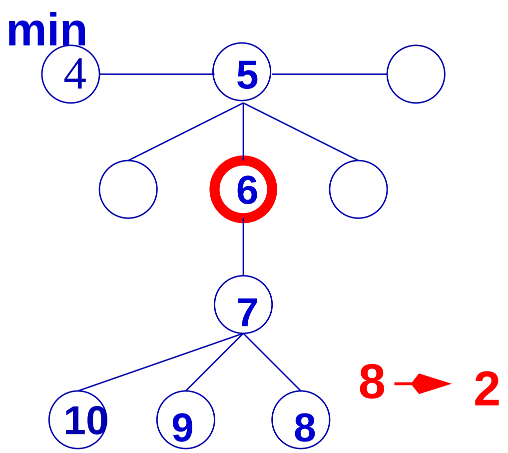
- inicializácia poľa $A \approx D(n[H]) + 1$ kreditov
- vkladanie stromov do poľa A a spájanie stromov (cyklus 1–14): označme m počet stromov v halde, ktorú máme upraviť. Každý strom má na svojom účte 2 kredity. Počas cyklus vložíme do poľa A m stromov a vykonáme k spájaní. Každé spájanie zníži počet stromov o 1 a preto $k < m$. Na zaplatenie všetkých operácii preto postačujú kredity, ktoré majú na svojich účtoch stromy.
- vytvorenie nového koreňového zoznamu na základe informácií uložených v A a hľadanie nového minima (cyklus 16–23) $\approx D(n[H]) + 1$ kreditov
- každému stromu v zrekonštruovanej halde dáme na účet 2 kredity tj. spolu maximálne $D(n[H]) + 1$ kreditov

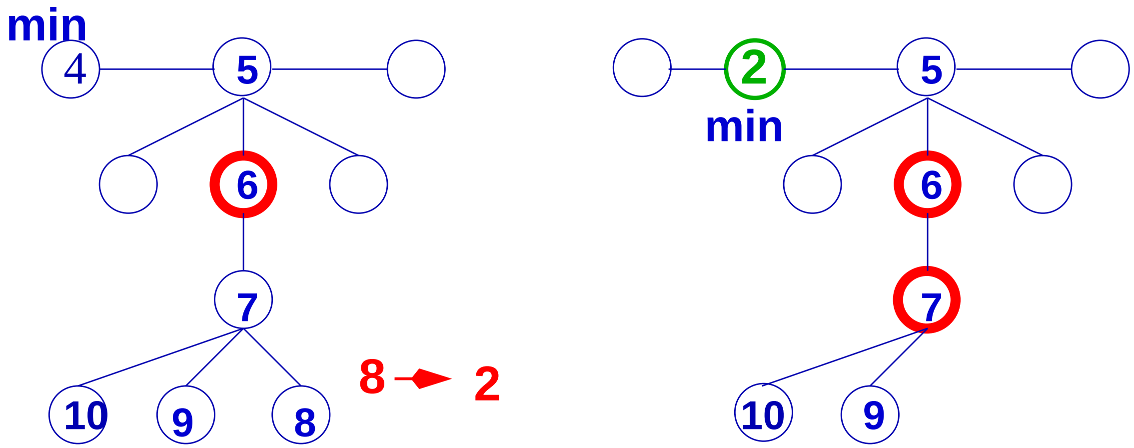
Zníženie hodnoty kľúča

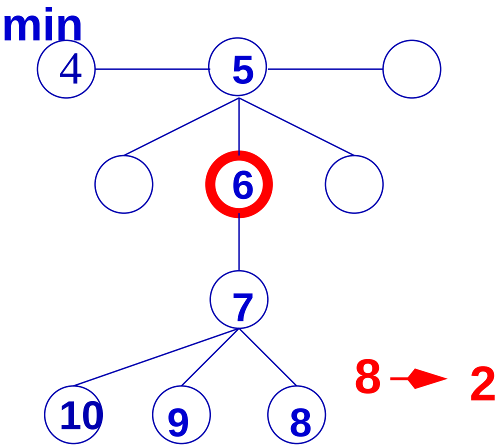
Ak sa znížením kľúča vo vrchole x poruší vlastnosť haldy, tak namiesto výmeny vrcholov (ako v binárnej a binomiálnej halde) odrežeme celý podstrom s koreňom x a urobíme ho novým stromom haldy, tj. pripojíme ho do koreňového zoznamu ($\textsc{Cut}$).

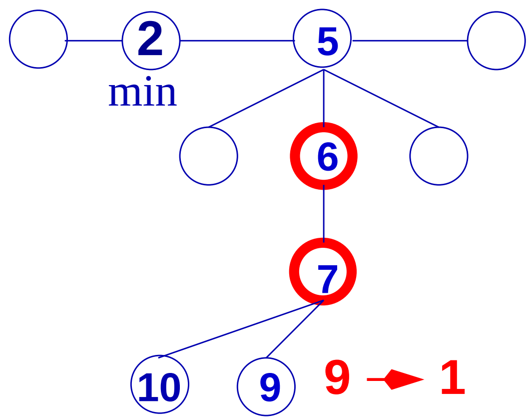
V dôsledku prerezávania sa môže stať, že strom vysokého stupňa má veľmi málo vrcholov a následne halda obsahuje veľmi veľa stromov. Preto pri opakovanom odrezávaní podstromov zároveň znižujeme stupeň stromu.

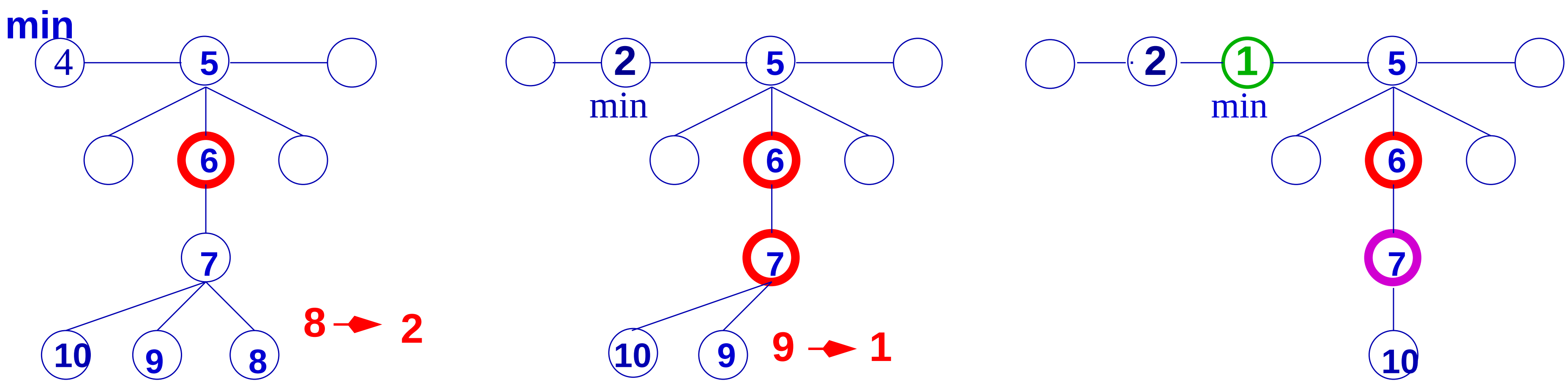
Konkrétne: ak niektorému vrcholu odrežeme druhého syna (príznak $mark$), tak odrežeme aj tento vrchol od jeho otca a v prípade potreby postupujeme s odrezávaním smerom ku koreňu ($\textsc{Cascading\_Cut}$)

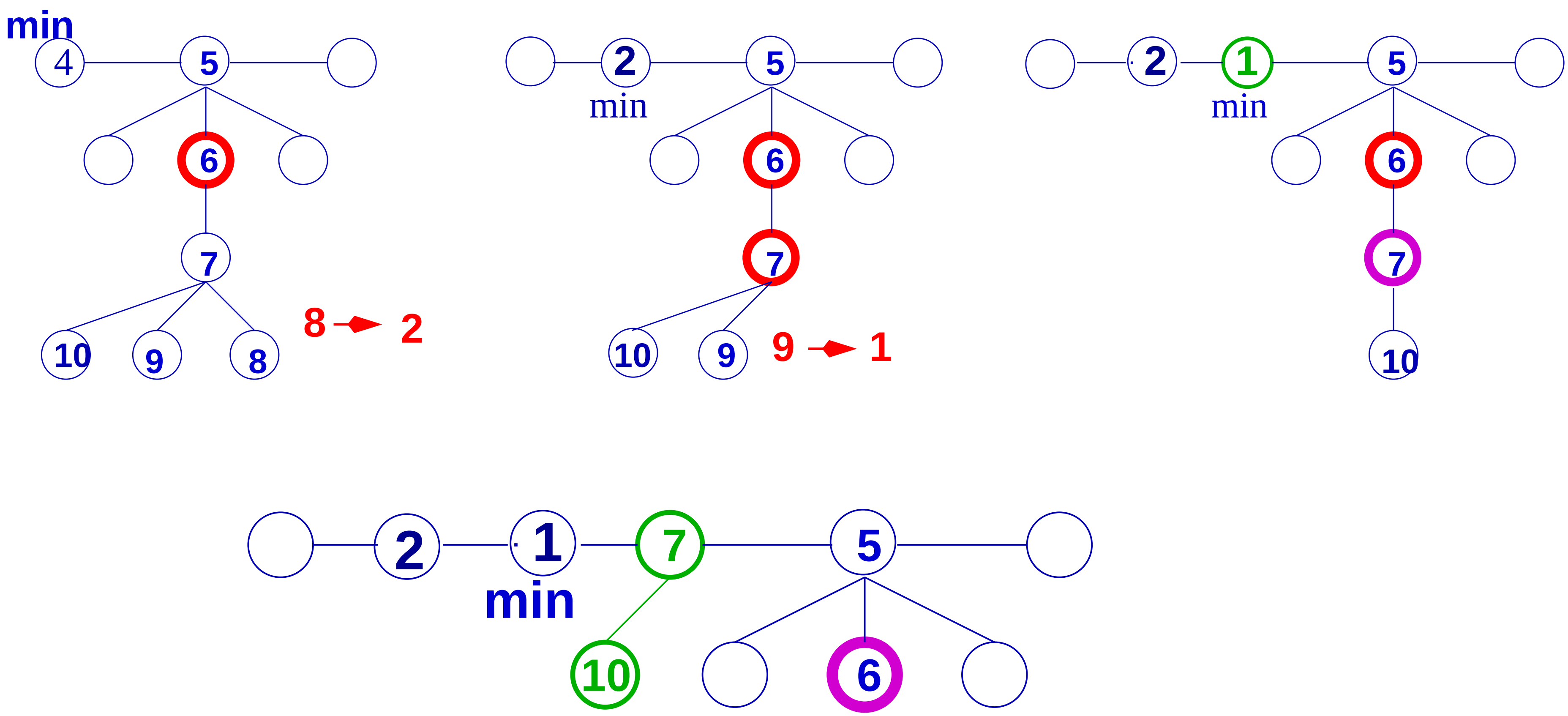

min
4
5
6
7
10
9
8
8
2

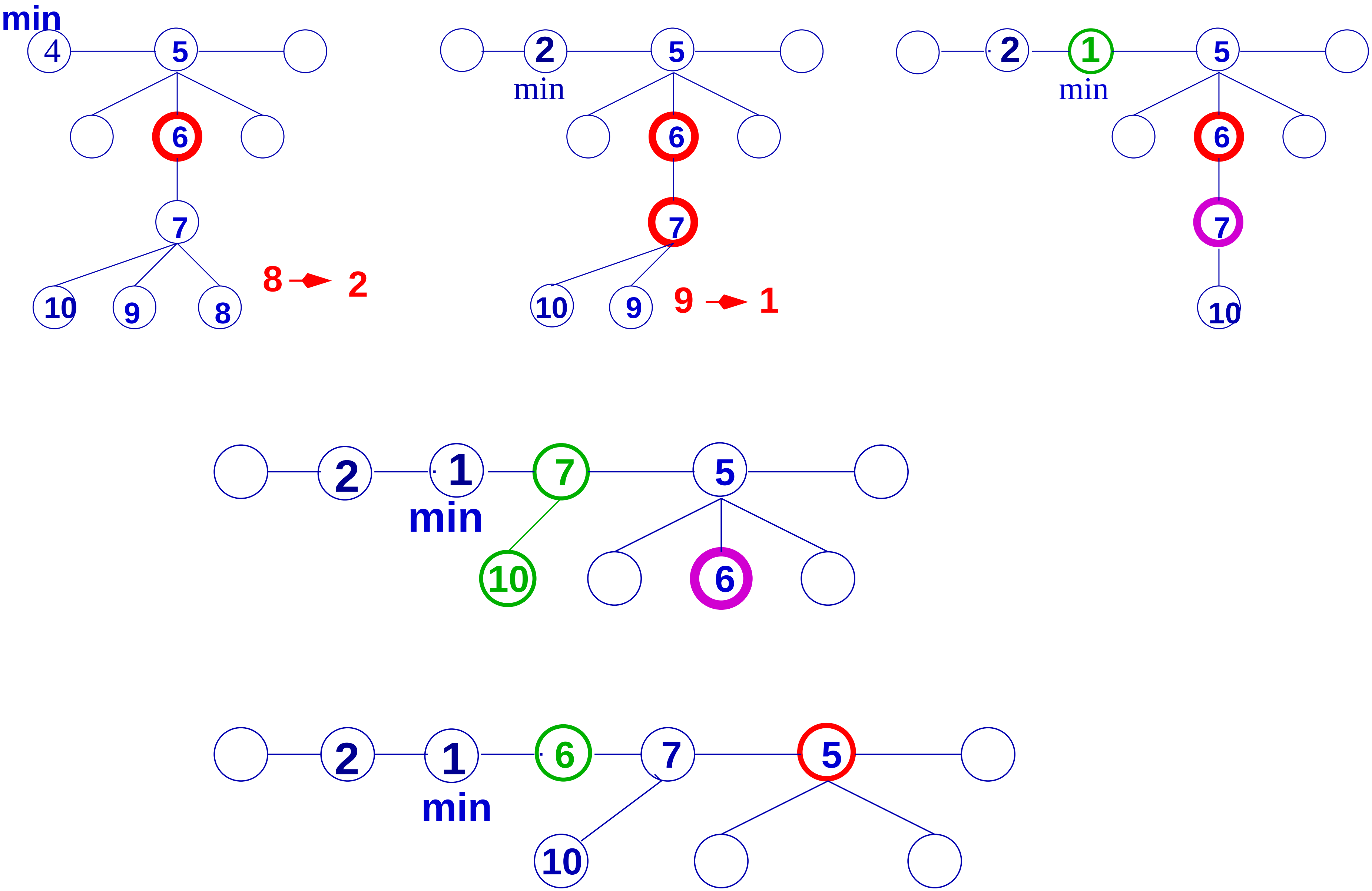

min
4
5
6
7
10
9
8
8
2
2
min
5
6
7
10
9


min
4
5
6
7
10
9
8
8 → 2


2
5
min
6
7
10
9
9 → 1


min
4
5
6
7
10
9
8
8 → 2
2
5
min
6
7
10
9
9 → 1
2
1
5
min
6
7
10


min
4
5
6
7
10
9
8
8 → 2
2
5
min
6
7
10
9
9 → 1
2
1
5
min
6
7
10
2
1
7
5
min
10
6


min
4
5
6
7
10
9
8
8 → 2
2
min
5
6
7
10
9
9 → 1
2
1
min
5
6
7
10
2
1
min
7
5
10
6
2
1
min
6
7
5
10

FIB_DECREASE_KEY(H, x, k)

if $k > key[x]$ **then** chyba, nový kľúč je väčší než pôvodný **fi**
$key[x] \leftarrow k$
$y \leftarrow p[x]$
if $y \neq$ NIL $\wedge$ $key[x] < key[y]$
    then CUT(H, x, y)
        CASCADING_CUT(H, y)
fi
if $key[x] < key[min[H]]$
    then $min[H] \leftarrow x$
fi

$\textsc{Cut}(H, x, y)$

odstráň x zo zoznamu detí vrchola y; zníž $degree[y]$
pridaj x do koreňového zoznamu haldy H
$p[x] \leftarrow$ NIL
$mark[x] \leftarrow$ FALSE

$\textsc{Cascading\_Cut}(H, y)$

$z \leftarrow p[y]$
if $z \neq$ NIL
  then if $mark[y] =$ FALSE **then** $mark[y] \leftarrow$ TRUE
    else $\textsc{Cut}(H, y, z)$
      $\textsc{Cascading\_Cut}(H, z)$
  fi
fi

Amortizovaná zložitosť operácie Fib_Decrease_Key

Na zaplatenie operácie postačuje 6 kreditov

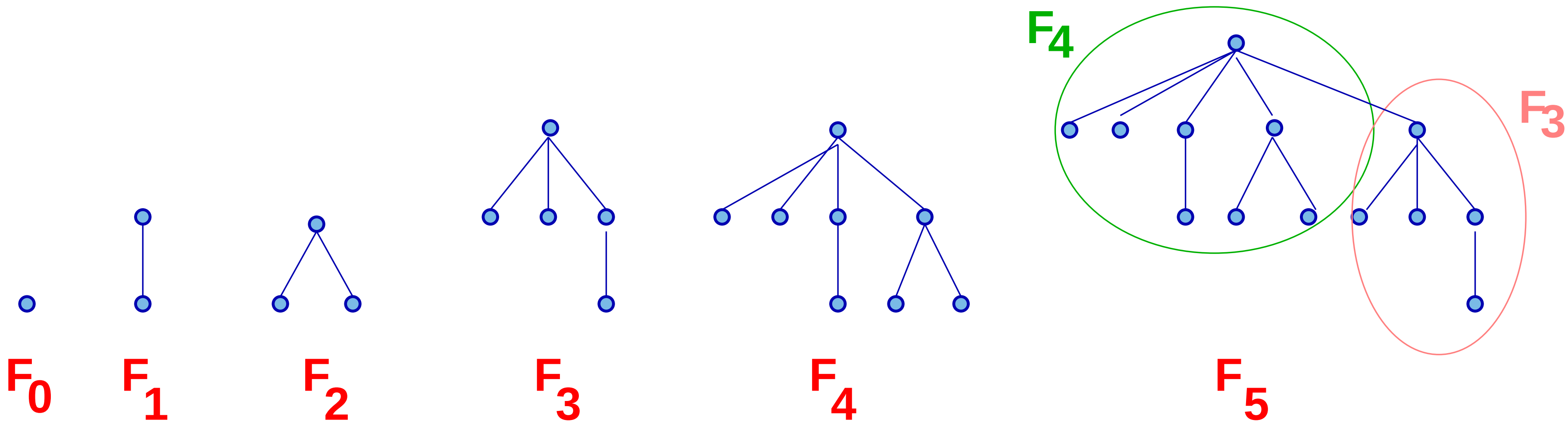
- ak za neodrezáva žiaden vrchol, kredity sa nevyužijú
- ak sa odreže x od y
  1 kredit zaplatí odrezanie
  2 kredity dostane na účet strom s koreňom x (zachovanie invariantu výpočtu)
  3 kredity dostane na účet vrchol y
- v okamihu, keď potrebujeme odrezať y od jeho otca, má y na svojom účte 6 kreditov, ktorými zaplatíme toto odrezanie

Odhad hodnoty $D(n)$

Lema. *Ak strom vo Fibonacciho halde má koreň stupňa k, tak má aspoň $2^{\lceil k/2 \rceil}$ vrcholov.*

Dôkaz. Fixujme časový okamih. Nech x je ľubovoľný vrchol haldy a $y_1, \ldots, y_m$ jeho synovia v poradí, v akom boli pripájaní k x. 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 vo fixovanom okamihu má y_i stupeň aspoň $i-2$.

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



Platí $f_0 = 1, \quad f_1 = 2, \quad f_{n+2} = f_{n+1} + f_n$.

Postupnosť (f_i) má kladné členy, preto je rastúca a $f_k > 2f_{k-2}$.
Dosadením pre k liché
$f_k > 2 \cdot f_{k-2} > 2 \cdot 2 \cdot f_{k-4} > \ldots > 2^{\lfloor k/2 \rfloor} \cdot f_1 = 2^{\lceil k/2 \rceil}$.
Podobne pre k sudé.
$\square$

Pozn.: presnejší odhad je $f_k \geq ((1 + \sqrt{5})/2)^k$.

Dôsledok. *Fibonacciho halda s n vrcholmi má nanajvýš $2 \log n$ stromov, tj. $D(n) \leq 2 \log n$.*

Dôsledok. *Amortizovaná zložitosť operácií* $\textsc{Fib\_Extract\_Min}$ *a* $\textsc{Fib\_Delete}$ *v postupnosti n operáciií nad Fibonacciho haldou je $\mathcal{O}(\log n)$.*

Poznámka: Pod Fibonacciho haldou rozumieme každú štruktúru, ktorá vznikne z prázdnej haldy (tj. haldy vytvorenej operáciou $\textsc{Make\_Fib\_Heap}$*) aplikáciou uvedených operácií.*

Dátové štruktúry pre disjunktné množiny

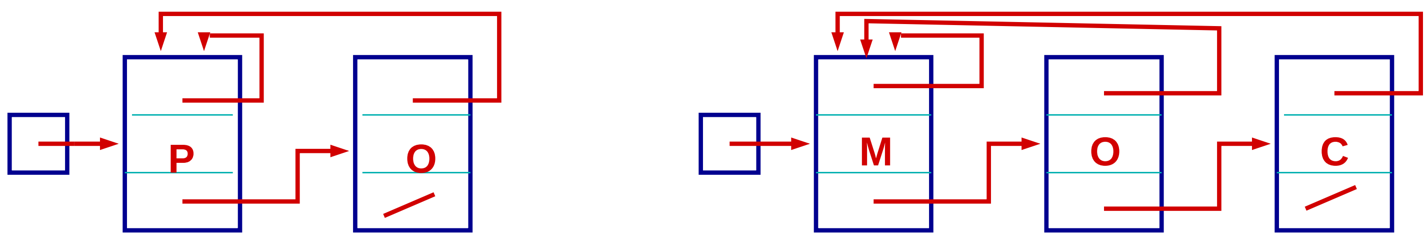
Dátové štruktúry pre reprezentáciu disjunktných množín, z ktorých každá má svojho jednoznačne určeného reprezentanta.

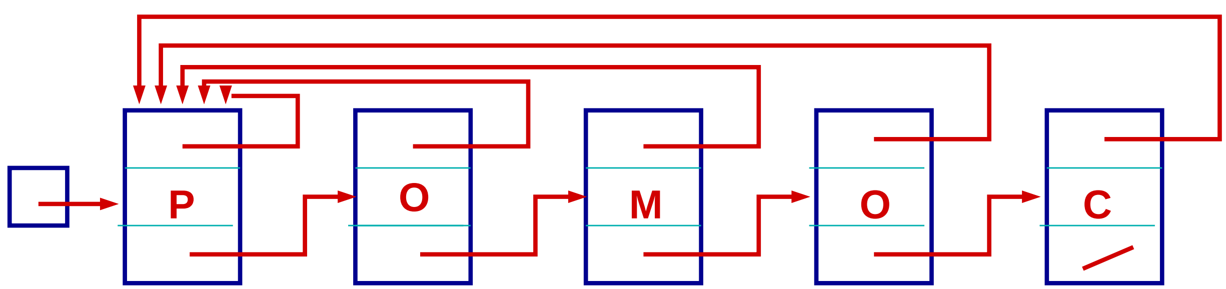
Podporované operácie:
$\textsc{Make\_Set}(x)$ – vytvorí množinu obsahujúcu prvok x
$\textsc{Union}(H_1, H_2)$ – vytvorí novú množinu zjednotením množín H_1 a H_2
$\textsc{Find\_Set}(x)$ – nájde reprezentanta množiny obsahujúcej prvok x.
Operácia $\textsc{Find\_Set}$ dovoľuje efektívne testovať, či dva prvky patria do tej istej množiny.

Implementácia pomocou spájaných zoznamov a lesa stromov.

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.





Operácia $\textsc{Find\_Set}(x)$ má konštantnú zložitosť.

Operácia $\textsc{Union}(H_1, H_2)$ vyžaduje spojenie zoznamov a aktualizáciu ukazateľov na reprezentanta. Vždy pripájame množinu s menším počtom prvkov k množine s väčším počtom prvkov. Zložitosť je rovná kardinalite množiny, ktorú pripájame.

Lema. *Pri reprezentácii množín spájanými zoznamami je zložitosť postupnosti obsahujúcej m operácií* $\textsc{Union}$ *a* $\textsc{Find\_Set}$ *a n operácií* $\textsc{Make\_Set}$ *rovná $\mathcal{O}(m + n\log n)$.*

Dôkaz. Metódou účtov.
Operácii $\textsc{Make\_Set}(x)$ priradíme $\mathcal{O}(\log n)$ kreditov, ktoré uložíme na účet prvku x. Operáciam $\textsc{Union}$ a $\textsc{Find\_Set}$ priradíme $\mathcal{O}(1)$ kreditov.

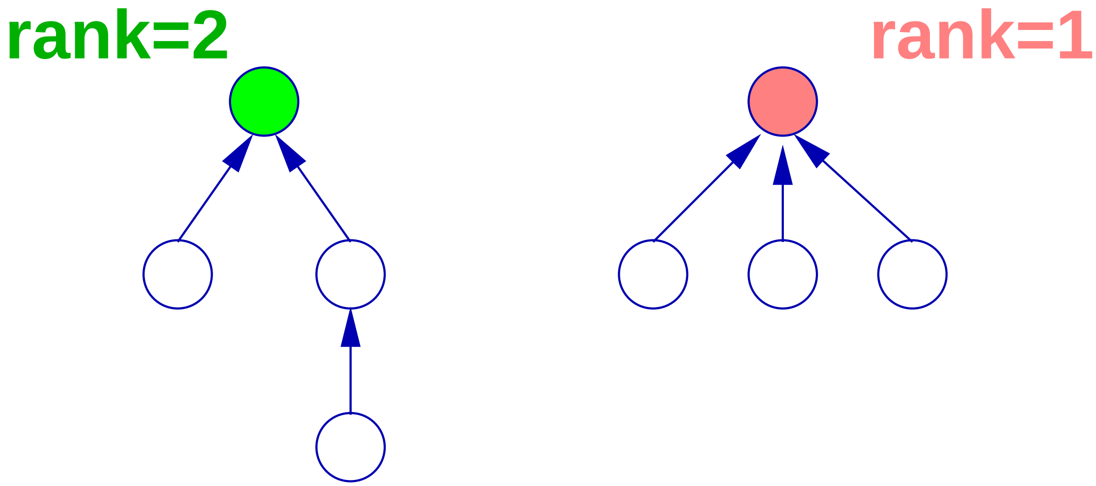
Indukciou k i overíme, že ak prvok x sa zúčastnil i zjednocovaní (i-krát sa zmenil jeho ukazateľ na reprezentanta), tak množina obsahujúca x má kardinalitu aspoň 2^i. Preto kredity, ktoré dostal prvok x, stačia na zaplatenie všetkých operácií, ktorých sa x účastní.
□

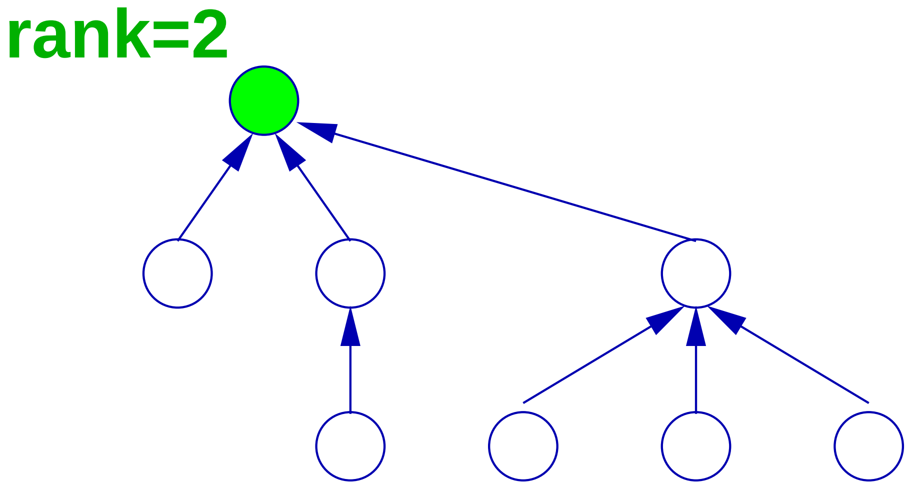
Les stromov

Množinu reprezentujeme ako strom. Každý vrchol ukazuje na svojho otca. Reprezentantom množiny je prvok uložený v koreni stromu. Každý vrchol x obsahuje údaj $rank[x]$, ktorý zhora ohraničuje výšku podstromu s koreňom x.

UNION – koreň stromu s menším $rank$om sa stane synom koreňa stromu s väčším $rank$om. Zložitosť operácie je konštantná.

UNION





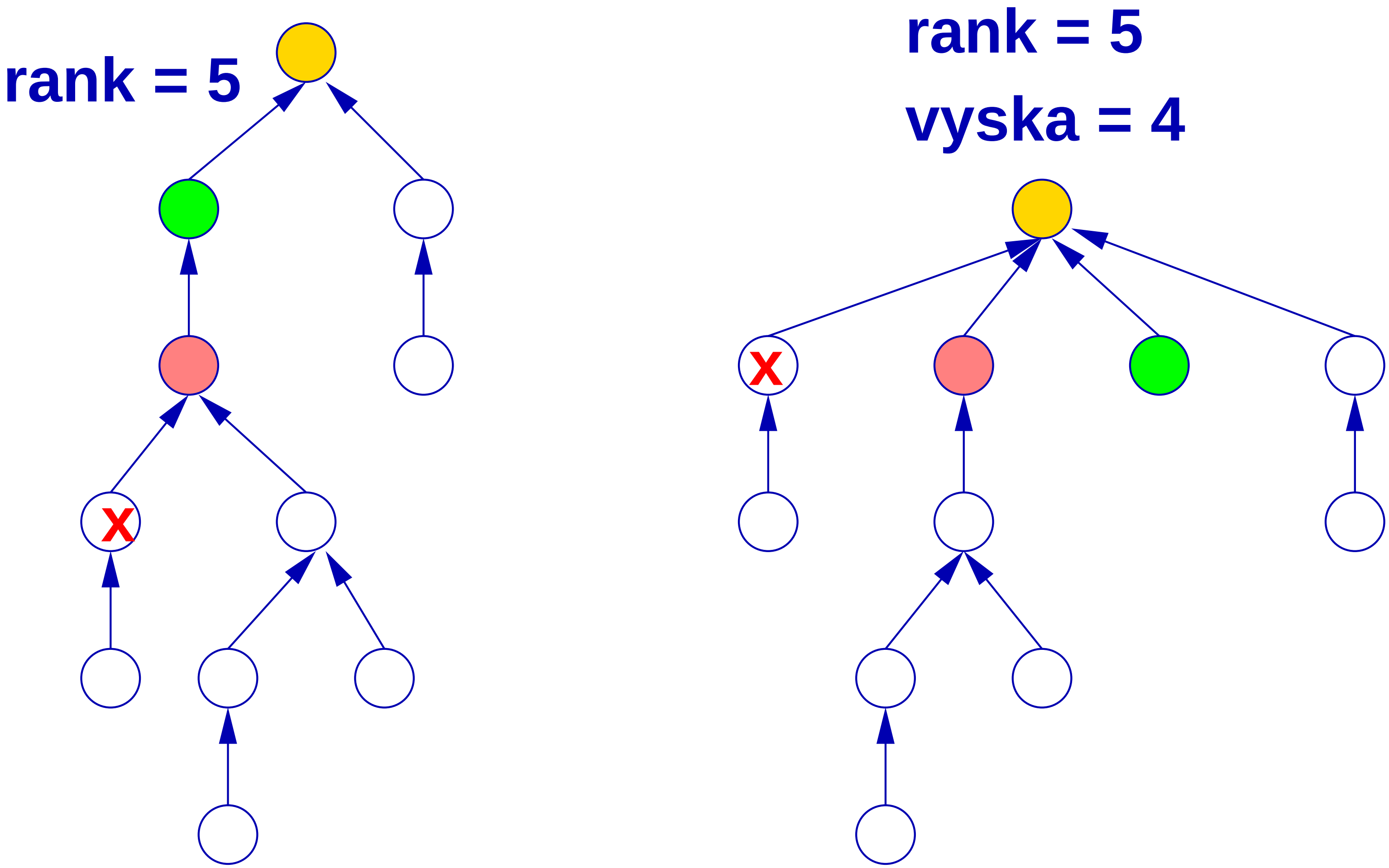
Les stromov

Množinu reprezentujeme ako strom. Každý vrchol ukazuje na svojho otca. Reprezentantom množiny je prvok uložený v koreni stromu. Každý vrchol x obsahuje údaj $rank[x]$, ktorý zhora ohraničuje výšku podstromu s koreňom x.

$\textsc{Union}$ – koreň stromu s menším $rank$om sa stane synom koreňa stromu s váčším $rank$om. Zložitosť operácie je konštantná.

$\textsc{Find\_Set}(x)$ – sledujeme cestu od vrcholu obsahujúceho x do koreňa. Následne prechádzame cestu ešte raz a každý vrchol cesty urobíme synom koreňa.

FIND(x)



MAKE_SET(x)

$p[x] \leftarrow x$
$rank[x] \leftarrow 0$

UNION(x, y)

LINK(FIND_SET(x), FIND_SET(y))

LINK(x, y)

if $rank[x] > rank[y]$ **then** $p[y] \leftarrow x$
else $p[x] \leftarrow y$
if $rank[x] = rank[y]$
then $rank[y] \leftarrow rank[y] + 1$ **fi fi**

FIND_SET(x)

if $x \neq p[x]$ **then** $p[x] \leftarrow$ FIND_SET($p[x]$) **fi**
return $p[x]$

Lema. *Pri reprezentácii množín stromami je zložitosť postupnosti obsahujúcej m operácií* UNION *a* FIND_SET *a n operácií* MAKE_SET *rovná $\mathcal{O}(m\alpha(n))$.*

Definícia funkcie $\alpha(n)$

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

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

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

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

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

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

Dátová štruktúra Union-Find a Kruskalov algoritmus

Úlohou je pre daný neorientovaný graf $G = (V, H)$ s ohodnotením hrán w nájsť najlacnejšiu kostru. Algoritmus využíva dátovú štruktúru $\textsc{Union-Find}$ na reprezentáciu disjunktných množín; každá množina reprezentuje jeden strom v aktuálnom lese. 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 množiny zjednotíme.

$\textsc{Kruskal}((V, H), w)$

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

Techniky návrhu algoritmov

- Rozdeľ a panuj
- Dynamické programovanie
- Hladové algoritmy
- Backtracking
- ...

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 $S[1..n]$. Zložitostné kritérium - počet porovnaní prvkov.

MAX(S)

```
max ← S[1]
for i = 2 to n do
    if S[i] > max then max ← S[i] fi
od
```

Minimum nájdeme medzi zvyšnými $n-1$ prvkami podobne.
Celkove $(n-1)+(n-2)$ porovnaní.

Prístup Rozdeľ a panuj

1. Problém rozdeľ na podproblémy
2. Vyrieš podproblémy
3. Z riešení podproblémov zostav riešenie problému

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

function $\textsc{Maxmin}(x, y)$ $\quad$ **predpoklad $x < y$

if $y = x + 1$ **then return** $(\max(S[x], S[y]), \min(S[x], S[y]))$ **fi**

if $y = x + 2$

  then použitím 3 porovnaní utrieď $S[x], S[x+1], S[y]$

    return $(\max, \min)$

fi

if $y > x + 2$

  then $(max1, min1) \leftarrow \textsc{Maxmin}(x, \lfloor (x+y)/2 \rfloor)$

    $(max2, min2) \leftarrow \textsc{Maxmin}(\lfloor (x+y)/2 \rfloor + 1, y)$

    return $(\max(max1, max2) \min(min1.min2))$

fi

Korektnosť: indukciou vzhľadom k $n = y - x + 1$ ukážeme, že $\textsc{Maxmin}(x, y)$ vráti maximálnu a minimálnu hodnotu $S[x..y]$.

Zložitosť: (počet porovnaní)

$$T(n) = \begin{cases} 1 & \text{pre } n = 2 \\ 3 & \text{pre } n = 3 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & \text{inak} \end{cases}$$

Indukciou k n overíme, že $T(n) \leq \frac{5}{3}n - 2$.
1. Pre $n = 2$ platí $\frac{5}{3} \cdot 2 - 2 > 1 = T(2)$.
Pre $n = 3$ platí $\frac{5}{3} \cdot 3 - 2 \geq 3 = T(3)$.
2. Predpokladajme platnosť nerovnosti pre všetky hodnoty $2 \leq i < n$, dokážeme jej platnosť pre n.

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

Dynamické programovanie

1. Charakterizuj štruktúru optimálneho riešenia
   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.

Násobenie matíc

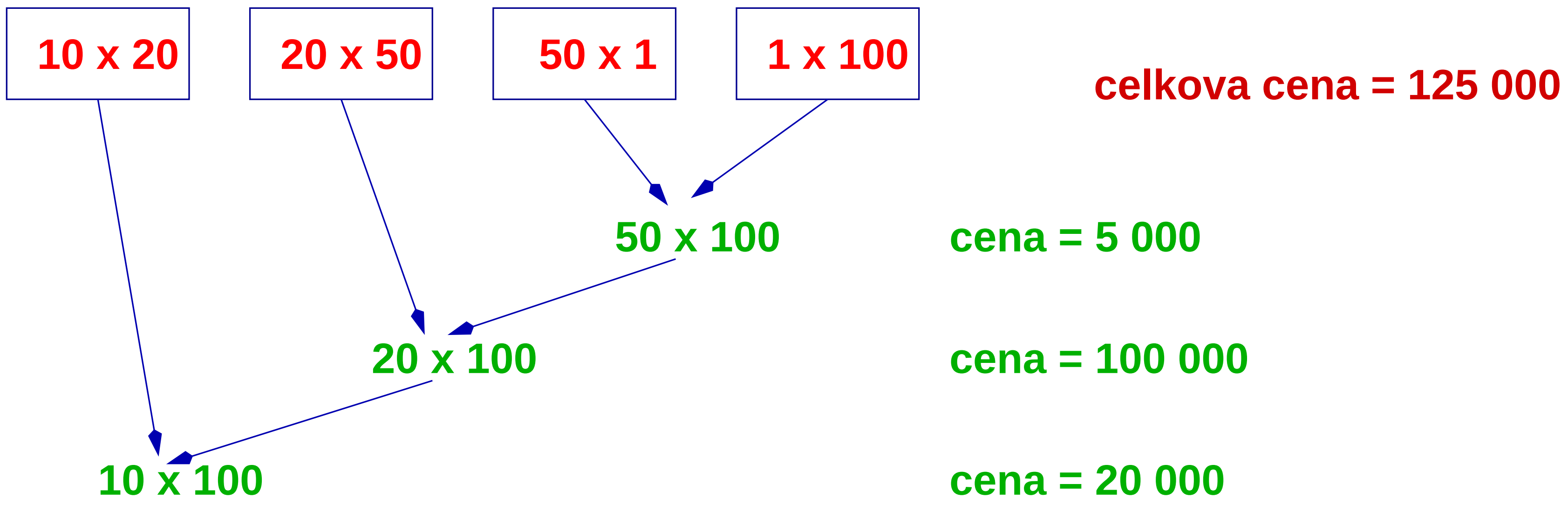
Je daná postupnosť matíc $\langle A_1, \ldots, A_n \rangle$
Matica A_i má rozmery $p_{i-1} \times p_i$.
Úlohou je vypočítať ich súčin, zložitostné kritérium je počet skalárnych násobení. Zložitosť výpočtu závisí od poradia, v akom násobíme matice.

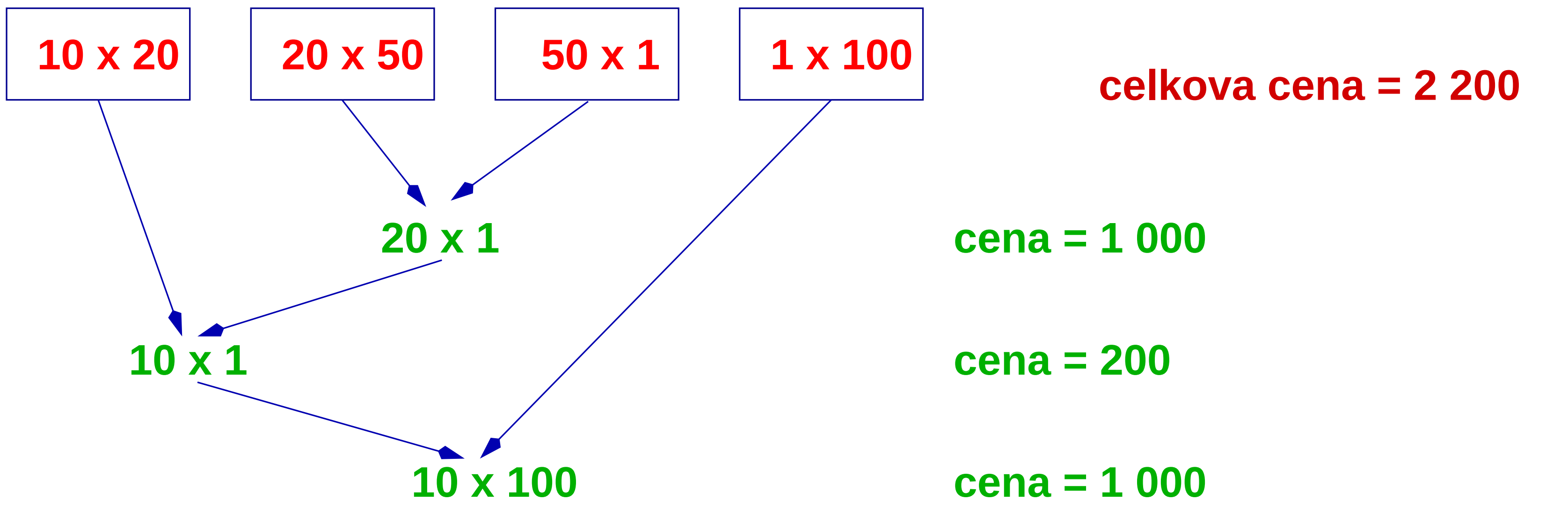
Úlohou je navrhnúť algoritmus, ktorý určí, v akom poradí sa majú matice násobiť tak, aby celková zložitosť výpočtu bola minimálna.

Počet rôznych spôsobov, ako môžeme uzátvorkovať postupnosť n matíc, $P(n)$, 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 $P(n) \geq 2^{n-2}$.


10 x 20
20 x 50
50 x 1
1 x 100
celkova cena = 125 000
50 x 100
cena = 5 000
20 x 100
cena = 100 000
10 x 100
cena = 20 000


10 x 20
20 x 50
50 x 1
1 x 100
celkova cena = 2 200
20 x 1
cena = 1 000
10 x 1
cena = 200
10 x 100
cena = 1 000

Štruktúra optimálneho riešenia

Označme $A_{i..j}$ $(i \leq j)$ súčin matíc $A_i, \ldots, A_j$.

Pre výpočet $A_{i..j}$ musíme najprv vypočítať $A_{i..k}$ a $A_{k+1..j}$ pre nejaké k a nakoniec vynásobiť matice $A_{i..k} \cdot A_{k+1..j}$.

Podproblémami problému ozátvorkovania post. $< A_i, \ldots, A_j >$ je ozátvorkovanie postupností $< A_i, \ldots, A_k >$, $< A_{k+1}, \ldots, A_j >$ pre všetky $i \leq k < j$.

Nech optimálne ozátvorkovanie rozdelí $A_i \cdots A_j$ medzi A_k a A_{k+1}. Ľahko overíme (sporom), že aj ozátvorkovanie $A_i \cdots A_k$ a $A_{k+1} \cdots A_j$ musí byť optimálne.

Preto ak poznáme optimálne riešenia všetkých podproblémov, môžeme zostaviť riešeniu problému.

Hodnota optimálneho riešenia

Označme $m[i,j]$ minimálny počet skalárnych násobení potrebný na výpočet $A_{i..j}$. Platí

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

Pretože $m[i,j]$ určuje len hodnotu optimálneho riešenia ale nie optimálny spôsob ozátvorkovania, definujeme $s[i,j]$ ako tú hodnotu k, pre ktorú sa realizuje minimum.

Výpočet hodnoty optimálneho riešenia technikou Rozdeľ a panuj

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

Zložitosť: nech $T(n)$ je zložitosť výpočtu $m[i,j]$ pre $j - i + 1 = n$.
Pre $n > 0$ je (d je vhodná konštanta)

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

Platí $T(n) - T(n-1) = 2T(n-1) + d$ a $T(n) = \Theta(3^n)$

Výpočet hodnoty optimálneho riešenia technikou Dynamického programovania

$\textsc{Násobenie matíc}\langle A_1, \ldots, A_n \rangle$
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**
            $q \leftarrow m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
            if $q < m[i,j]$ **then** $m[i,j] \leftarrow q;\ s[i,j] \leftarrow k$ **fi**
        od
    od
od

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

Najväčšia spoločná podpostupnosť

Nech $X =< x_1, \ldots, x_m >$ a $Z =< z_1, \ldots, z_k >$ sú postupnosti symbolov. Z je *podpostupnosťou* X práve ak existuje rastúca postupnosť indexov $< i_1, \ldots, i_k >$ postupnosti X taká, že pre všetky $j = 1, \ldots, k$ platí $x_{i_j} = z_j$.

Pre dané dve postupnosti symbolov X a Y hovoríme, že postupnosť Z je ich *spoločnou podpostupnosťou* práve ak Z je podpostupnosťou X aj Y.

Problém najdlhšej spoločnej podpostupnosti je pre dané dve postupnosti symbolov X a Y nájsť ich najdlhšej spoločnú podpostupnosť (NSP).

Štruktúra optimálneho riešenia

Označenie: pre postupnosť $X =< x_1, \ldots, x_m >$ definujeme jej i-ty prefix (pre $i = 0, \ldots, m$) ako $X_i =< x_1, \ldots, x_i >$.

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 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}.

Hodnota optimálneho riešenia

Definujme $c[i,j]$ ako dĺžku NSP postupností X_i a Y_j. Platí

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

Poradie výpočtu:

$$\begin{array}{l} c[1,1], \ldots, c[1,n], \\ c[2,1], \ldots, c[2,n], \\ \quad\;\vdots \qquad\;\; \vdots \\ c[m,1], \ldots, c[m,n] \end{array}$$

Výpočet hodnoty optimálneho riešenia

$\mathrm{NSP}(X, Y)$

for $i = 1$ **to** m **do** $c[i, 0] \leftarrow 0$ **od**
for $j = 0$ **to** n **do** $c[0, j] \leftarrow 0$ **od**
for $i = 1$ **to** m **do**
  for $j = 1$ **to** n **do**
    if $X_i = Y_j$ **then** $c[i, j] \leftarrow c[i-1, j-1] + 1$
      $b[i, j] \leftarrow \clubsuit$
    else if $c[i-1, j] \geq c[i, j-1]$
      then $c[i, j] \leftarrow c[i-1, j]$
        $b[i, j] \leftarrow \spadesuit$
      else $c[i, j] \leftarrow c[i, j-1]$
        $b[i, j] \leftarrow \heartsuit$
    fi fi od od
return c, b

Zostavenie optimálneho riešenia

$\text{PRINT\_NSP}(b, X, i, j)$

if $i = 0 \lor j = 0$ **then return fi**
if $b[i, j] = \clubsuit$ **then** $\text{PRINT\_NSP}(b, X, i-1, j-1)$
    print x_i
  else if $b[i, j] = \spadesuit$
    then $\text{PRINT\_NSP}(b, X, i-1, j)$
    else $\text{PRINT\_NSP}(b, X, i, -1)$
  fi
fi

Ďalšie príklady algoritmov dynamického programovania

- Floydov algoritmus
- Warshallov algoritmus
- Konštrukcia vyváženého binárneho vyhľadávacieho stromu

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 (KLO), 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 $h_1, \ldots, h_k$, 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. Platí

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

Dynamický prístup: počítame hodnoty $MIN[1], \ldots, MIN[N]$. ZLožitosť je $\mathcal{O}(Nk)$, algoritmus vždy vypočíta optimálnu hodnotu.

Hladový prístup: KLO = ak máme zaplatiť hodnotu i, tak vyberieme mincu h_x maximálnej hodnoty takú, že $h_x \leq i$.

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: KLO $=$ vyber spomedzi ešte neuložených súbor najkratší a ulož ho na pásku (tj. ulož súbory na pásku od najkratšieho po najdlhší).

Korektnosť

Lema 1. *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

Je daných n prednášok, každá prednáška i má pevne stanovený svoj začiatok z_i a koniec k_i ($z_i \leq k_i$). Prednášky i a j nazveme *kompatibilné* práve ak $z_i \geq k_j$ alebo $z_j \geq k_i$. Úlohou je vybrať čo najväčšiu množinu vzájomne kompatibilných prednášok (tj. prednášok, ktoré sa môžu konať v jednej posluchárni).

Hladový algoritmus: KLO = vyber prednášku, ktorá má minimálnu hodnotu k_i a je kompatibilná s už vybranými prednáškami.

predpokladáme, že $k_1 \leq k_2 \leq \ldots \leq k_n$
$A \leftarrow \{1\}$; $j \leftarrow 1$
for $i = 2$ **to** n **do if** $z_i \geq k_j$ **then** $A \leftarrow A \cup \{i\}$
$j \leftarrow i$ **fi od**
return A

Korektnosť

Indukciou vzhľadom k veľkosti vypočítaneho rozvrhu A.

1. $|A| = 1$
   Zrejme $A = \{1\}$ a z výpočtu plynie, že žiadna z prednášok nie je kompatibilná s 1. Potom ale žiadne dve prednášky nie sú kompatibilné a nájdené riešenie je optimálne.
2. $|A| = i$, predpokladáme platnosť pre $|A| \leq i - 1$
   V prvom kroku sa do A zaradí prednáška 1. V následnom výpočte sa algoritmus aplikuje na výpočet rozvrhu pre množinu S' prednášok kompatibilných s 1 a vypočíta sa $A' = A \setminus \{1\}$. Pretože $|A'| = i-1$, je podľa IP optimálny. Ukážeme, že potom aj A je optimálny rozvrh pre všetky prednášky.

(a) A zrejme obsahuje kompatibilné úlohy.

(b) A je maximálny – ukážeme sporom.
Nech by existoval rozvrh B pre S taký, že $|B| > |A|$. Potom existuje rozvrh B' taký, že $|B'| = |B|$ a B' obsahuje 1 ($B' = (B \setminus \{i\}) \cup \{1\}$, kde i je úloha z B s najmenším k_i. Rozvrh B' je kompatibilný, pretože $k_1 \leq k_i$.). Ale potom aj $B' \setminus \{1\}$ je rozvrhom pre S'. Platí $|B'| - 1 = |B| - 1 > |A| - 1 = |A'|$ a to je spor s optimalitou A'.

Príklady hladových algoritmov

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

Backtracking

Technika prehľadávania priestoru potenciálnych riešení založená na princípe Rozdeľ a panuj.

- do priestoru potenciálnych riešení vnesieme stromovú štruktúru, (binárne reťazce, k-árne reťazce, permutácie, kombinácie)
- stromovú štruktúru prehľadávame do hĺbky
- prehľadávanie zefektívnime odrezávaním ciest, ktoré dokázateľne nevedú k hľadanému riešeniu

Príklady

Batoh – potenciálne riešenia sú všetky podmnožiny predmetov

Hamiltonovský cyklus – p.r. sú všetky permutácie vrcholov grafu

k*-klika* – p.r. sú všetky kombinácie k-tej triedy vrcholov grafu

Návrh backtrackovacieho algoritmu

1. Voľba spôsobu reprezentácie potenciálnych riešení.

2. Generovanie potenciálnych riešení technikou Rozdeľ a panuj.

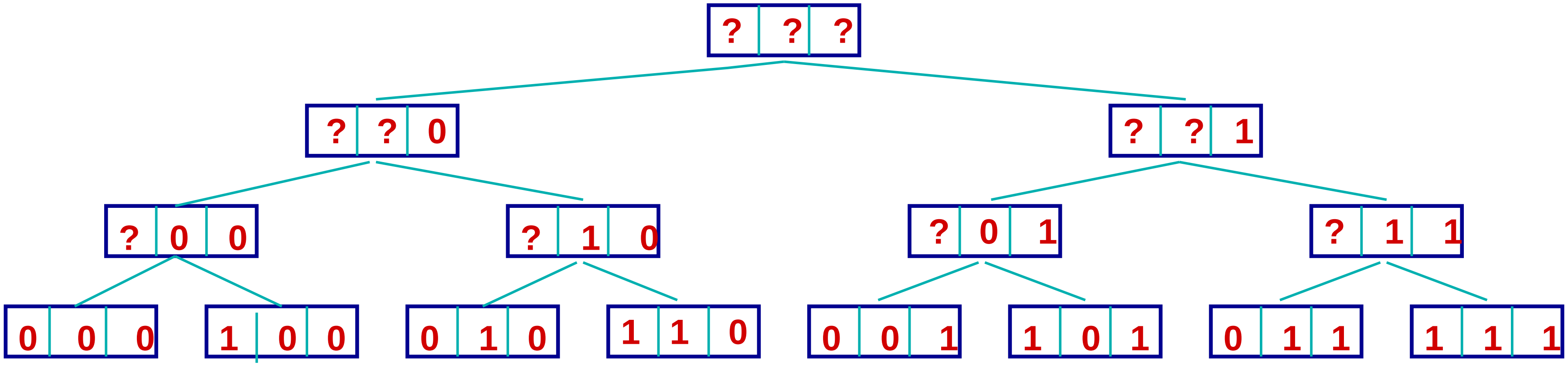
3. Testovanie požadovanej vlastnosti.

4. Odrezávanie neperspektívnych ciest.

Problém tyčí

Daných je n tyčí dĺžky $d_1, \ldots, d_n$ a číslo $D \in \mathbb{N}$. Úlohou je vybrať tyče tak, aby súčet ich dĺžok bol presne D.

1. Potenciálne riešenia sú všetky spôsoby výberu tyčí. Riešenia budeme reprezentovať ako binárne reťacze, konkrétne pomocou binárneho poľa $A[1..n]$. Hodnota $A[i] = 1$ práve ak tyč i je vybraná. Binárne reťazce vytvárajú stromovú štruktúru



$\textsc{Tyč}(m, l)$

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

  else $A[m] \leftarrow 0$

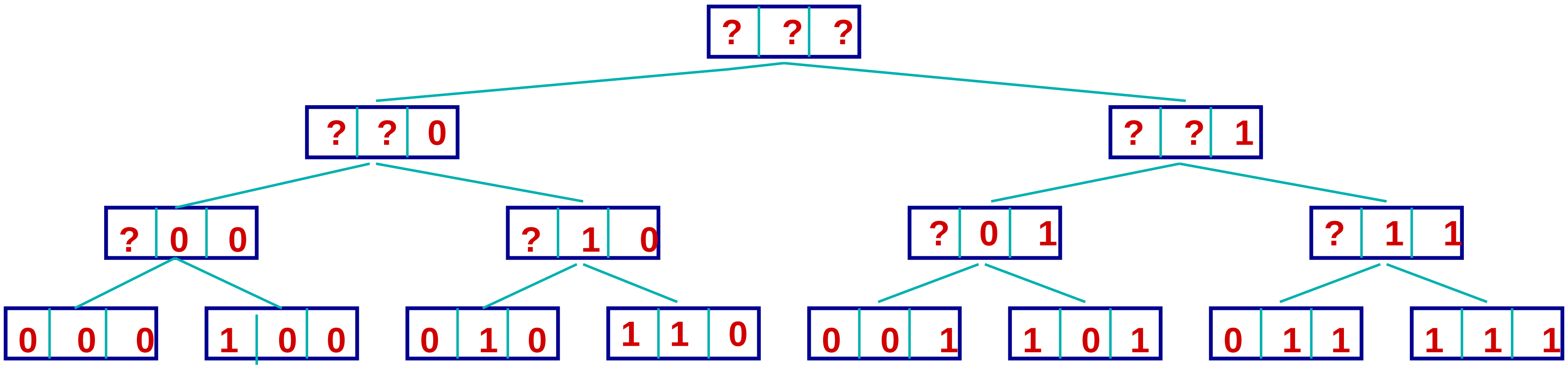
    $\textsc{Tyč}(m-1, l)$

    if $d_m \leq l$ **then** $A[m] \leftarrow 1;$

      $\textsc{Tyč}(m-1, l-d_m)$ **fi**

fi

2. Strom prehľadávame do hĺbky: pole A napĺňame zprava doľava.



3. Požadovanú vlastnosť testujeme v okamihu, keď máme vygenerované celé potenciálne riešenie (riadok 2, základ rekurzie).

4. Cesty odrezávame, keď dĺžka vybraných tyčí prekročí daný limit (riadok 5).

Poznámka: technikou dynamického programovania sa problém dá riešiť v čase $\mathcal{O}(nD)$.
Zobecnenie pre prípad, že daných je k_i *tyčí dĺžky* d_i.

Hamiltonovský cyklus

Hamiltonovský cyklus v orientovanom grafe je cyklus, ktorý navštívi každý vrchol grafu práve raz. Úlohou je nájsť v danom grafe $G = (V, H)$ Hamiltonovský cyklus.

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.

1. Potenciálne riešenia sú všetky cykly dĺžky n v grafe. Riešenia reprezentujeme ako reťazce nad abecedou $\{1, \ldots, n\}$ (pole $A[1..n]$), cyklus je $A[n] \rightarrow A[n-1] \rightarrow \cdots \rightarrow A[1] \rightarrow A[n]$.

Aby sme mohli prerezávať neperspektívne cesty, ktoré nie sú Hamiltonovské, použijeme pomocné pole $P[1..n]$; hodnota $P[i]$ je false ak aktuálna cesta už navštívila vrchol i.

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

2. Strom riešení prehľadávame do hĺbky; pole A napĺňame zprava
3. Po vygenerovaní Hamiltonovskej cesty dĺžky n testujeme (procedúra KONTROLA), či je možné cestu uzavrieť do cyklu.
4. Neperspektívne cesty odrezávame vždy keď sa nich zopakuje niektorý vrchol druhý krát (riadok 8).

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

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

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

Hamiltonovský cyklus - riešenie založené na permutáciach

Potenciálne riešenia – permutácie postupnosti $(1, \ldots, n)$. Permutácie sú uložené v poli $A[1..n]$, iniciálne $A[i] = i$ pre $i = 1, \ldots, n$

Algoritmus generovania permutácií

procedure $\textsc{Permutácie}(m)$
if $m = 1$ **then** kontrola vlastností
    else $\textsc{Permutácie}(m-1)$
        for $i = m-1$ **downto** 1 **do**
            vymeň $A[m]$ a $A[i]$
            $\textsc{Permutácie}(m-1)$
            vymeň $A[m]$ a $A[i]$
        od fi

Algoritmus doplníme o kontrolu, či vygenerovaná permutácia tvorí Ham. cyklus. Graf je zadaný maticou susednosti $M[1..n, 1..n]$.

```
for i = 1 to n do A[i] ← i od
HAMILTON_PERM(n − 1)
procedure HAMILTON_PERM(m)
if m = 0 then KONTROLA_PERM(A)
        else for i = m downto 1 do
                if M[A[m + 1], A[i]]
                  then vyměň A[m] a A[i]
                       HAMILTON_PERM(m − 1)
                       vyměň A[m] a A[i] fi
             od
fi
procedure KONTROLA_PERM(A)
if M[A[1], n] then print A fi
```

Označme $T(n)$ počet výmen, ktoré urobí počas výpočtu algoritmus $\textsc{Hamilton\_Perm}$(n). Potom

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

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

Zložitosť problému Hamiltonovského cyklu je

- $\mathcal{O}(s^n)$ – 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

- Prehľadávanie (prieskum) grafov a rôzne druhy súvislosti
- Stromy a kostry
- Optimálne sledy
- Toky v sieťach
- ....

Hledání minimální kostry grafu

Je dán souvislý neorientovaný graf $G = (V, H)$ spolu s ohodnocením hran (váhovou funkcí) $w : H \to \mathbb{R}^+$.

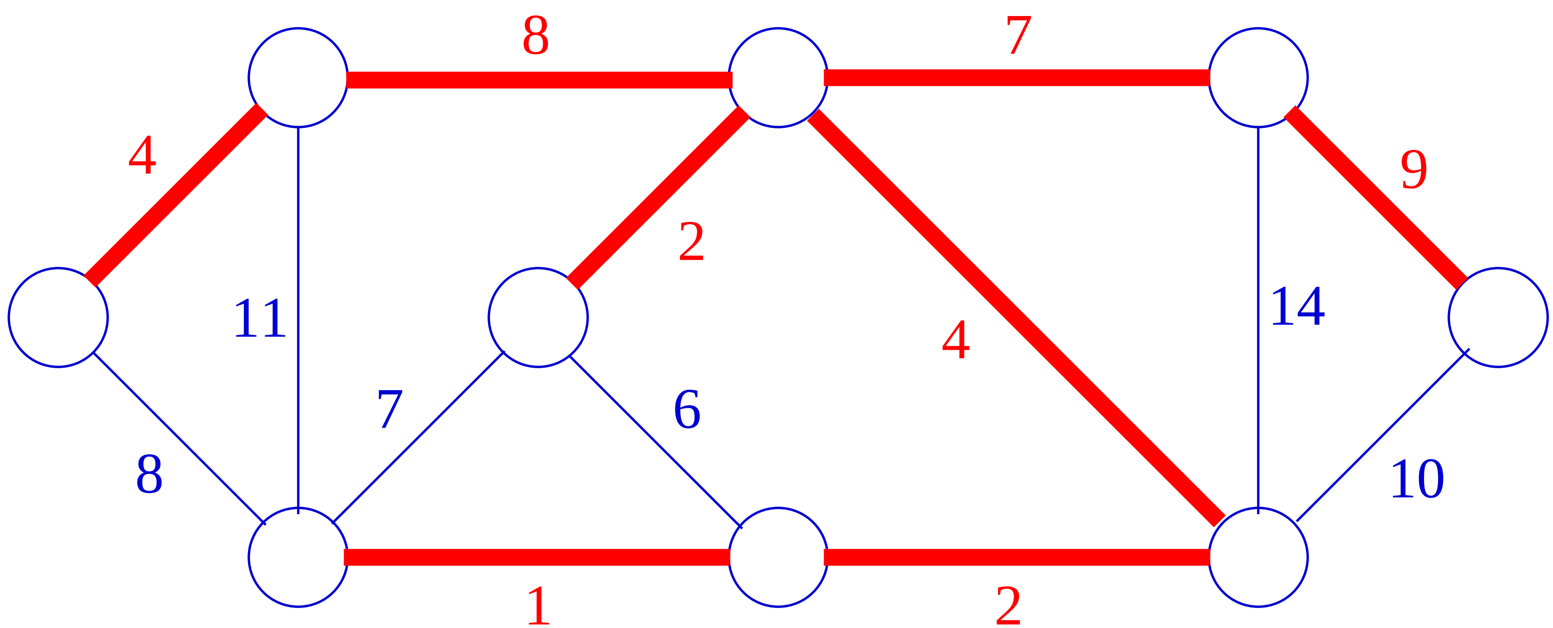
Množinu $K \subseteq H$ nazveme kostrou grafu G, jestliže je graf $G' = (V, K)$ souvislý a acyklický (kostra je vždy stromem).

Definujeme váhu kostry K předpisem

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

Minimální kostrou rozumíme kostru s minimální váhou.

Příklad:



Minimální kostra K má váhu

$$w(K) = 4 + 8 + 7 + 9 + 2 + 4 + 1 + 2 = \textcolor{red}{37}.$$

Růst minimální kostry

Inicializace : $K = \emptyset$
Do K přidáváme postupně (po jedné) hrany, dokud nedosáhneme kostry.

Invariant : K je podmnožinou nějaké minimální kostry.

Nechť K je množina hran, která je podmnožinou nějaké minimální kostry grafu G. Hranu $h \in H$ nazveme bezpečnou hranou pro množinu $K \subseteq H$, jestliže $h \notin K$ a $K \cup \{h\}$ je stále podmnožinou nějaké minimální kostry.

Obecný algoritmus pro nalezení minimální kostry

Min-kostra$((V,H),w)$

1 $K \leftarrow \emptyset$
2 **while** K netvoří kostru **do**
3   Vyber hranu h, která je bezpečná pro K
4     $K \leftarrow K \cup \{h\}$.
5 **od**
6 **return** K

Korektnost algoritmu plyne z definice bezpečných hran.
Algoritmus vždy zastaví neboť cyklus **while** proběhne vždy právě $|V|-1$ krát.

Nejobtížnější částí algoritmu je krok 3 – nalezení bezpečné hrany pro aktuální K.

Hledání bezpečných hran

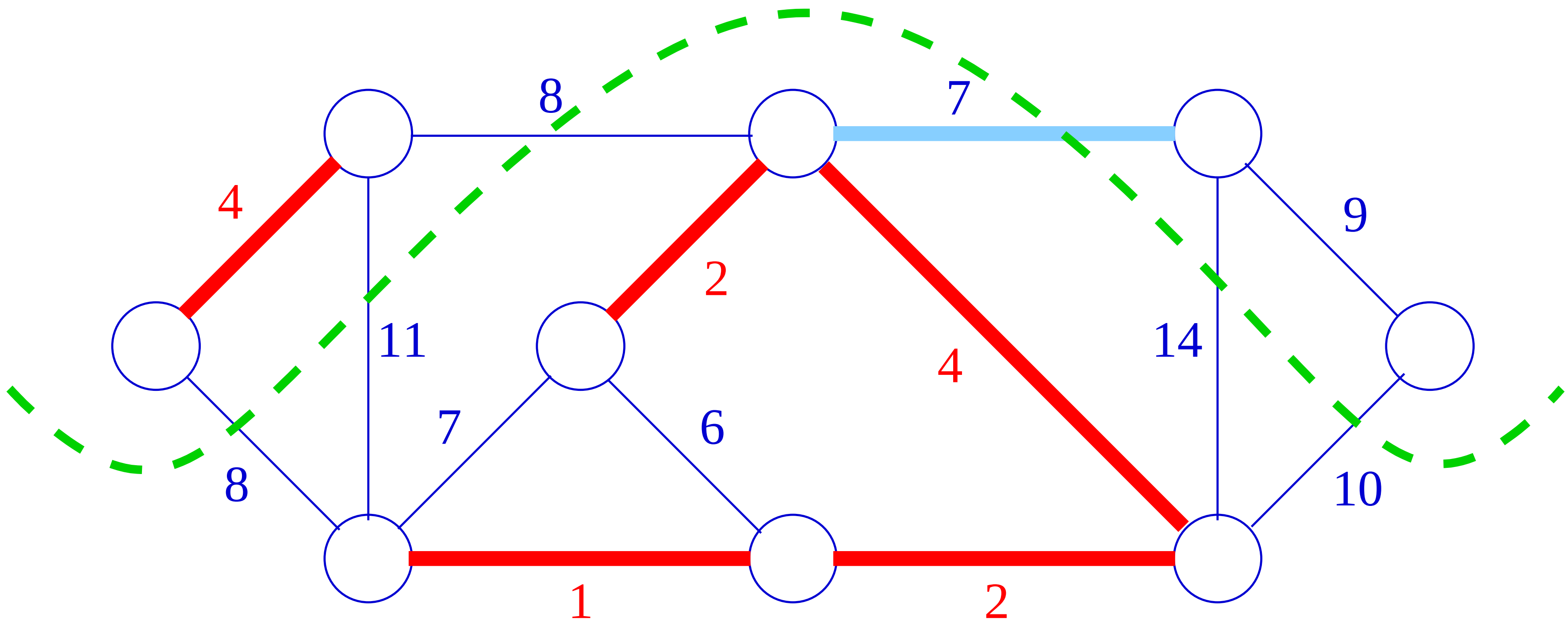
Řezem $(S, V - S)$ v grafu rozumíme libovolné rozdělení množiny vrcholů do dvou podmnožin.

Řekneme, že hrana h křižuje řez $(S, V - S)$, jestliže jeden z jejich koncových vrcholů leží v S a druhý v $V - S$.

Řez $(S, V - S)$ respektuje množinu hran K, pokud žádná hrana z množiny K nekřižuje řez $(S, V - S)$.

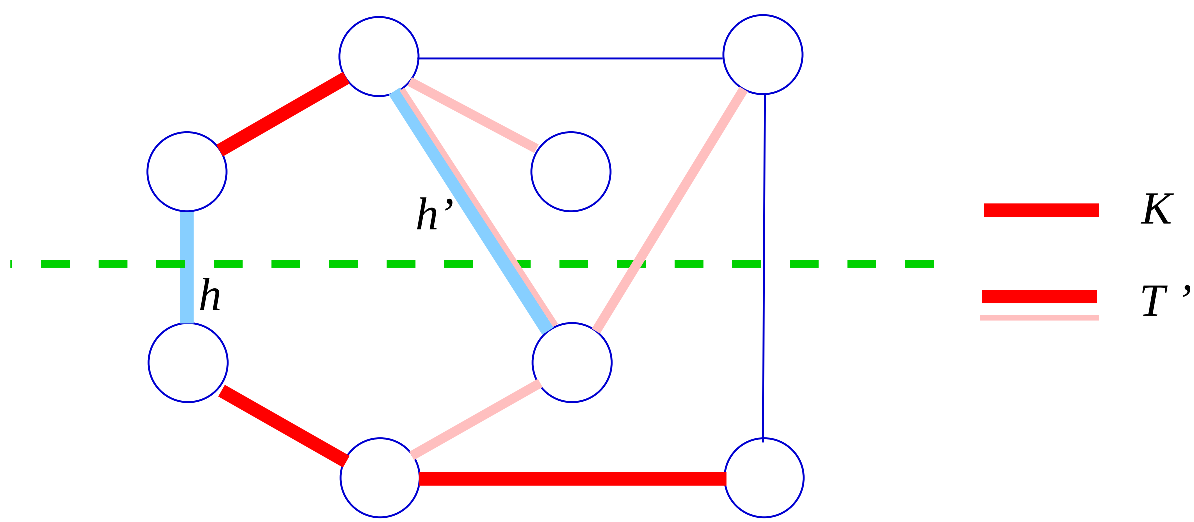
Hranu $h \in H$ nazveme lehkou hranou vzhledem k řezu $(S, V - S)$, jestliže tento řez kříží a má ze všech takových hran minimální váhu.

Příklad řezu respektujícího danou množinu hran. Příslušná lehká hrana je zvýrazněna světlemodře.



Následující lemma poskytuje návod jak hledat bezpečné hrany.

Lema 2. *Nechť $G = (V, H)$ je souvislý neorientovaný graf s vahovou funkcí $w : H \to \mathbb{R}^+$. Dále nechť $K \subseteq H$ je množina hran, která je podmnožinou nějaké minimální kostry grafu G. Uvažujme libovolný řez $(S, V - S)$, který respektuje množinu K a hranu h, která je lehkou hranou vzhledem k řezu $(S, V - S)$. Pak h je bezpečná pro množinu K.*



Dôkaz. Nechť T' je nějaká minimální kostra, jejíž podmnožinou je K. Pokud $h \in T'$, není co dokazovat. Předpokládejme tedy, že $h \notin T'$. Sestrojíme minimální kostru, která obsahuje $K \cup \{h\}$, čímž dokážeme, že hrana h je bezpečná pro K.

Množina hran $T' \cup \{h\}$ jistě obsahuje cyklus, musí tedy existovat hrana $h' \neq h$ z tohoto cyklu, která křižuje řez $(S, V-S)$. Označme $T \stackrel{\text{def}}{=} T' \cup \{h\} - \{h'\}$. Zřejmě je (V, T) souvislý graf, a protože $|T| = |T'| = |V| - 1$, je T kostra grafu G. Z minimality kostry T' plyne $w(T) \geq w(T) \Rightarrow w(h) \geq w(h')$. Naopak z lehkosti hrany h plyne $w(h) \leq w(h')$. Celkem tedy $w(h) = w(h')$, odkud máme, že T je minimální kostra. Navíc $K \cup \{h\} \subseteq T$ a tedy h je bezpečná pro K. ■

□

Kruskalův algoritmus

Množina K tvoří les. V každém kroku Kruskalova algoritmu vybíráme hranu h s minimální vahou ze všech hran spojujících různé komponenty z K.

Nechť K_1, K_2 jsou stromy, které hrana h spojuje. Je vidět, že hrana h je lehkou hranou vzhledem k řezu $(K_1, V - K_1)$, a proto je podle lemmatu 2 bezpečnou hranou pro množinu K.

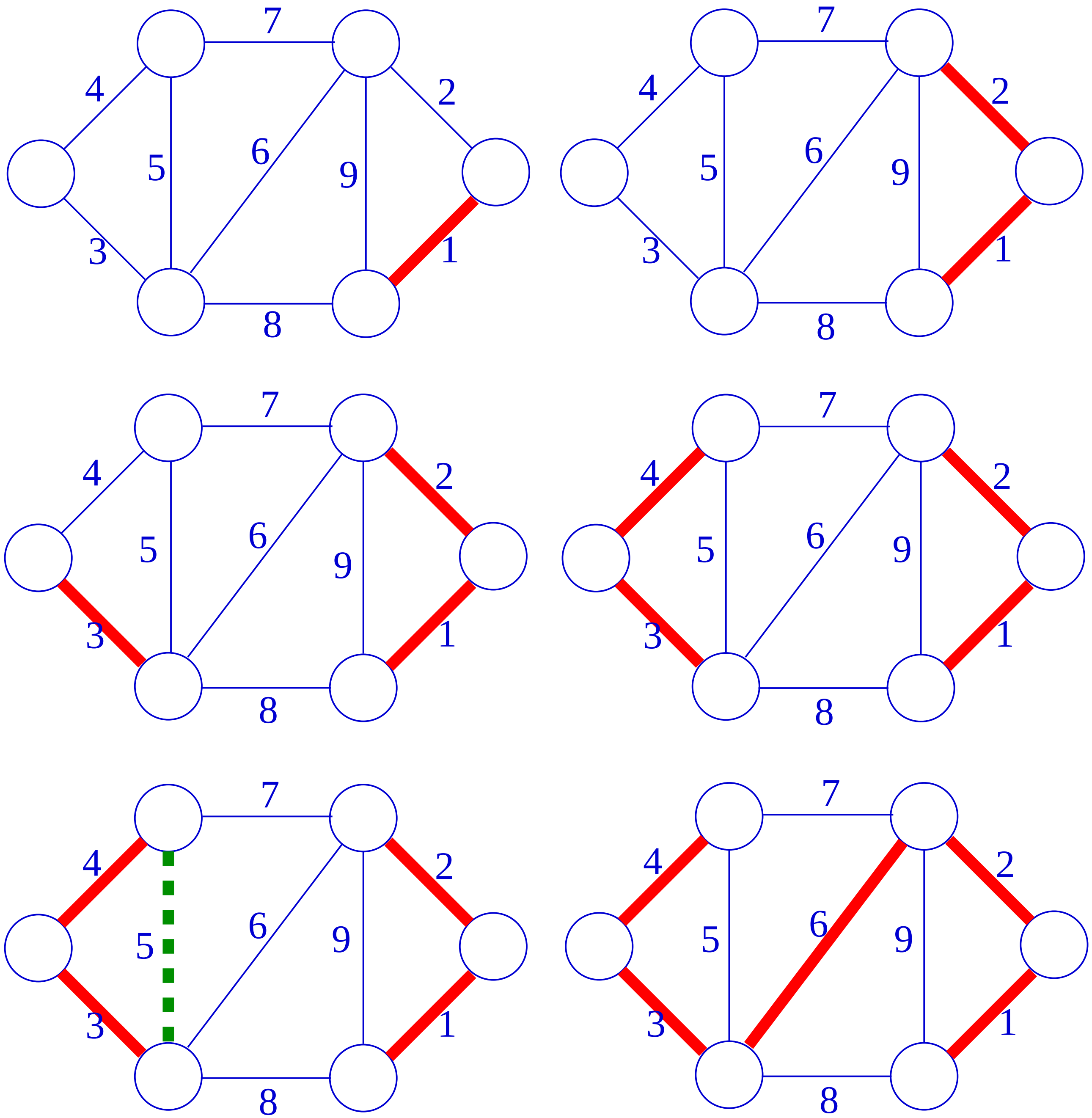
Kruskalův algoritmus je příkladem hladového algoritmu. V každém kroku vybíráme lokálně nejlepší variantu.

Implementace algoritmu využívá struktury Union-Find. V jednotlivých množinách jsou právě vrcholy ze stejné komponenty vzhledem ke K.

$\textsc{Kruskal}((V, H), w)$

1 $K \leftarrow \emptyset$
2 **foreach** $v \in V$ **do**
3     $\textsc{Make-Set}(v)$ **od**
4 Setřiď hrany podle w do neklesající posloupnosti
5 **foreach** $(u, v) \in H$ v tomto pořadí **do**
6   **if** $\textsc{Find-Set}(u) \neq \textsc{Find-Set}(v)$
7   **then** $K \leftarrow K \cup \{u, v\}$
8                 $\textsc{Union}(u, v)$ **fi od**
9 **return** K

Inicializace (řádky 1-4) má složitost $\Theta(|V|) + \Theta(|H| \log |H|)$. Cyklus 5-8 proběhne $|H|$ krát, celkově se tedy provede $\Theta(|H|)$ operací $\textsc{Find-Set}$, $\textsc{Union}$, což má složitost $\mathcal{O}(|H|\alpha(|V|))$. Celková složitost Kruskalova algoritmu je tedy $\mathcal{O}(|H| \log |H|)$.


7
4
2
5
6
9
3
1
8
7
4
2
5
6
9
3
1
8
7
4
2
5
6
9
3
1
8
7
4
2
5
6
9
3
1
8
7
4
2
5
6
9
3
1
8
7
4
2
5
6
9
3
1
8

Primův algoritmus

V případě Primova algoritmu množina K vždy tvoří jediný strom. Na začátku zvolíme libovolný vrchol r za kořen a postupně budeme přidávat další vrcholy, resp. hrany. Označme V_K množinu těch vrcholů ze kterých vede nějaká hrana $h \in K$ (v případě $K = \emptyset$ definujeme $V_K = \{r\}$).

V každém kroku vybereme nějakou hranu h, která je lehká vzhledem k řezu $(V_K, V - V_K)$ a přidáme ji do K. Lemma 2 nám opět zaručuje korektnost tohoto postupu.

Primův algoritmus je opět příkladem hladového algoritmu.

Efektivní implementace Primova algoritmu využívá strukturu Q vybavenou operacemi Extract-Min a Decrease-Key.

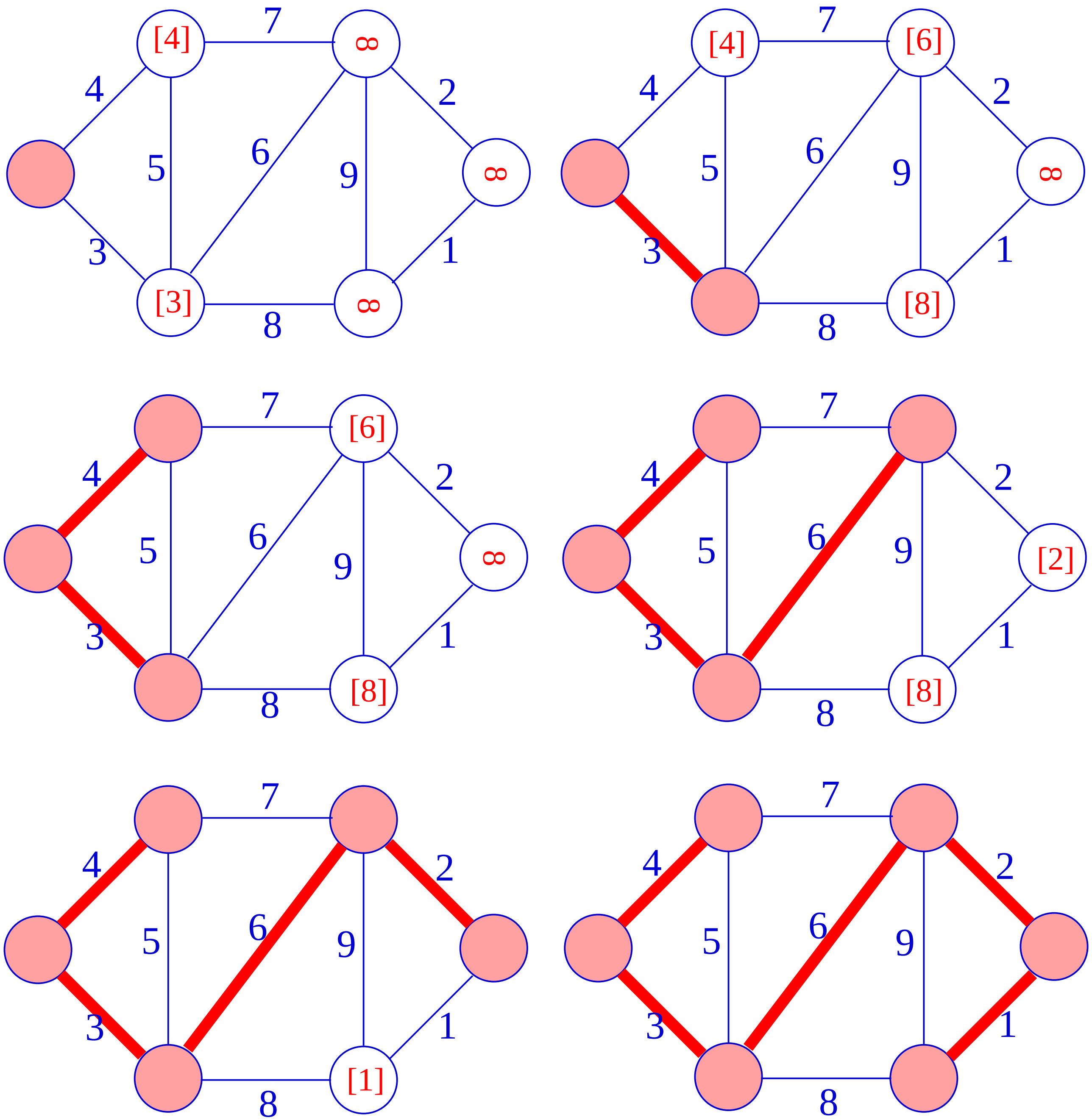
$\textsc{Prim}((V, H), w, r)$

$Q \leftarrow V$
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 $\{(\pi[v], v) \mid v \in V, v \neq r\}$

$\pi[v]$ značí otce vrcholu v v aktuálním stromě. Po skončení algoritmu je tento strom minimální kostrou.

Nejlepších výsledků dosáhneme, pokud budeme implementovat strukturu Q jako Fibonacciho haldu. Inicializace (řádky 1-5) zaberou čas $\Theta(|V|)$. Během **while** cyklu se provede $|V|$ operací $\textsc{Extract-Min}$ a $|H|$ operací $\textsc{Decrease-Key}$. V případě Fibonacciho haldy je amortizovaná složitost operace $\textsc{Extract-Min}$ $\mathcal{O}(\log|V|)$ a amortizovaná složitost operace $\textsc{Decrease-Key}$ konstantní.

Celkově se v algoritmu provede $|V|$ operací $\textsc{Extract-Min}$ a nejvýše $|E|$ operací $\textsc{Decrease-Key}$. Celková složitost algoritmu je tedy $\mathcal{O}(|V|\log|V| + |H|)$.



[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

Optimálne sledy

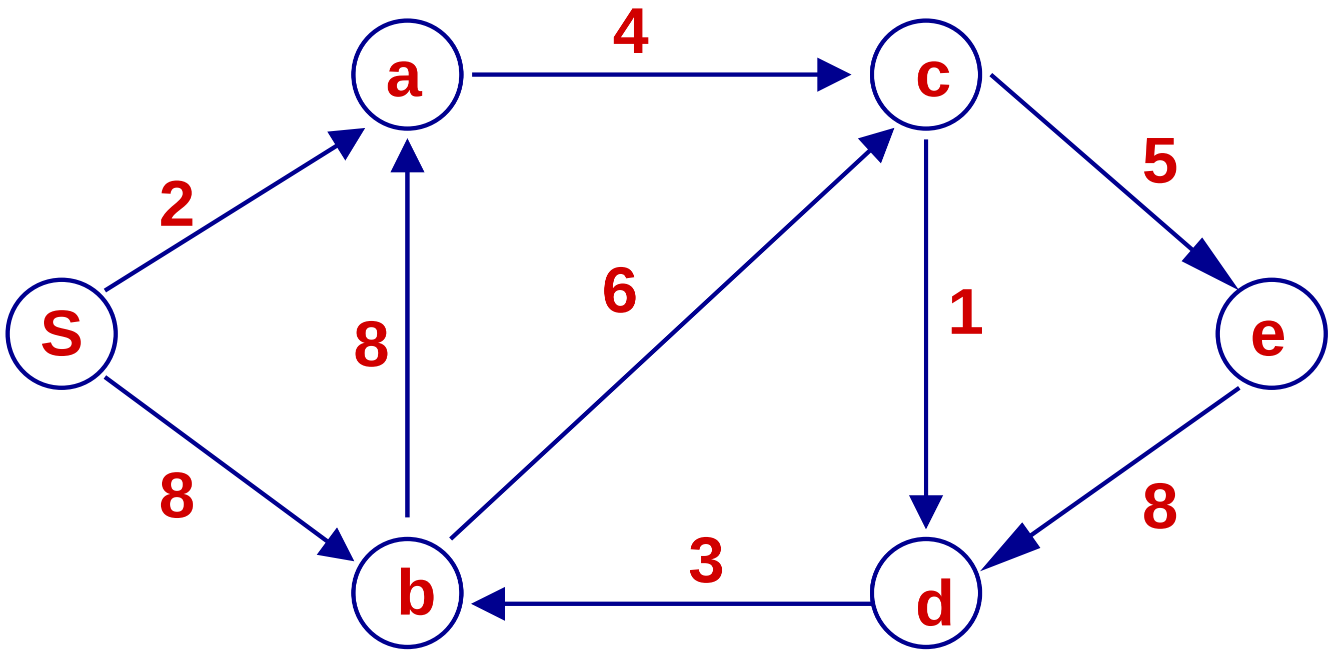
Daný je orientovaný graf $G = (V, H)$ spolu s ohodnotením hrán $w : H \to \mathbb{R}$.

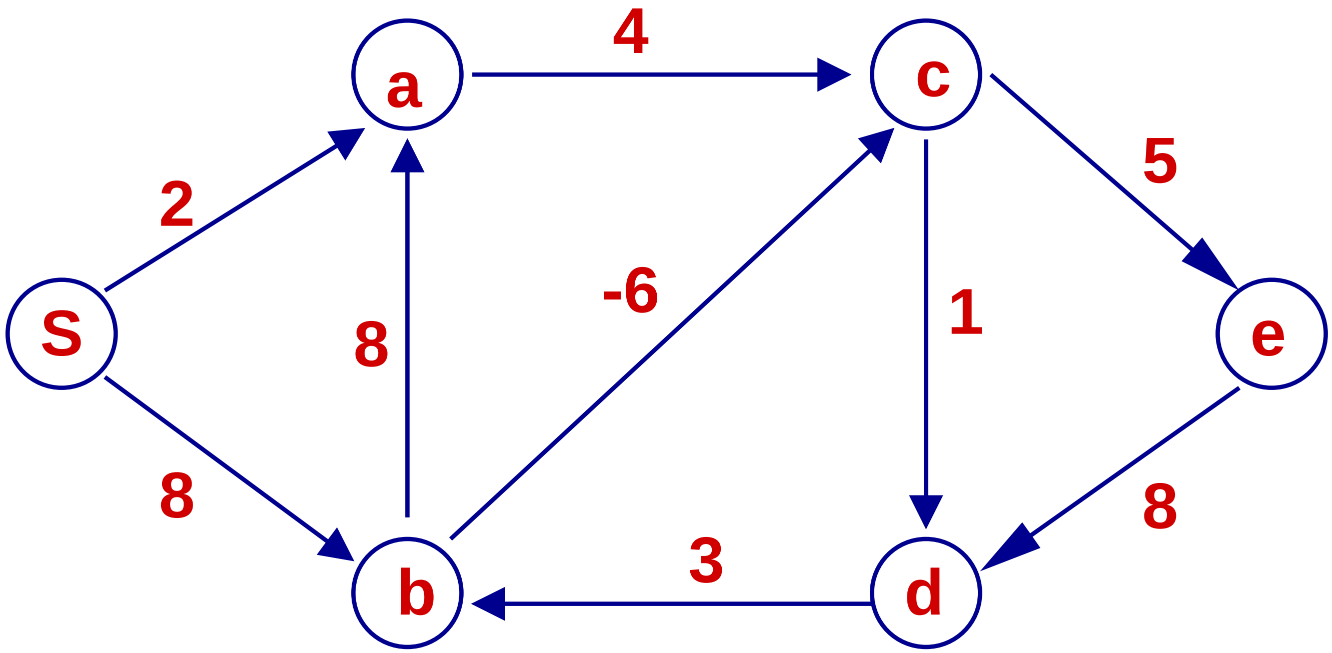
Sled $p =< v_0, v_1, \ldots, v_k >$ symbolicky značíme $v_0 \overset{p}{\leadsto} v_k$.

Dĺžkou sledu $p =< v_0, v_1, \ldots, v_k >$ je súčet dĺžok jeho hrán,

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

Dĺžka sledu $p =< v >$ neobsahujúceho žiadnu hranu je $w(p) = 0$.


a
c
S
e
b
d
4
2
8
6
1
5
8
3
8



záporný cyklus tvoria hrany dĺžky -6, 1, 3

všetky vrcholy v okrem vrcholu s majú $\delta(s, v) = -\infty$

Cyklus $c =< v_0, v_1, \ldots, v_k >$, $v_0 = v_k$, nazývame záporný cyklus práve ak jeho dĺžka $w(c) < 0$.

Ak žiadna cesta z vrcholu u do vrcholu v neobsahuje vrchol patriaci zápornému cyklu, tak dĺžka najkratšej cesty z u do v je zhodná s dĺžkou najkratšieho sledu z u do v.

Naopak, ak existuje cesta z u do v obsahujúca vrchol patriaci zápornému cyklu, tak žiaden sled z u do v nemôže byť najkratší.

Definujeme dĺžku najkratšej cesty z u do v ako

$$\delta(u,v) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{ak neexistuje cesta z } u \text{ do } v \\ -\infty & \text{ak nejaká cesta z } u \text{ do } v \text{ obsahuje} \\ & \text{vrchol patriaci zápornému cyklu} \\ \min\{w(p) \mid u \stackrel{p}{\leadsto} v\} & \text{inak} \end{cases}$$

Ak $\delta(u,v)$ je konečné, tak
najkratšia cesta z u do v je ľubovoľná cesta z u do v dĺžky $\delta(u,v)$.

Problémy

- Najkratšie cesty z daného vrcholu s do ostatných vrcholov grafu
- Najkratšie cesty zo všetkých vrcholov grafu do daného vrcholu t
- Najkratšia cesta medzi danou dvojicou vrcholov
- Najkratšie cesty medzi všetkými dvojicami vrcholov grafu

Úlohou je vypočítať dĺžku najkratšej cesty medzi určenými dvojicami vrcholov. Naviac, ak $\delta(u, v)$ je konečné, tak požadujeme, aby algoritmu našiel niektorú najkratšiu cestu z u do v.

Optimálna subštruktúra najkratších ciest

Lema 3. *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 sú založené na princípoch dynamického programovania resp. na princípe hladových algoritmov.

Najkratšie cesty z koreňa s do ostatných vrcholov grafu

- Bellman-Fordov algoritmus
- Algoritmus pre acyklické grafy
- Dijkstrov algoritmus pre grafy s nezápornými dĺžkami hrán

Reprezentácia najkratších ciest

Pre každý vrchol v udržujeme hodnotu $p[v]$ – otec vrchola v. Iniciálna hodnota je $p[v] = Nil$. Na konci výpočtu je graf $G_p = (V_p, H_p)$ indukovaný hranami $(p[v], v)$ stromom najkratších ciest z s do ostaných vrcholov grafu. Pritom
$V_p = \{v \in V \mid p[v] \neq Nil\} \cup \{s\}$
$H_p = \{(p[v], v) \in H \mid v \in V_p \setminus \{s\}\}$

Ďalej pre každý vrchol v udržujeme hodnotu $d[v]$ – dĺžka cesty z s do v v aktuálnom grafe indukovanom hranami $(p[v], v)$. Iniciálna hodnota je pre $v \neq s$ rovná $d[v] = \infty$ (v indukovanom grafe neexistuje cesta z s do v) a $d[s] = 0$. Na konci výpočtu je $d[v] = \delta(s, v)$.

Bellman-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 $d[v]$ (operácia $\textsc{Relaxácia}$).

Algoritmus vráti hodnotu true práve ak graf neobsahuje cyklus zápornej dĺžky dosiahnuteľný z koreňa s.

BELLMAN-FORD(G, w, s)

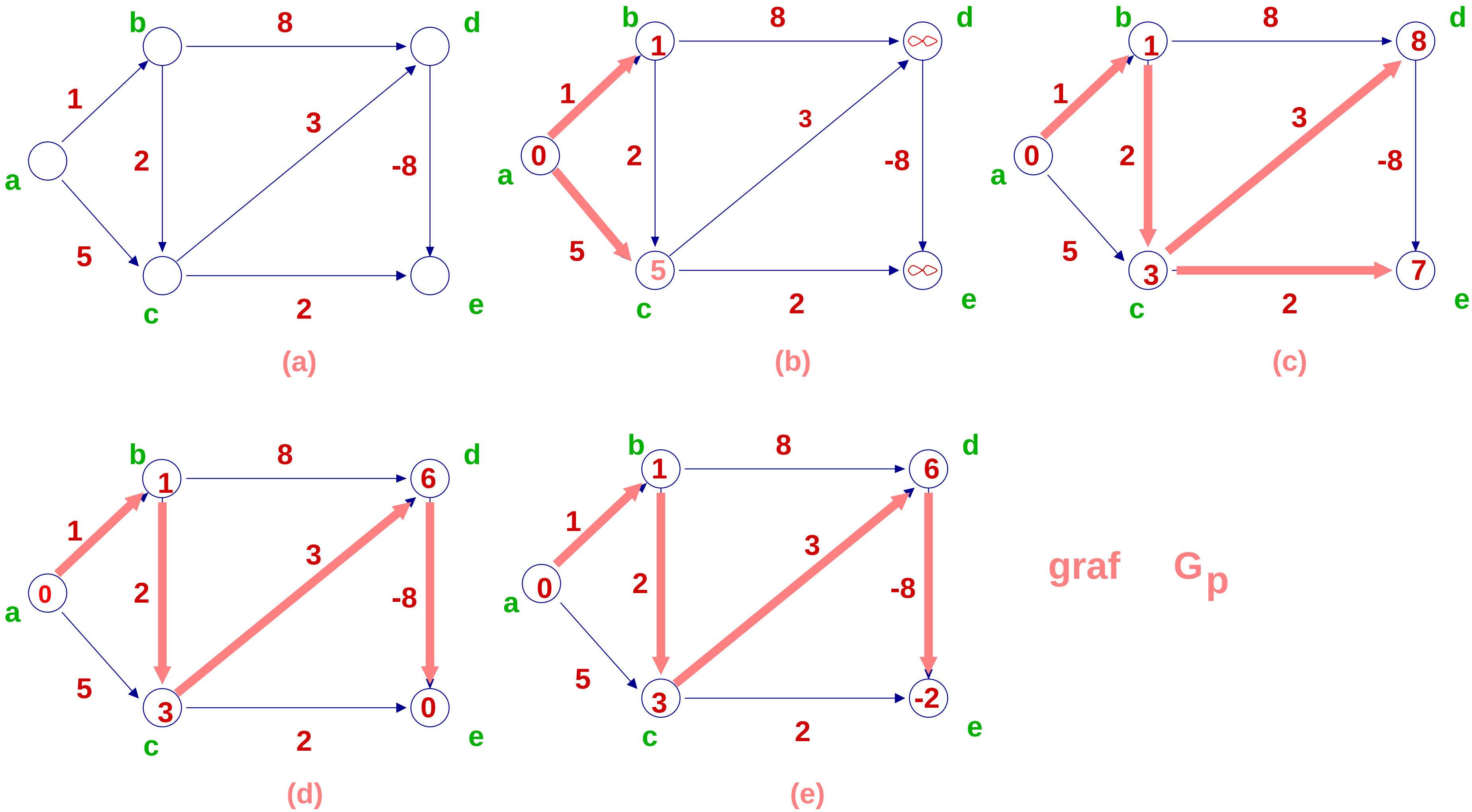
INICIALIZÁCIA(G, s)
for $i = 1$ **to** $|V| - 1$ **do**
    for každú hranu $(u, v) \in H$ **do** RELAXÁCIA(u, v, w) **od od**
for každú hranu $(u, v) \in H$ **do**
    if $d[v] > d[u] + w(u, v)$ **then return** false **fi od**
return true

INICIALIZÁCIA(G, s)

for každý vrchol $v \in V$ **do**
    $d[v] \leftarrow \infty;\ p[v] \leftarrow Nil$ **od**
$d[s] \leftarrow 0$

RELAXÁCIA(u, v, w)

if $d[v] > d[u] + w(u, v)$
  then $d[v] \leftarrow d[u] + w(u, v);\ p[v] \leftarrow u$ **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).

Korektnosť algoritmu Bellman-Ford

Lema 4. *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 $\textsc{Relaxácia}(v_{k-1}, v_k, w)$ 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

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

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.

$\square$

Lema 5. *Nech v grafe G neexistuje záporný cyklus dosiahnuteľný z koreňa s. Predpokladajme, že pre G voláme* $\textsc{Inicializácia}(G, s)$ *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 4 je G_p strom. Zostáva ukázať, že pre každý vrchol $v \in V_p$ je jediná cesta $s \overset{p}{\leadsto} v$ 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 $w(p) = \delta(s, v_k)$.

□

Lema 6. *Nech v grafe G neexistuje záporný cyklus dosiahnuteľný z koreňa s. Po tom po $|V| - 1$ iteráciach cyklu 2-3 algoritmu* $\textsc{Bellman-Ford}(G, w, s)$ *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 2-3 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.

$\square$

Veta. *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 6 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 5 nevráti hodnotu false.

Naopak, nech v G existuje záporný cyklus $c =< v_0, v_1, \ldots, v_k >$, kde $v_0 = v_k$, dosiahnuteľný z s. Predpokladajme, že algoritmus vráti true. Potom pre všetky $i = 1, 2, \ldots, k$ 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 Bellman-Fordovho algoritmu

Inicializácia grafu má zložitosť $\Theta(|V|)$. Cyklus 2–3 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. Príkladom sú algoritmy pre acyklické grafy, pre grafy s malým ohodnotením hrán a pre grafy s nezáporným ohodnotením hrán.

Algoritmus pre acyklické grafy

Optimálne poradie relaxácie hrán v Bellman-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.

Topologické utriedenie pre orientovaný acyklický graf $G = (V, H)$ je lineárne usporiadanie vrcholov grafu také, že ak graf obsahuje hranu (u, v), tak u predchádza v v usporiadaní. Topologické utriedenie grafu môžeme vypočítať tak, že graf prehľadávame do hĺbky a vrcholy usporiadame v poradí, v akom sa v prehľadávaní ukončil ich prieskum.

NAJKRATŠIE-CESTY-V-ACYKLICKOM-GRAFE($(V, H), w, s$)

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

Každá hrana grafu sa relaxuje iba raz, celková zložitosť algoritmu je $\Theta(n + m)$.

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.

$\textsc{Dijkstra}(G, w, s)$

$\textsc{Inicializácia}(G, s)$
$S \leftarrow \emptyset;\ Q \leftarrow V$
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

Veta 1. *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 $s \overset{p_1}{\leadsto} x \longrightarrow y \overset{p_2}{\leadsto} u$ 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 $\delta(s, y) \le \delta(s, u)$. Pretože ale vrchol u bol vybraný do S, tak zároveň bezprostredne pred zaradením u do S platí $d[u] \le d[y] = \delta(s, y)$. Spojením dostávame $\delta(s, u) \le d[u] \le \delta(s, y) \le \delta(s, u)$ 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 $\textsc{Extract\_Min}$) a aktualizácie hodnôt $d[v]$ pre vrcholy susediace s u (operácia $\textsc{Decrease\_Key}$).

Ak hodnoty $d[v]$ sú uložené v poli, tak $\textsc{Extract\_Min}$ má zložitosť $\mathcal{O}(|V|)$ a $\textsc{Decrease\_Key}$ má konštantnú zložitosť. Celková zložitosť Dijkstrovho algoritmu je $\mathcal{O}(|V|^2 + |H|) = \mathcal{O}(|V|^2)$.

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. Celková zložitosť je $\mathcal{O}(|V| \log |V| + |H|)$.

Najkratšie cesty medzi všetkými dvojicami vrcholov grafu

- algoritmus založený na násobení matíc – $\theta(|V|^3 \log |V|)$
- Floyd-Warshallov algoritmus – $\theta(|V|^3)$
- Johnsonov alg. pre riedke grafy – $\mathcal{O}(|V|^2 \log |V| + |V| \cdot |H|)$

Prvé dva algoritmy predpokladajú, že graf je reprezentovaný maticou susednosti. Johnsonov algoritmus využíva Bellman-Fordov a Dijkstrov algoritmus.

Algoritmus založený na násobení matíc

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

Navrhovaný 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 $i \overset{p'}{\leadsto} k \to j$, kde p' má $m - 1$ hrán. Podľa Lemy 3 je p' 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

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

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 $\textsc{Extend}$ je asociatívne a preto môžeme vypočítať $L^{(n-1)}$ efektívnejšie:

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

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 $L^{(m)}$

Floyd-Warshallov algoritmus

Daný je orientovaný graf $G = (V, H)$, $V = \{1, 2, \ldots, n\}$, s ohodnotením hrán $w : H \rightarrow \mathbf{R}$ a bez záporných cyklov. Predpokladáme, že graf je zadaný $n \times n$ maticou susednosti $W = (w_{ij})$.

Nech $p = < v_1, v_2, \ldots, v_l >$, $l \geq 2$, je cesta v grafe. Vnútorné vrcholy cesty p sú vrcholy $\{v_2, \ldots, v_{l-1}\}$.

Pre ľubovoľnú dvojicu vrcholov $i, j \in V$ uvažujme všetky cesty z i do j, ktorých vnútorné vrcholy sú z množiny $\{1, 2, \ldots, k\}$, nech p je z nich najkratšia. Floydov - Warshallov algoritmus využíva vzťah medzi touto množinou ciest z i do j 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 3 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 s vnútornými vrcholmi z množiny $\{1, 2, \ldots, k\}$. Platí

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

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.

Presnejšie, 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 $G = (V, H)$ s ohodnotením $w : H \rightarrow \mathbb{R}$.

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

Lema 7. *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:

$$w(p) = \delta(v_0, v_k) \text{ vtedy a len vtedy ak } \widehat{w}(p) = \widehat{\delta}(v_0, v_k)^1.$$

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

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

Dôkaz. Nech $p =< v_0, v_1, \ldots, v_k >$. 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 $c =< v_0, v_1, \ldots, v_k >$, $v_0 = v_k$, je cyklus. Podobne odvodíme $\widehat{w}(c) = w(c) + h(v_0) - h(v_k) = w(c)$. Z toho plynie druhé tvrdenie Lemy. $\square$

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.

1. Skonštruujeme $\overline{G} = (\overline{V}, \overline{H})$ s ohodnotením $\overline{w} : \overline{H} \to \mathbb{R}$, kde
   - $\overline{V} = V \cup \{s\}$, $s \notin V$
   - $\overline{H} = H \cup \{(s,v) \mid v \in V\}$
   - $\overline{w}(u,v) = w(u,v)$ 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.

2. Definujeme

- $$h(v) \stackrel{\text{def}}{=} \delta(s,v) \text{ pre } v \in \overline{V}$$

  Pre každú $(u,v) \in \overline{H}$ platí: $h(v) \le h(u) + w(u,v)$

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

  Pre každú $(u,v) \in \overline{H}$ platí: $\widehat{w}(u,v) \ge 0$

JOHNSON(G)

skonštruuj $\overline{G} = (\overline{V}, \overline{H})$podľa bodu 1.
if BELLMAN-FORD$(\overline{G}, \overline{w}, s) = false$
    then graf má záporný cyklus
    else foreach vrchol $v \in V$
            do $h(v) \leftarrow \delta(s, v)$ (vypočítané alg. BELL.-FORD) **od**
        foreach hranu $(u, v) \in H$
            do $\widehat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$ **od**
        foreach vrchol $u \in V$
            do DIJKSTRA$(G, \widehat{w}, u)$
            foreach vrchol $v \in V$
                do $d_{uv} \leftarrow \widehat{\delta}(u, v) + h(v) - h(u)$
            od od
fi
return $D = (d_{uv})$

Zložitosť Johnsonovho algoritmu

zložitosť algoritmu Bellman-Ford $\approx$ $\mathcal{O}(|V| \cdot |H|)$

$+$

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

$=$

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

Pre riedke grafy je Johnsonov algoritmus efektívnejší než Floyd-Warshallov algoritmus.

Toky v sietiach

Sieť $G = (V, H)$ je orientovaný graf, ktorého každá hrana $(u, v) \in H$ má nezápornú kapacitu (priepustnosť) $c(u, v) \geq 0$. Ak $(u, v) \notin H$, kladieme $c(u, v) = 0$. Predpokladáme, že v sieti sú vyznačené dva vrcholy — zdroj s a cieľ t a že všetky vrcholy grafu ležia na nejakej ceste z s do t.

Tok v sieti G je funkcia $f : V \times V \to \mathbb{R}$ spĺňajúca

Kapacitné ohraničenia $f(u, v) \leq c(u, v)$ pre všetky $u, v \in V$

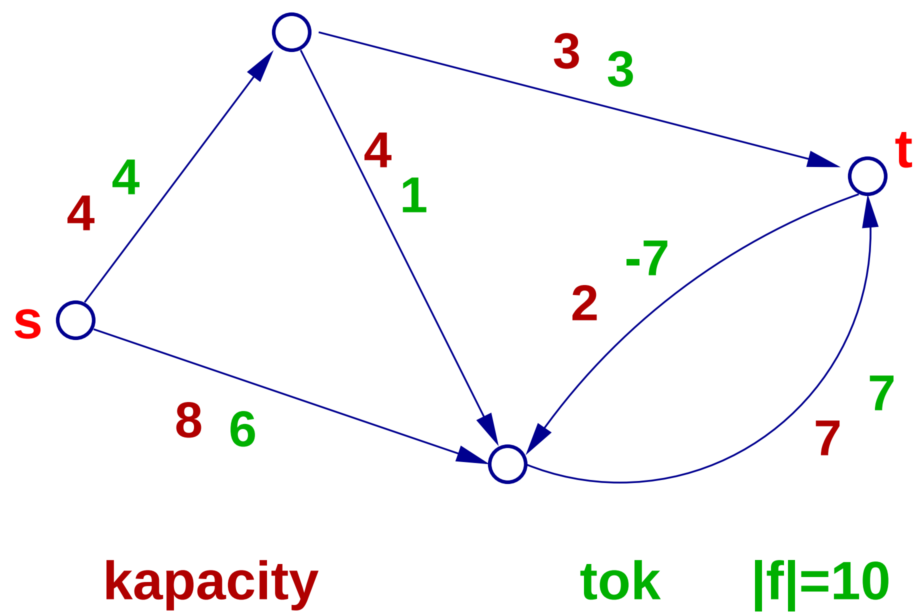
Podmienky symetrie $f(u, v) = -f(v, u)$ pre všetky $u, v \in V$

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

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

Hodnota $f(u, v)$ sa nazýva tok z u do v. Podmienky kontinuity vyjadrujú, že to, čo do vrcholu priteká, to z neho aj odteká.
Hodnota toku f je $|f| \stackrel{\text{def}}{=} \sum_{v \in V} f(s, v)$.

Problém maximálneho toku v sieti je nájsť tok maximálnej hodnoty.



Značenie: pre $X, Y \subseteq V$ je $f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)$.
Podobne pre kapacitu c.

Ford-Fulkersonova metóda

Iteratívny algoritmus. 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.

Varianty algoritmu 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
```

Reziduálna sieť

Nech G je graf a f tok v G.

Reziduálna kapacita hrany (u, v) je

$$c_f(u, v) \stackrel{\text{def}}{=} c(u, v) - f(u, v).$$

Reziduálna sieť indukovaná grafom G a tokom f je $G_f = (V, H_f)$,

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

Zlepšujúca cesta p je cesta z s do t v G_f.

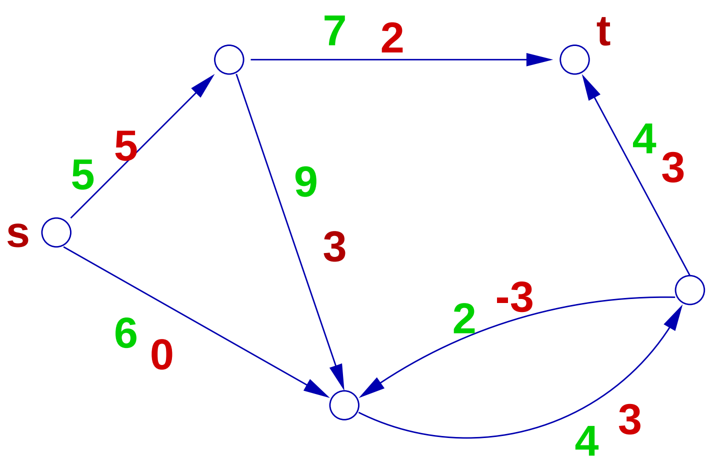
Reziduálna kapacita cesty p je

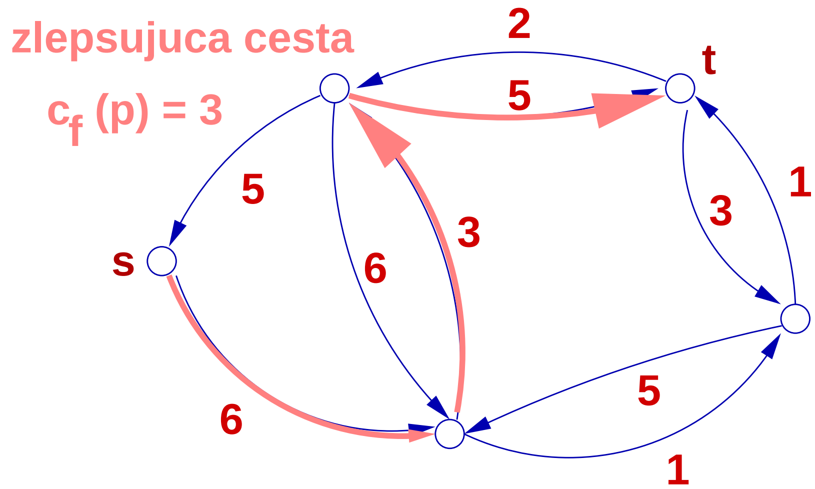
$$c_f(p) \stackrel{\text{def}}{=} \min\{c_f(u, v) \mid (u, v) \text{ leží na } p\}.$$

Ford-Fulkersonov algoritmus

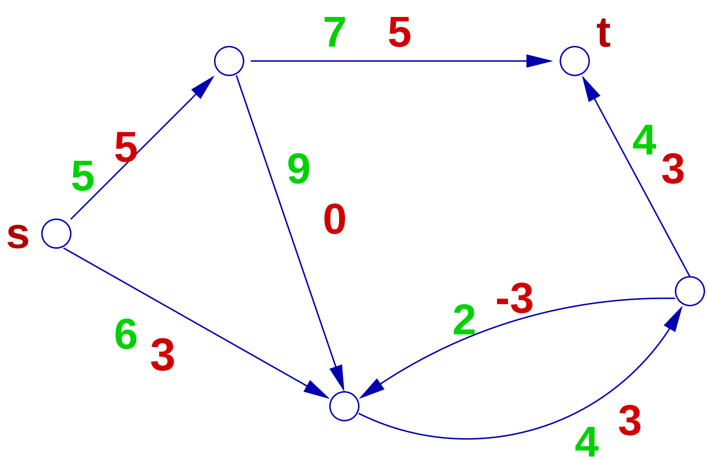
FORD_FULKERSONOV_ALGORITMUS

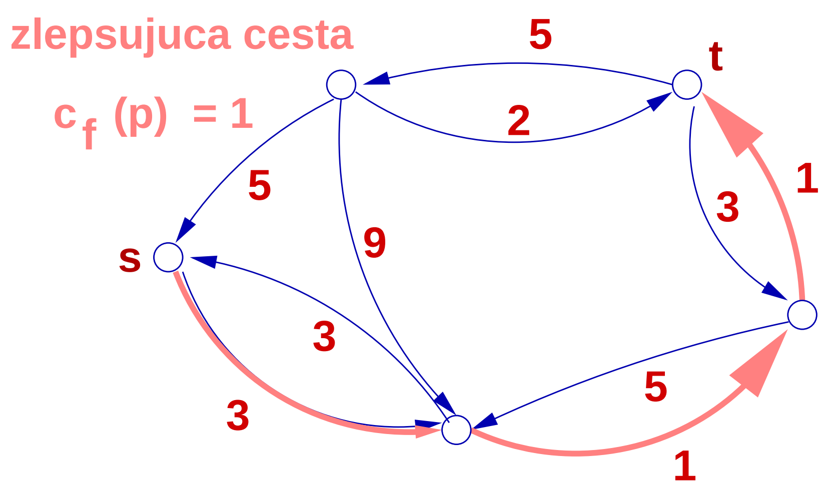
for každú hranu $(u, v) \in H$ **do**
  $f[u, v] \leftarrow 0;\ f[v, u] \leftarrow 0$
od
while existuje zlepšujúca cesta p **do**
  $c_f(p) \leftarrow \min\{c_f(u, v) \mid (u, v)$ leží na $p\}$
  for každú hranu (u, v) na p **do**
    $f[u, v] \leftarrow f[u, v] + c_f(p)$
    $f[v, u] \leftarrow -f[u,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





kapacity tok |f| = 5 rezidualna siet rezidualne kapacity





kapacity tok |f| = 8 rezidualna siet rezidualne kapacity

Korektnosť Ford-Fulkersonovej metódy

Lema 8. *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*

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

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

Naviac $\tilde{f} = f + f_p$ je tok v G a jeho hodnota je $|\tilde{f}| > |f|$.

Dôkaz. Overíme, že f_p v G_f a $\tilde{f}$ v G spĺňajú kapacitné ohraničenia a podmienky symetrie a kontinuity.

□

Rez (S,T) v sieti G je rozdelenie V na S a T také, že $s \in S$ a $t \in T$. Kapacita rezu je $c(S,T)$. Minimálny rez je rez, ktorého kapacita je spomedzi všetkých rezov grafu najmenšia.

Lema 9. *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.

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

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

Veta 2. *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 8.

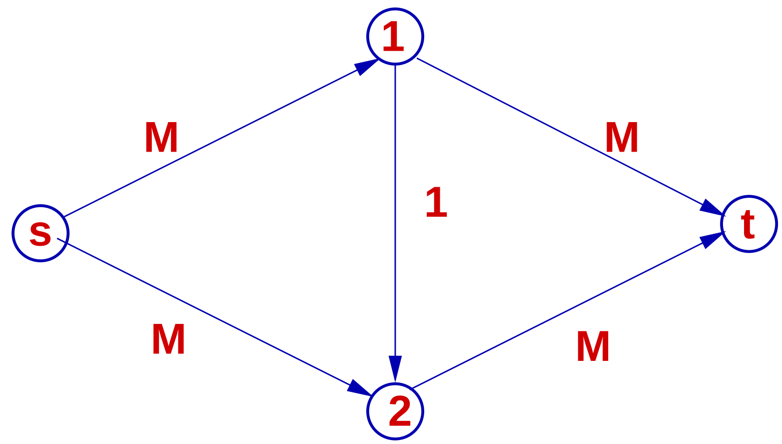
$(2) \Rightarrow (3)$ Nech G_f nemá žiadnu zlepšujúcu cestu. Definujme $S = \{v \in V \mid$ existuje cesta z s do v v $G_f\}$ a $T = V \setminus S$. Rozdelenie (S,T) je rezom v G_f. Pre každú dvojicu vrcholov u, v takú, že $u \in S$ a $v \in T$ platí $f(u,v) = c(u,v)$ (v opačnom prípade by $(u,v) \in H_f$) a teda $f(S,T) = c(S,T)$. Podľa Lemy 9 platí $|f| = f(S,T)$.

$(3) \Rightarrow (1)$ Podľa Lemy 9 pre každý rez platí $|f| \le c(S,T)$. Rovnosť $|f| = c(S,T)$ preto implikuje maximalitu toku. $\square$

Zložitosť Ford-Fulkersonovho algoritmu

Ak v sieti G s celočíselnými kapacitami je $|f^*|$ hodnota maximálneho toku, tak počet iterácií cyklu 2–6 je nanajvýš $|f^*|$.
Zložitosť jednej iterácie je rovná zložitosti nájdenia cesty z s do t a je $\mathcal{O}(|H|)$.
Celková zložitosť je $\mathcal{O}(|f^*| \cdot |H|)$.

Príklad grafu, pre ktorý zložitosť Ford-Fulkersonovho algoritmu je $|f^*| \cdot \mathcal{O}(|H|)$.



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 $|f| = 3$. ...

V prípade, že kapacity hrán sú iracionálne čísla, konečnosť Ford-Fulkersonovej metódy nie je zaručená.

Varianty Ford-Fulkersonovej metódy

Líšia sa v spôsobe, akým sa vyberá zlepšujúca cesta.

Algoritmus Edmonds-Karp Tok zväčšujeme vždy po najkratšej zlepšujúcej ceste (dĺžka cesty je rovná počtu hrán cesty). Zložitosť algoritmu je $\mathcal{O}(|V| \cdot |H|^2)$ pre grafy s ľubovoľnými kapacitami.

Algoritmus najširších ciest Tok zväčšujeme vždy po zlepšujúcej ceste s maximálnou reziduálnou kapacitou. Zložitosť algoritmu je $\mathcal{O}(|V|^2 \cdot |H| \ln |f^*|)$ pre grafy s celočíslenými kapacitami.

Algoritmus zjemňovania stupnice

Ďalšie metódy: metóda "push-relabel" vedie k algoritmu zložitosti $\mathcal{O}(|V|^3)$.

Varianty problému maximálneho toku

- Siete s násobnými zdrojmi a cieľmi
- Najlacnejší maximálny tok
- Viacproduktové toky
- Najpočetnejšie párovania

Vyhľadávacie algoritmy

Algoritmy pre vyhľadávanie, porovnávanie a editáciu reťazcov.

Problémy

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

Vyhľadávanie vzorky v texte[2]

Je daný text T a vzorka[3] P – reťazce nad abecedou Σ. Úlohou je vyhľadať všetky výskyty vzorky v texte. Text je daný ako pole $T[1..n]$, vzorka ako $P[1..m]$.

Hovoríme, že vzorka P sa vyskytuje v texte T s posunom s ak $0 \leq s \leq n - m$ a $T[s + 1..s + m] = P[1..m]$ (tj. $T[s + j] = P[j]$ pre $1 \leq j \leq m$). Číslo s uvedených vlastností sa nazýva platným posunom pre text T a vzorku P.

Problém vyhľadávania vzorky môžeme formulovať ako úlohu nájsť pre dané T a P všetky platné posuny.

[2] String matching
[3] Pattern

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	$\mathcal{O}(m\vert\Sigma\vert)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$
Boyer-Moore		$\mathcal{O}((n-m+1)m)$

Algoritmy Karp-Rabin a Boyer-Moore majú výrazne lepšiu priemernú zložitosť

Úplné prehľadávanie

ÚPLNÉ_PREHĽADÁVANIE(T, P)

1 **for** $s = 0$ **to** $n - m$ **do**
2 **if** $P[1..m] = T[s+1..s+m]$
3 **then** print "s je platný posun **fi**
4 **od**

Zložitosť: cyklus sa vykoná $n - m + 1$ krát
v každom cykle sa na riadu 2 vykoná m porovnaní.

Spolu $\mathcal{O}((n - m + 1)m)$

Algoritmus Karp-Rabin

Predpokladajme, že $\Sigma = \{0, 1, \ldots, 9\}$[4]. Každý reťazec nad abecedou Σ môžeme chápať ako číslo zapísané v desiatkovej sústave.

Označme p číslo zodpovedjúce 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 $\Theta(m)$)

$$p = P[m]+10(P[m-1]+10(P[m-2]+\cdots+10(P[2]+10P[1])\ldots))$$

Výpočet čísla t_0 – podobne ako p (čas $\Theta(m)$)

Výpočet čísel $t_1, \ldots, t_{n-m}$ (čas $\Theta(n-m)$)

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$$

[4] zobecnenie pre $\Sigma = \{0, 1, \ldots, d\}$ je priamočiare

Algoritmus Karp-Rabin so zvyškami

Algoritmus Karp-Rabin sa nedá použiť ak čísla p, 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 $t_s \equiv p(\mod q)$. Číslo s, pre ktoré platí uvedená rovnosť je len potenciálnym posunom, jeho platnosť sa musí overiť porovnaním príslušných reťazcov.

Zložitosť predspracovania je opäť $\Theta(m)$, 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.

$\textsc{Finite\_Automaton\_Matcher}(T, A)$

```
q ← 0
for i = 1 to n do
    q ← δ(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: $P_q = P[1] \cdots P[q]$, $T_q = T[1] \cdots [q]$

Definujeme sufixovú funkciu $\sigma : \Sigma^* \to \{0, 1, \ldots, m\}$ kde $\sigma(x)$ je dĺžka najdlhšieho prefixu vzorky P, ktorý je sufixom slova x.

Napr. pre $P = popapopa$ *je* $\sigma(\varepsilon) = 0$, $\sigma(papap) = 1$, $\sigma(papop) = 3$ *a* $\sigma(popapo) = 6$.

Konečný automat pre P je $A = (\{0, \ldots, m\}, \Sigma, \delta, \{0\}, \{m\})$, kde prechodová funkcia δ je definovaná predpisom

$$\delta(q, a) \stackrel{\text{def}}{=} \sigma(P_q\, a)$$

Algoritmus pre výpočet prechodovej funkcie automatu

$\textsc{Automat}(P, \Sigma)$

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_k je sufixom $P_q\ a$
    $\delta(q, a) \leftarrow k$
  od od
return δ

Zložitosť konštrukcie automatu: je $\mathcal{O}(m^3|\Sigma|)$; existuje efektívnejšia procedúra zložitosti$\mathcal{O}(m|\Sigma|)$.

Korektnosť algoritmu

Veta. *Ak A je automat pre vzorku P a T je text, tak pre $i = 0, 1, \ldots, n$ platí*
$$\hat{\delta}(T_i) = \sigma(T_i).$$

Dôkaz. Indukciou vzhľadom k i.

1. $T_0 = \varepsilon$ a preto $\hat{\delta}(T_0) = 0 = \sigma(T_0)$.

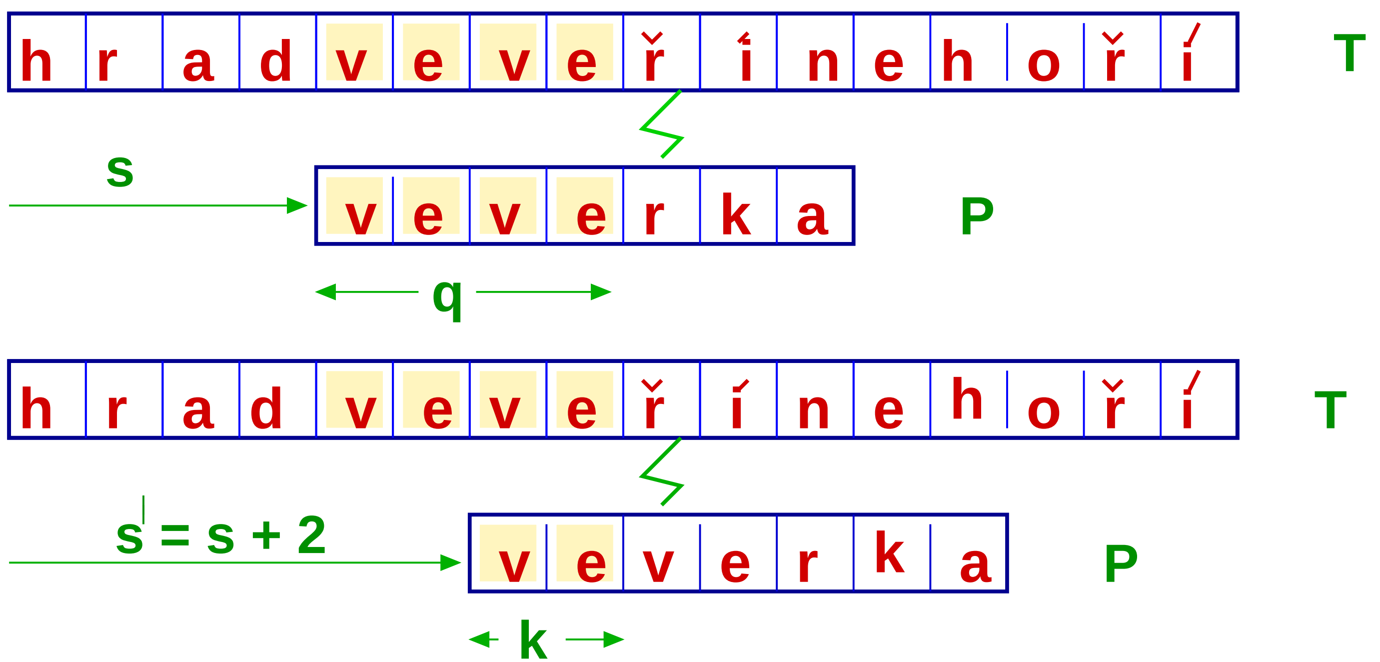
2. Indukčný predpoklad – platnosť tvrdenia pre i

$$\begin{aligned}
\hat{\delta}(T_{i+1}) &= \hat{\delta}(T_i a) && \text{nech } T[i+1] = a \\
&= \delta(\hat{\delta}(T_i), a) && \text{z definície } \hat{\delta} \\
&= \delta(q, a) && \text{nech } \hat{\delta}(T_i) = q \\
&= \sigma(P_q\ a) && \text{z definície } \sigma \\
&= \sigma(T_i\ a) && \text{indukčný predpoklad} \\
&= \sigma(T_{i+1}) && \text{z definície } T_{i+1}
\end{aligned}$$

□

Algoritmus Knuth-Morris-Pratt

Vychádza z podobného princípu ako konečné automaty, ale nekonštruuje celý konečný automat. Namiesto neho sa pred samotným vyhľadávaním vypočíta v čase $\theta(m)$ zo vzorky tzv. prefixová funkcia. Samotné vyhľadávanie vzorky sa realizuje v čase $\theta(n)$.



Pre vzorku $P[1..q]$ zhodnú s textom $T[s+1..s+q]$ testujeme, aké je najvắčšie $s' > s$ pre ktoré

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

kde $s' + k = s + q$. K výpočtu s' nepotrebujeme poznať text, pretože $T[s'+1..s'+k]$ je prefixom vzorky.

Pre danú vzorku $P[1..m]$ definujeme prefixovú funkciu $\pi : \{1, 2, \ldots, m\} \rightarrow \{0, 1, \ldots m-1\}$ predpisom

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

P_9 k r o k r o k r o k y $\pi[9] = 6$

P_6 k r o k r o k r o k y $\pi[6] = 3$

P_3 k r o k r o k r o k y $\pi[3] = 0$

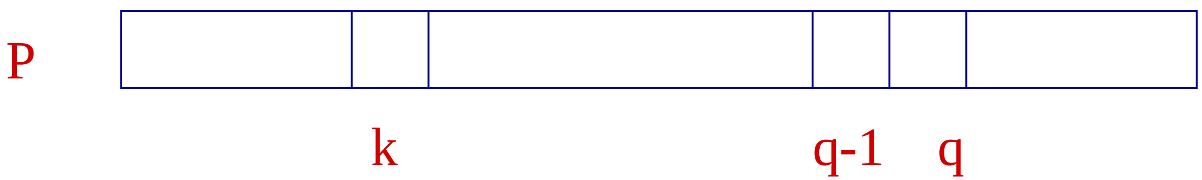
$\mathrm{KMP}(T, P)$

$\pi \leftarrow$ PREFIXOVÁ_FUNKCIA(P); $\quad q \leftarrow 0$
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

Počítame $\pi[q]$, poznáme $\pi[1], \ldots, \pi[q-1]$. Nech $\pi[q-1] = k$.



$$P[1..k] = P[\star..q-1]$$

ak $P[k+1] = [q]$ tak $P[1..k+1] = P[\star..q]$ a $\pi[q] = k+1$

ak $P[k+1] \neq [q]$ tak hľadáme iný prefix $\triangle$ taký, že $\triangle$ je sufixom P_q — kandidátom je práve $\pi[k]$ (z rovnosti $P[1..k] = P[\star..q-1]$ vyplýva, že aj sufixy týchto reazcov sa musia zhodovať).
Preto testujeme $P[\pi[k]+1] \stackrel{?}{=} P[q]$.
V prípade nerovnosti postupujeme analogicky.

Časová zložitosť algoritmu Knuth-Morris-Pratt

Pre PREFIXOVÁ_FUNKCIA použijeme metódu účtov.

Každý **for** cyklus z riadkov 2-6 dostane 3 kredity. 2 kreditmi zaplatíme príkazy z riadkov 5 a 6. Zvyšný 1 kredit uložíme na účet.

Pri každom vykonaní príkazu v riadku 4 sa zníží hodnota k, lebo $\pi[k] < k$. Súčasne je $\pi[k] \geq 0$ pre všetky k a preto počet opakovaní príkazu nie je v́äčší než počet ukončených opakovaní **for** cyklu. Všetky priradenia v riadku 4 preto môžeme zaplatiť kreditmi z účtu.

Zložitosť výpočtu PREFIXOVÁ_FUNKCIA je $\Theta(m)$.

Analogickým postupom sa ukáže, že zložitosť algoritmu KMP je $\Theta(n)$.