Obsah

1 Vyhľadávanie, triedenie a súvisiace problémy **2**
1.1 Hľadanie k-teho najmenšieho prvku . . . 2
1.2 Priemerný počet porovnaní triediacich algoritmov . . . 4
1.3 Algoritmy na dynamických množinách . . . 6
1.3.1 Realizácia slovníka hashovaním . . . 6
1.3.2 Realizácia slovníka pomocou 2-3 stromov . . . 7
1.3.3 UNION/FIND-SET problém . . . 10
1.3.4 Zrýchlenie algoritmu pre UNION/FIND-SET problém . . . 12
1.4 Ďalšia literatúra . . . 13

2 Grafové algoritmy **14**
2.1 Najlacnejšia kostra grafu . . . 14
2.2 Najlacnejšie cesty v grafe . . . 17
2.2.1 Dijkstrov algoritmus . . . 17
2.2.2 Floyd–Warshallow algoritmus . . . 18
2.3 Ďalšia literatúra . . . 20

3 Algoritmy na maticiach **21**
3.1 Strassenov algoritmus násobenia matíc . . . 21
3.1.1 Násobenie booleovských matíc . . . 22
3.2 LUP dekompozícia matíc . . . 23
3.3 Ďalšia literatúra . . . 24

4 Metódy tvorby efektívnych algoritmov **25**
4.1 Princíp neustáleho zlepšovania . . . 25
4.2 Voľba vhodnej štruktúry údajov . . . 26
4.3 Princíp vyváženosti . . . 26
4.4 Metóda "Rozdeľuj a panuj" . . . 27
4.5 Dynamické programovanie . . . 28
4.5.1 Problém násobenia reťazca matíc . . . 29
4.5.2 0-1 knapsack problém . . . 30
4.6 Greedy algoritmy . . . 31
4.7 Ďalšia literatúra . . . 32

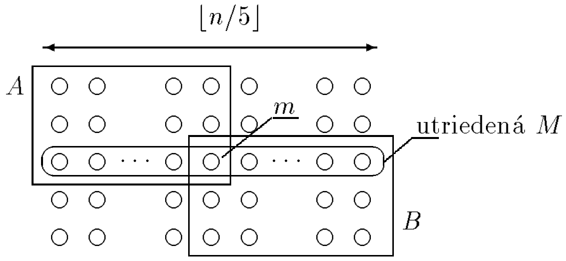
5 $\mathcal{NP}$-úplnosť **33**
5.1 Triedy $\mathcal{P}$ a $\mathcal{NP}$ . . . 33
5.2 $\mathcal{NP}$-úplné problémy . . . 35
5.2.1 Booleovské výrazy . . . 35
5.2.2 $\mathcal{NP}$-úplné problémy na neohodnotených grafoch . . . 37
5.2.3 Optimalizačné versus rozhodovacie problémy . . . 38
5.3 Ďalšia literatúra . . . 40

6 Aproximatívne algoritmy **41**

Dôkaz: Zoraďme (v neklesajúcom poradí "zľava doprava") jednotlivé (utriedené) päťprvkové postupnosti podľa veľkosti ich mediánov [1] — pozri obr. 1 (stĺpce predstavujú jednotlivé päťprvkové postupnosti utriedené neklesajúco "zhora dolu"). Obdĺžnik B obsahuje $3\lceil\lfloor n/5\rfloor/2\rceil$ prvkov a hodnota každého z nich je aspoň m. Preto

$$|S_1| \le n - |B| = n - 3\lceil\lfloor n/5\rfloor/2\rceil \le 3n/4$$

pre $n \ge 50$. Podobne tvrdenie platí pre S_3 (v dôkaze použi obdĺžnik A). □



obr. 1: Odhad veľkostí $|S_1|, |S_3|$

Veta 1.3 *Algoritmus SELECT nájde k-ty najmenší prvok v n-prvkovej postupnosti S v čase $O(n)$.*

Dôkaz: Nech $T(n)$ je čas potrebný na nájdenie k-teho najmenšieho prvku v n prvkovej postupnosti. Keďže $|S_1| \le 3n/4$ a $|S_3| \le 3n/4$ (pozri lemu 1.2), rekurzívne volanie v riadku (2) alebo (3) potrebuje čas nanajvýš $T(3n/4)$. Rekurzívne volanie v riadku (1) potrebuje čas nanajvýš $T(n/5)$. Všetky ostatné časti algoritmu potrebujú čas $O(n)$.

Teda existuje c ($c > 0$) také, že

$$T(n) \le T(n/5) + T(3n/4) + cn.$$

Indukciou ľahko dokážeme, že $T(n) \le 20cn$.

Dôkaz korektnosti algoritmu prenechávame na čitateľa. □

V ďalšom uvedieme tvrdenia, ktoré ukazujú, že nie je možné zostrojiť algoritmus s asymptoticky lepšou časovou zložitosťou ako $O(n)$, či už uvažujeme o priemernom alebo najhoršom prípade.

Definícia 1.4 *Hĺbka vrcholu w v strome T je dĺžka cesty z w do koreňa stromu T.*

Veta 1.5 *Ak T je rozhodovací strom, ktorý hľadá k-ty najmenší prvok v množine S ($|S| = n$), potom každý list stromu T má hĺbku aspoň $n - 1$.*

Dôsledok 1.6 *Nájdenie k-teho najmenšieho prvku v S potrebuje aspoň $n - 1$ porovnaní v priemernom aj najhoršom prípade.*

Cvičenia

Cvičenie 1.1 Uvažujme algoritmus 1. Je nutné, aby sme rozdeľovali pôvodnú postupnosť práve na päťprvkové postupnosti, alebo je možné toto číslo zmeniť (napríklad na trojprvkové alebo sedemprvkové)? Bude takáto zmena mať vplyv na časovú zložitosť algoritmu, ak áno, aký?

Cvičenie 1.2 Prečo pre menej ako 50 prvkov postupujeme v algoritme 1 odlišne? Je dôležité, že je to práve 50? Aký bude mať vplyv zmena tohoto čísla na časovú zložitosť algoritmu?

[1] Všimnite si, že algoritmus nezoraďuje päťprvkové postupnosti ako na obr. 1, namiesto toho hľadá prvok m ako medián postupnosti M

Dôkaz: Nech T je striktne binárny strom s m listami, ktorého suma hĺbok listov D_T je minimálna, tj. $D_T = d(m)$. Nech T_1 je ľavý podstrom a T_2 je pravý podstrom stromu T. Nech i je počet listov stromu T_1. Potom počet listov T_2 je $m - i$. Zrejme platí

$$D_T = i + D_{T_1} + (m - i) + D_{T_2}$$

(hĺbka každého z i listov stromu T_1 je v strome T o jedno väčšia ako v strome T_1 a to isté platí pre $m - i$ listov stromu T_2).

Keďže T má minimálnu hodnotu D_T, musí tiež platiť $D_{T_1} = d(i)$ a takisto $D_{T_2} = d(m-i)$ (ináč by bolo možné nahradiť ľavý podstrom T_1 stromu T podstromom T_1', ktorý by mal hodnotu $D_{T_1'}$ menšiu a tým by sme zmenšili hodnotu D_T; podobne pre T_2).

Teda

$$d(m) = m + d(i) + d(m - i).$$

Matematickou indukciou ľahko dokážeme, že $d(m) \geq m \log m$, lebo funkcia $f(x) = x \log x + (m - x) \log(m - x)$ nadobúda minimum pre $x = m/2$.

Keďže T_R je striktne binárny strom s m listami, musí platiť:

$$D_{T_R} \geq d(m) \geq m \log m.$$

□

Veta 1.9 *Každý algoritmus triediaci porovnávaním urobí v priemernom prípade $\Omega(n \log n)$ porovnaní za predpokladu, že všetky permutácie postupnosti n prvkov sa na vstupe vyskytujú s rovnakou pravdepodobnosťou.*

Dôkaz: Nech A je ľubovoľný algoritmus triediaci porovnávaním. Nech n je ľubovoľný rozsah vstupu a nech T_A^n je rozhodovací strom triediaci n prvkov zodpovedajúci algoritmu A. Strom T_A^n má práve $n!$ listov (pre každú permutáciu na vstupe práve jeden list) a suma hĺbok jeho listov je celkový počet porovnaní, ktoré vykoná algoritmus A na všetkých $n!$ vstupných permutáciách. Teda priemerný počet porovnaní algoritmu A na vstupoch rozsahu n je na základe lemy 1.8

$$\frac{D_{T_A^n}}{n!} \geq \log n! \geq \log \left(\frac{n}{2}\right)^{\frac{n}{2}} = \Omega(n \log n).$$

□

Cvičenia

Cvičenie 1.6 Ukážte, že na nájdenie minimálneho prvku v danej postupnosti je potrebných aspoň $\lceil \frac{n}{2} \rceil$ porovnaní. Ukážte, že zložitosť tohto problému[2] vzhľadom na operácie porovnania je $n - 1$.

Cvičenie 1.7 Uvažujme problém súčasného nájdenia minimálneho aj maximálneho prvku v danej postupnosti prvkov. Nájdite riešenie používajúce $\lceil 3n/2 \rceil - 2$ porovnaní. Dokážte, že menej porovnaní nestačí.[3]

Cvičenie 1.8 Ukážte, že druhý najmenší prvok z n prvkov sa dá nájsť pomocou $n + \lceil \log_2 n \rceil - 2$ porovnaní.

Cvičenie 1.9 Ukážte, že na vyhľadanie prvku v utriedenom poli je potrebných v priemernom prípade $\Omega(\log n)$ porovnaní.

[2] Zložitosťou $t_P(n)$ problému P, kde n je veľkosť vstupu, rozumieme $\min\{t(A, n) | A \in A_P\}$, kde $t(A, n)$ je počet príslušných operácií (v našom prípade porovnaní) algoritmu A pre najhorší prípad vstupu n a A_P je množina všetkých algoritmov riešiacich problém P.

[3] Môžete použiť metódu stavových priestorov: Označme **a** počet neporovnaných, **b** počet porovnaných vždy menších, **c** počet porovnaných vždy väčších a **d** počet ostatných prvkov a sledujme počet porovnaní potrebných na zmenu stavu $(n, 0, 0, 0) \to (0, 1, 1, n - 2)$.

Suma cien všetkých m^n ciest z koreňa do m^n listov je teda

$$\sum_{i=0}^{n-1} m^i(i+m)m^{n-i-1} = m^n \sum_{i=0}^{n-1}(1+\frac{i}{m}).$$

Z toho vyplýva, že priemerný čas potrebný na "spracovanie" n prvkovej postupnosti je

$$O(\sum_{i=0}^{n-1}(1+\frac{i}{m})) = O(n)$$

pre $n \leq m$, lebo cena cesty z koreňa do listu je úmerná času potrebnému na spracovanie príslušnej n prvkovej postupnosti. □

Poznámka: Nevýhodou hashovania je zložitosť v najhoršom prípade až $\Omega(n^2)$ — napríklad σ=(INSERT(a_1,S), INSERT(a_2,S), ..., INSERT(a_n,S), MEMBER(a_1,S), ..., MEMBER(a_n,S)), kde $h(a_1) = h(a_2) = \ldots = h(a_n)$.

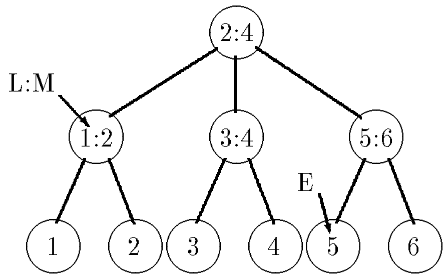
1.3.2 Realizácia slovníka pomocou 2-3 stromov

V kapitole 1.3.1 sme sa zoznámili s problémom slovníka a s jeho riešením pomocou hashovania. V tejto kapitole si ukážeme iné riešenie a to pomocou 2-3 stromov.

Definícia 1.12 *2-3 strom je strom, v ktorom každý vrchol, ktorý nie je list, má dvoch alebo troch synov a všetky cesty z koreňa do listov sú rovnako dlhé.*

Lineárne usporiadanú množinu S možno reprezentovať 2-3 stromom priradením prvkov z S listom 2-3 stromu (zľava doprava od najmenšieho prvku po najväčší).

Označenie: Nech $E[l]$ označuje prvok z S priradený listu l. Nech v je vrchol 2-3 stromu, ktorý nie je list. Nech $L[v]$ (resp. $M[v]$) označuje najväčší prvok z S priradený listom podstromu, ktorého koreňom je najľavejší (resp. druhý) syn vrcholu v.



obr. 2: 2-3 strom reprezentujúci množinu $S = \{4, 1, 3, 6, 2, 5\}$

Lema 1.13 *Nech T je 2-3 strom s výškou h. Potom počet vrcholov stromu T je medzi $2^{h+1}-1$ a $(3^{h+1}-1)/2$ a počet listov je medzi 2^h a 3^h.*

Dôkaz: Matematickou indukciou vzhľadom na výšku stromu h. □

Operácia MEMBER. Nech v je koreň 2-3 stromu reprezentujúceho množinu S. Uvedieme procedúru SEARCH(a, v), ktorá prehľadáva strom T od koreňa k listom, pričom využíva hodnoty L a M v jednotlivých vrcholoch. Algoritmus postupuje metódou podobnou binárnemu prehľadávaniu. V prípade, že $a \in S$, procedúra vráti vrchol w, ktorý je otcom listu s hodnotou a. Ak $a \notin S$, potom bude výsledkom procedúry vrchol w, pod ktorý by bol zaradený list s hodnotou a.

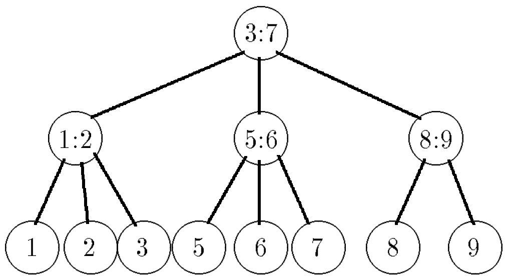
2. f má štyroch synov. Vytvoríme nový vrchol g, odpojíme od f jeho dvoch ľavých synov a pripojíme ich na g a g pripojíme na otca f. Ak má otec vrcholu f po tejto operácii troch synov, získali sme 2-3 strom reprezentujúci $S \cup \{a\}$, v opačnom prípade pokračujeme rekurzívnym spôsobom smerom ku koreňu stromu, až kým žiadny vrchol nemá štyroch synov.

Poznámka: V algoritme pre INSERT treba tiež priebežne upravovať hodnoty L a M.

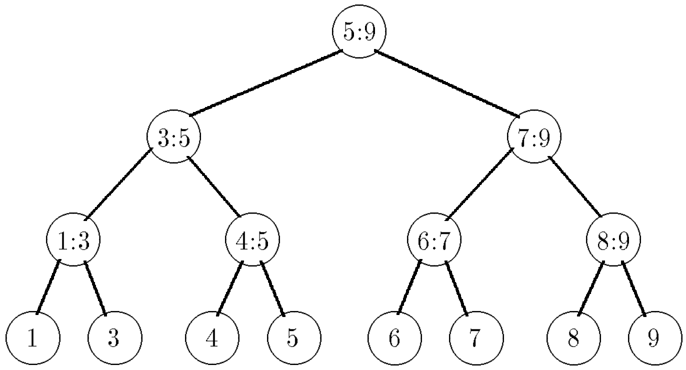
Lema 1.15 *Algoritmus pre INSERT vsunie nový prvok do 2-3 stromu s n listami v najhoršom prípade v čase $O(\log n)$. Naviac algoritmus zachováva usporiadanie hodnôt v listoch a výsledný strom je 2-3 strom.*

Dôkaz: Nech T je 2-3 strom s n listami. Z lemy 1.13 vyplýva, že výška stromu T je nanajvýš $\log n$. Preto nájdenie vrcholu f =SEARCH(a,v), kde v je koreň stromu T, potrebuje čas $O(\log n)$. Vsunutie nového listu do stromu potrebuje čas $O(1)$. Následná možná úprava na 2-3 strom, pri ktorej algoritmus postupuje pozdĺž cesty od vrcholu f do koreňa v, potrebuje čas najviac $O(\log n)$.

Druhá časť lemy vyplýva priamo z realizácie algoritmu INSERT. □



obr. 4: Výsledok operácie INSERT(2,v) pre strom T



obr. 5: Výsledok operácie INSERT(4,v) pre strom T

Operácia DELETE. Nech v je koreňom 2-3 stromu reprezentujúceho množinu S, nech $a \in S$. Nech l je list s hodnotou a. Nech ďalej f je výsledkom procedúry SEARCH(a,v)[4].

Môže nastať jedna z nasledujúcich možností:

1. f má troch synov. Potom možno odstrániť l a skončiť.

[4] Teda f je otcom listu l s hodnotou a

if *FIND-SET(u)≠FIND-SET(v)* **then**
UNION(FIND-SET(u),FIND-SET(v))

Definícia 1.18 *Výška stromu T je dĺžka najdlhšej cesty z koreňa stromu T do jeho listov. Výška vrcholu w stromu T je výška podstromu stromu T, ktorého koreňom je vrchol w.*

Dynamickú množinu S_i možno reprezentovať koreňovým stromom T_i, ktorého množina hrán bola vytvorená operáciami UNION (pozri ďalej).

Informácie o hranách a koreňoch stromov reprezentujúcich jednotlivé množiny možno kódovať v celočíselných poliach $p[1 \dots n]$, $h[1 \dots n]$ takto:

- Ak $p[j] = j$, potom vrchol j je koreňom niektorého stromu a $h[j]$ je výška tohto stromu.
- Ak $p[j] = l$ pre $l \neq j$, potom vrchol j je synom vrcholu l (v niektorom strome).

Pred vykonaním postupnosti σ (keď $S_i = \{i\}$ pre všetky i), platí $p[i] = i$ a $h[i] = 0$ pre všetky i.

Napíšeme procedúru UNION(l_i,l_j), ktorej vstupné hodnoty l_i, l_j sú korene stromov reprezentujúcich množiny S_i, S_j. Procedúra vytvorí strom reprezentujúci množinu $S_i \cup S_j$.

Algoritmus 4

procedure *UNION(l_i,l_j)*
begin
if $h[l_j] < h[l_i]$ **then** *pripoj koreň l_j ako syna na koreň l_i*
else *pripoj koreň l_i ako syna na koreň l_j*
end

Poznámka: V procedúre UNION sú vynechané detaily súvisiace s prípadnou úpravou hodnôt $p[l_i]$, $p[l_j]$, $h[l_i]$, $h[l_j]$.

Procedúra FIND-SET(j) pre daný vrchol j stromu T reprezentujúceho množinu S vráti koreň stromu T.

Algoritmus 5

procedure *FIND-SET(j)*
begin
if $p[j] = j$ **then return** j
else return *FIND-SET(p[j])*
end

Lema 1.19 *Na vytvorenie stromu s výškou h treba aspoň $2^h - 1$ operácií typu UNION.*

Dôkaz: Indukciou vzhľadom na h. Tvrdenie zrejme platí pre $h = 0$. Predpokladajme, že tvrdenie platí pre všetky stromy s výškou najviac h. Nech T je strom s výškou $h + 1$, ktorý bol vytvorený operáciou UNION zo stromu T_1 s výškou $i \leq h$ a zo stromu T_2 s výškou $j \leq h$, kde $i \leq j$.

Ak by $i < j \leq h$ alebo $i = j < h$, potom by T mal výšku najviac h (pozri algoritmus 4), čo je v spore s predpokladom. Teda $i = j = h$. Podľa indukčného predpokladu bolo treba aspoň $2^h - 1$ operácií UNION na vytvorenie T_1 a ďalších aspoň $2^h - 1$ operácií UNION na vytvorenie T_2, čo je spolu s operáciou UNION, ktorá vytvorí T, aspoň $2^{h+1} - 1$ operácií UNION. Preto tvrdenie platí aj pre všetky stromy s výškou $h + 1$. □

Veta 1.20 *Nech $S_i = \{i\}$ pre $i = 1, 2, \dots, m$. Časová zložitosť postupnosti σ skladajúcej sa z najviac $m - 1$ operácií UNION a z n operácií FIND-SET je v najhoršom prípade $O(m + n \log m)$.*

Cvičenie 1.15 Majme postupnosť $W = (W_1, \ldots, W_n)$ slov. Nájdite čo najefektívnejší algoritmus, ktorý nájde postupnosť čísel $V = (V_1, \ldots, V_n)$, pričom $V_i = 0$, ak sa medzi slovami $W_1, \ldots, W_{i-1}$ nenachádza prešmyčka slova W_i, alebo $V_i = k < i$, ak existuje slovo W_k $(k < i)$, že W_k je prešmyčkou slova W_i (ak ich je viac, nech k je najmenšie možné).

1.4 Ďalšia literatúra

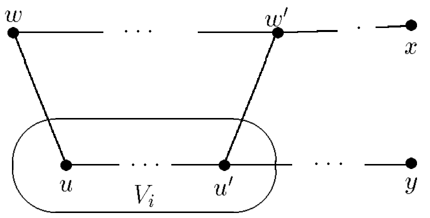
[AHU76] Aho A. V., Hopcroft J. E., Ullman J. D: *The Design and Analysis of Computer Algorithms*, Addison-Wesley 1976

[CLR90] Cormen T. H., Leiserson C. E., Rivest R. L.: *Introduction to Algorithms*, MIT Press and McGraw-Hill, 1990

[KN373] Knuth D. E.: *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley 1973

[WIE91] Wiederman J.: *Vyhledávání*, SNTL Praha 1991

[WIR87] Wirth N.: *Algoritmy a štruktúry údajov*, Alfa Bratislava 1987



obr. 8: Pôvodná a nová cesta z x do y

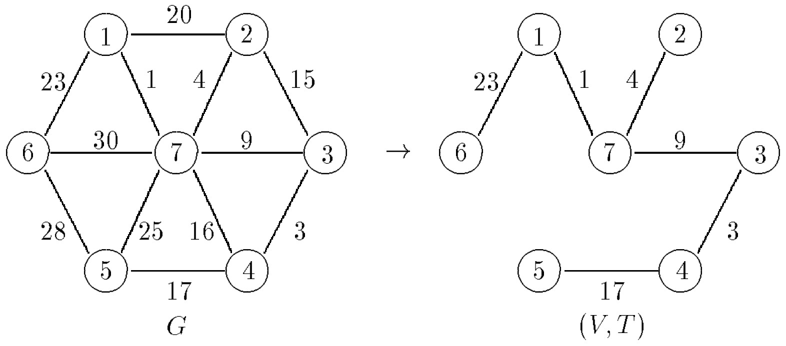
vedie v S' cez vrcholy $w', \ldots, w, u, \ldots, u'$, viď obr. 8). Teda S' je kostra grafu G, ktorej cena nie je väčšia než cena kostry S, keďže $h(u,w) \le h(u',w')$. □

Nasledujúci greedy algoritmus nájde najlacnejšiu kostru grafu. Vstupom je neorientovaný súvislý graf $G = (V, E)$ s ohodnotenými hranami. Vrcholy sú reprezentované prirodzenými číslami $1, 2, \ldots, |V|$, hrany spolu s cenami sú dané v zozname.

Algoritmus 7 (Kruskal)

begin
- $T \leftarrow \emptyset$ *(1)*
- *pre každý vrchol* $v \in V$ *vytvor množinu* $\{v\}$ *(2)*
- *utrieď hrany v* E *podľa cien v neklesajúcom poradí* *(3)*
- *pre každú hranu v* $(u, w) \in E$ *v poradí podľa neklesajúcih cien:* *(4)*
- **begin**
  - **if** *FIND-SET*$(u) \neq$ *FIND-SET*(w) **then begin** *(5)*
    - $T \leftarrow T \cup \{(u, w)\}$ *(6)*
    - *UNION(FIND-SET*(u)*, FIND-SET*(w)*)* *(7)*
  - **end** *(8)*
- **end**
- **return** T

end



obr. 9: Príklad Kruskalovho algoritmu

Veta 2.4 *Algoritmus 7 nájde najlacnejšiu kostru grafu* $G = (V, E)$ *s časovou zložitosťou v najhoršom prípade* $O(|E| \log |E|)$.

2.2 Najlacnejšie cesty v grafe

Definícia 2.5 *Nech $G = (V, E)$ je orientovaný graf s ohodnotenými hranami, tj. pre G je daná funkcia $h : E \to R$. Cesta v grafe G je postupnosť vrcholov $[v_0, v_1, \ldots, v_k]$, kde $(v_{i-1}, v_i) \in E$ pre $i = 1, 2, \ldots, k$. Cena cesty $P = [v_0, v_1, \ldots, v_k]$ je $\sum_{i=1}^{k} h(v_{i-1}, v_i)$. Cenu cesty P označme symbolom $|P|$.*

V súvislosti s cenou cesty nás budú zaujímať tri problémy:

1. Nájsť pre danú dvojicu vrcholov u, v cenu najlacnejšej cesty z u do v.
2. Nájsť pre daný vrchol $v_0 \in V$ cenu najlacnejšej cesty z v_0 do v pre všetky $v \in V$.
3. Nájsť cenu najlacnejšej cesty z u do v pre všetky $u, v \in V$.

2.2.1 Dijkstrov algoritmus

Ak sú ceny hrán grafu nezáporné reálne čísla, potom možno problém 2 riešiť Dijkstrovým algoritmom. Algoritmus dostane ako vstup orientovaný graf $G(V, E)$, vrchol v_0 a čiastočnú funkciu $h : V \times V \to R_0^+$. Predpokladáme, že $h(u, u) = 0$, ak $(u, v) \in E$ tak $h(u, v)$ je ohodnotenie hrany (u, v). Vrcholy grafu sú reprezentované celými číslami $1, 2, \ldots, |V|$ a predpokladáme, že funkciu h možno vypočítať v čase $O(1)$. Po skončení algoritmu bude pre každý vrchol $v \in V$ v $D[v]$ uložená cena najlacnejšej cesty z v_0 do v.

Poznámka: Pre jednoduchosť čiastočnú funkciu h zúplníme tak, že v prípade $(u, v) \notin E$ položíme $h(u, v) = \infty$. V tomto zmysle potom pre ľubovoľnú postupnosť vrcholov $[v_0, v_1, \ldots, v_k]$ vieme vypočítať cenu zodpovedajúcej cesty, pričom ak pre nejaké i $(0 \leq i < k)$ platí $(v_i, v_{i+1}) \notin E$, cena uvedenej cesty bude ∞.

Algoritmus 8 (Dijkstra)

begin
  $S \leftarrow \{v_0\}$ *(1)*
  $D[v_0] \leftarrow 0$ *(2)*
  pre každý vrchol $v \in V \setminus \{v_0\}$: $D[v] \leftarrow h(v_0, v)$ *(3)*
  while $S \neq V$ **do begin** *(4)*
    vyber $w \in V \setminus S$ *taký, že hodnota* $D[w]$ *je minimálna* *(5)*
    $S \leftarrow S \cup \{w\}$ *(6)*
    pre každý $v \in V \setminus S$: *(7)*
      $D[v] \leftarrow min\{D[v], D[w] + h(w, v)\}$ *(8)*
  end
end

Veta 2.6 *Algoritmus 8 vypočíta cenu najlacnejšej cesty z v_0 do každého vrcholu grafu G v najhoršom prípade v čase $O(|V|^2)$.*

Dôkaz:

Časová zložitosť: Riadok (5) a tiež aj cyklus v riadkoch (7) a (8) potrebujú čas $O(|V|)$. Riadok (6) potrebuje čas $O(1)$. Keďže riadky (5) až (8) sú vykonané $(|V| - 1)$ krát a riadky (1) až (3) potrebujú čas $O(|V|)$, na vykonanie algoritmu stačí čas $O(|V|^2)$.

Korektnosť: Korektnosť algoritmu ukážeme metódou invariantov. Stanovíme invariant, ktorý bude platný pred začatím každej iterácie cyklu while (riadky (4) až (8)) resp. po jeho skončení. Pre každé $v \in V$:

1. Ak $v \in S$, potom $D[v]$ je cena najlacnejšej cesty z v_0 do v, pričom existuje cesta z v_0 do v celá ležiaca v S s cenou $D[v]$.

Algoritmus 9 (Floyd–Warshall)

> **begin**
> $n \leftarrow |V|$
> $C^{(0)} \leftarrow W$
> **for** $k \leftarrow 1$ **to** n **do**
> **for** $i \leftarrow 1$ **to** n **do**
> **for** $j \leftarrow 1$ **to** n **do**
> $c_{ij}^{(k)} \leftarrow MIN(c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)})$
> **return** $C^{(n)}$
> **end**

Veta 2.7 *Algoritmus 9 vypočíta cenu najlacnejšej cesty z každého vrcholu i do každého vrcholu j v najhoršom prípade v čase $O(|V|^3)$.*

Dôkaz:

Časová zložitosť: Zrejmá.

Korektnosť algoritmu: Indukciou vzhľadom na k možno dokázať, že $c_{ij}^{(k)}$ je cena najlacnejšej cesty z vrcholu i do vrcholu j, ktorej všetky vnútorné vrcholy sú z množiny $\{1, 2, \dots, k\}$ (vnútorné vrcholy cesty sú všetky vrcholy cesty okrem jej prvého a posledného vrcholu). □

Poznámka: Podobne ako v prípade algoritmu 8 možno aj algoritmus 9 upraviť tak, aby okrem hodnôt $c_{ij}^{(k)}$ počítal aj hodnoty $p_{ij}^{(k)}$, kde $p_{ij}^{(k)}$ je predposledný vrchol najlacnejšej cesty z vrcholu i do vrcholu j, ktorej všetky vnútorné vrcholy sú z množiny $\{1, 2, \dots, k\}$. Pomocou hodnôt $p_{ij}^{(n)}$ možno zostrojiť najlacnejšiu cestu medzi ľubovoľnými dvoma vrcholmi grafu G. Pre $p_{ij}^{(k)}$ platí nasledujúci vzťah:

$$p_{ij}^{(k)} = \begin{cases} \text{nil} & \text{, ak } k = 0 \wedge (i = j \vee w_{ij} = \infty) \\ i & \text{, ak } k = 0 \wedge i \neq j \wedge w_{ij} < \infty \\ p_{ij}^{(k-1)} & \text{, ak } k > 0 \wedge c_{ij}^{(k-1)} \leq c_{ik}^{(k-1)} + c_{kj}^{(k-1)} \\ p_{kj}^{(k-1)} & \text{, ak } k > 0 \wedge c_{ij}^{(k-1)} > c_{ik}^{(k-1)} + c_{kj}^{(k-1)} \end{cases}$$

Vzťah možno dokázať matematickou indukciou.

Cvičenia

Cvičenie 2.4 Ukážte, že Dijkstrov algoritmus nefunguje pre záporné dĺžky hrán. Nájdite časť v dôkaze správnosti Dijkstrovho algoritmu, kde sa podmienka nezápornosti hrán využíva.

Cvičenie 2.5 Nájdite algoritmus časovej zložitost $O(|V|^3)$, ktorý nájde najkratšie cesty z vrcholu v_0 do všetkých ostatných vrcholov, ak neuvažujeme podmienku nezápornosti hrán, ale iba podmienku, že v grafe G sa nenachádza cyklus zápornej dĺžky.

Cvičenie 2.6 Ukážte, že Floyd-Warshallow algoritmus nefunguje, ak sa v grafe nachádza cyklus zápornej dĺžky. Nájdite, kde sa táto podmienka využije v dôkaze správnosti Floyd-Warshallowho algoritmu.

Cvičenie 2.7 *Hamiltonovská kružnica* je kružnica v grafe G, ktorá prechádza cez všetky vrcholy grafu G. Ukážte, že problém, či v grafe G existuje Hamiltonovská kružnica možno riešiť v polynomiálnom čase, ak možno v polynomiálnom čase riešiť problém nájdenia najkratších ciest z každého vrcholu do každého, ak neuvažujeme žiadne obmedzenia na ohodnotenie hrán.

3 Algoritmy na maticiach

V tejto kapitole sa budeme zaoberať výpočtovou zložitosťou násobenia matíc. Uvidíme, že časová zložitosť $O(n^3)$ priamočiareho algoritmu na násobenie matíc nie je optimálna. Ukážeme si algoritmus s časovou zložitosťou $O(n^{2.81})$. Najlepší algoritmus známy do roku 1990 má časovú zložitosť $O(n^{2.376})$. Množstvo iných problémov je možné zredukovať na násobenie matíc. V týchto prípadoch použitím lepšieho násobenia matíc dostaneme efektívne riešenia, ktoré majú nižšiu časovú zložitosť, ako by sa na prvý pohľad dalo predpokladať (napríklad riešenie sústav lineárnych rovníc).

3.1 Strassenov algoritmus násobenia matíc

Zlepšenie priamočiareho algoritmu násobenia matíc spočíva v použití techniky "rozdeľuj a panuj". Priamočiare použitie tejto techniky však neprinesie očakávaný výsledok.

Pokúsme sa vynásobiť matice rozmeru $2n \times 2n$ pomocou operácií násobenia a sčítania matíc rozmeru $n \times n$ (každú maticu rozdelíme na štyri podmatice rozmeru $n \times n$):

$$AB = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

Takto sme pôvodný problém násobenia dvoch matíc rozmerov $2n \times 2n$ previedli na 8 násobení a 4 sčítania matíc rozmerov $n \times n$. Nech $T(n)$ je počet operácií potrebných na násobenie násobenie dvoch matíc rozmerov $n \times n$. Pri použití metódy rozdeľuj a panuj dostávame rekurentný vzťah

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2).$$

Riešením tohto rekurentného vzťahu dostávame $T(n) = \Theta(n^3)$. Použitím tejto metódy sme teda nedosiahli žiadne zlepšenie.

Kľúčom k riešeniu je nájdenie takého spôsobu násobenia matíc rozmerov 2×2, pri ktorom sa použije menší počet násobení.

Lema 3.1 *Súčin dvoch matíc typu 2×2 možno vypočítať pomocou 7 násobení a 18 sčítaní/odčítaní.*

Dôkaz: Pre súčin matíc

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

platí

$$\begin{aligned} c_{11} &= m_1 + m_2 - m_4 + m_6 \\ c_{12} &= m_4 + m_5 \\ c_{21} &= m_6 + m_7 \\ c_{22} &= m_2 - m_3 + m_5 - m_7, \end{aligned}$$

kde

$$\begin{aligned} m_1 &= (a_{12} - a_{22})(b_{21} + b_{22}) \\ m_2 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ m_3 &= (a_{11} - a_{21})(b_{11} + b_{12}) \\ m_4 &= (a_{11} + a_{12})b_{22} \\ m_5 &= a_{11}(b_{12} - b_{22}) \\ m_6 &= a_{22}(b_{21} - b_{11}) \\ m_7 &= (a_{21} + a_{22})b_{11} \end{aligned}$$

□

Cvičenia

Cvičenie 3.1 Na vynásobenie dvoch komplexných čísel tvaru $a+bi$ stačia štyri násobenia reálnych čísel $((a+bi)(c+di) = (ac-bd)+(ad+bc)i)$. Nájdite spôsob, ako vynásobiť dve komplexné čísla pomocou troch násobení. Ukážte, že je to najmenší možný počet násobení.

Cvičenie 3.2 Predpokladajme, že by sme matice nedelili na štyri časti (2×2), ale na viac častí. Koľko najviac operácií násobenia by sme mohli použiť pri delení na deväť častí (3×3) resp. na 16 častí (4×4), aby sme dostali algoritmus asymptoticky lepší ako Strassenov algoritmus?

Cvičenie 3.3 Priamočiary algoritmus násobenia dvoch n-ciferných čísel vyžaduje $O(n^2)$ operácií. Pokúste sa nájsť asymptoticky lepší algoritmus.

Cvičenie 3.4 Orientovaný graf $G=(V,E)$ máme daný pomocou matice susednosti A (t.j. $a[u,v]=1$, ak $(u,v)\in E$ a $a[u,v]=0$, ak $(u,v)\notin E$). Úlohou je nájsť maticu B, pre ktorú bude platiť $b[u,v]=1$, ak existuje orientovaná cesta z u do v a $b[u,v]=0$ inak. Nájdite algoritmus s časovou zložitosťou $O(n^{2.81}\log n)$.

Cvičenie 3.5 Riešte predchádzajúcu úlohu pre neorientovaný graf. Viete nájsť algoritmus aj s lepšou časovou zložitosťou ako $O(n^{2.81}\log n)$?

3.2 LUP dekompozícia matíc

LUP dekompozícia je jednou z metód, ako redukovať niektoré problémy na násobenie matíc. Samotný algoritmus nájdenia LUP dekompozície uvádzať nebudeme, pretože je pomerne komplikovaný. Dôležitým faktom však pre nás je, že tento algoritmus má rovnakú časovú zložitosť ako násobenie matíc v ňom použité (časovo najnáročnejšia operácia). Ukážeme teda aspoň niekoľko problémov, ktoré možno efektívne riešiť redukciou na problém LUP dekompozície.

Definícia 3.4 *Nech $A=(a_{ij})$ je matica typu $n\times n$ nad R. A je horná trojuholníková matica, ak $a_{ij}=0$ pre $1\le j<i\le n$.*

A je jednotková dolná trojuholníková matica, ak $a_{ij}=0$ pre $1\le i<j\le n$ a $a_{ii}=1$ pre $1\le i\le n$.

A je permutačná matica, ak $a_{ij}\in\{0,1\}$ pre $1\le i,j\le n$ a v každom riadku a v každom stĺpci má A práve jednu jednotku.

Definícia 3.5 *LUP dekompozícia matice A je trojica matíc L, U, P typu $n\times n$, kde $LUP=A$, L je jednotková dolná trojuholníková matica, U je horná trojuholníková matica a P je permutačná matica.*

Lema 3.6 *Pre každú regulárnu štvorcovú maticu existuje LUP dekompozícia.*

Veta 3.7 *Predpokladajme, že pre každé n vieme násobiť dve matice typu $n\times n$ v čase $M(n)$, kde pre každé m a nejaké $\varepsilon>0$ platí $M(2m)\ge 2^{2+\varepsilon}M(m)$.*

Potom pre každú maticu A typu $n\times n$ možno v čase $O(M(n))$:

1. *Zistiť, či A je regulárna a ak je, potom nájsť jej LUP dekompozíciu.*
2. *Ak A je regulárna, nájsť riešenie systému lineárnych algebraických rovníc $Ax=b$.*
3. *Ak A je regulárna, nájsť inverznú maticu A^{-1}.*
4. *Vypočítať determinant matice A.*

Dôkaz:

1. Tvrdenie uvádzame bez dôkazu. Príslušný algoritmus môže čitateľ nájsť v [AHU76], [CLR90].

4 Metódy tvorby efektívnych algoritmov

V tejto kapitole sa budeme zaoberať niektorými technikami, ktoré sa používajú pri tvorbe efektívnych algoritmov. Vo všeobecnosti neexistuje univerzálna metóda konštrukcie efektívnych algoritmov. Napriek tomu však často možno použiť niektorú z nasledujúcich metód:

Princíp neustáleho zlepšovania. O každom algoritme, o ktorom sa zatiaľ nepodarilo dokázať že sa nedá zlepšiť, treba predpokladať, že sa zlepšiť dá.

Voľba vhodnej štruktúry údajov. Spôsob organizácie dát pri výpočte algoritmu často výrazne vplýva na jeho efektívnosť. Preto voľbou vhodnej štruktúry údajov môžeme získať efektívnejší algoritmus.

Princíp vyváženosti (Balancing). Navrhované algoritmy často používajú rekurzívne schémy výpočtu alebo rekurzívne dátové štruktúry. V týchto prípadoch sa ukazuje, že je výhodné z hľadiska efektívnosti, aby medzi jednotlivými podštruktúrami (prípadne podvýpočtami) bola istá vyváženosť.

Metóda "Rozdeľuj a panuj" (Divide and conquer). Rozdelíme úlohu na niekoľko menších podúloh, ktoré riešime samostatne. Potom z ich výsledkov vypočítame celkový výsledok.

Dynamické programovanie. Táto metóda je podobná ako "rozdeľuj a panuj" — riešenie problému takisto dostávame z riešení podproblémov. V dynamickom programovaní postupujeme zdola nahor. Vyriešime podproblémy a uložíme si výsledky, aby mohli byť použité pri výpočte väčších problémov.

Greedy algoritmy. Pri hľadaní optimálneho riešenia daného problému si pri výpočte zvolíme vždy lokálne najlepšiu možnosť. Tento prístup vedie k rýchlemu algoritmu, musíme však dávať pozor na to, aby riešenie bolo korektné.

4.1 Princíp neustáleho zlepšovania

Pri skúmaní určitého problému je zvykom postupovať z dvoch strán: na jednej strane sa snažíme nájsť stále efektívnejší algoritmus a takto nájdený algoritmus nám poskytuje horný odhad zložitosti problému. Na druhej strane sa snažíme nájsť čo najlepší dolný odhad.

Z tohto hľadiska o každom algoritme, o ktorom sa zatiaľ nepodarilo dokázať, že ho nemožno zlepšiť, treba predpokladať, že sa zlepšiť dá.

Príklad: Algoritmus QUICKSORT má v priemernom prípade zložitosť $O(n \log n)$. Ukázali sme, že zložitosť triedenia v priemernom prípade je $\Omega(n \log n)$. V tomto prípade sme teda dosiahli zhodu dolného a horného odhadu.

Príklad: Násobenie matíc: klasická metóda $O(n^3)$, Strassenova metóda $O(n^{2.81})$, najlepší výsledok do roku 1990 $O(n^{2.376})$. Známy dolný odhad $\Omega(n^2)$.

Príklad: Násobenie n-bitových čísel: klasická metóda $O(n^2)$, metóda "Rozdeľuj a panuj" $O(n^{1.59})$, Shönhage–Strassenova metóda pomocou Fourierovej transformácie $O(n \log n \log\log n)$

Cvičenia

V nasledujúcich cvičeniach nájdite viacero algoritmov s rôznymi časovými zložitosťami, pokúste sa čo najviac priblížiť k uvedenému dolnému odhadu. Pokúste sa tiež ukázať uvedený dolný odhad.

Cvičenie 4.1 Dané je n-prvkové pole A. Zostavte algoritmus, ktorý pre dané celé číslo k $(-n < k < n)$ cyklicky posunie prvky poľa A o k miest doprava. Dolný odhad je $\Omega(n)$.

Cvičenie 4.2 Daná je postupnosť n celých (kladných i záporných) čísel. Nájdite v nej podpostupnosť po sebe nasledujúcich členov postupnosti s najväčším súčtom. Dolný odhad je $\Omega(n)$.

Cvičenie 4.3 Dané sú súradnice n bodov v rovine. Nájdite konvexný obal týchto bodov (konvexný obal je najmenší konvexný mnohouholník, ktorý tieto body obsahuje). Dolný odhad je $\Omega(n \log n)$.

Ak však zabezpečíme, aby podstromy pod ľubovoľným prvkom mali približne rovnakú výšku (tzv. vyvážené stromy), získame vždy strom s výškou približne $\log n$ a teda vyhľadávanie v takomto strome má časovú zložitosť $O(\log n)$ aj v najhoršom prípade.

Príklad: 2-3 stromy (viď. strana 7), AVL stromy, červeno-čierne (RB) stromy, stromy pre UNION/FIND-SET problém (viď. strana 11), binárne prehľadávacie stromy, algoritmus SELECT pre hľadanie k-teho najmenšieho prvku (viď. strana 2).

4.4 Metóda "Rozdeľuj a panuj"

Metóda je založená na tom, že problém rozdelíme na niekoľko podproblémov, podobných ako pôvodný problém, ale menšieho rozsahu. Potom rekurzívne vyriešime tieto podproblémy a nakoniec zostrojíme riešenie celého problému pomocou riešení podproblémov.

Ak sa podarí rozdeliť problém rozsahu n na d problémov rozsahu n/c, pričom celkový čas potrebný na rozklad na podproblémy a konštrukciu riešenia problému pomocou riešení podproblémov nepresiahne čas bn, potom možno časovú zložitosť odhadnúť pomocou nasledujúcej vety.

Veta 4.1 *Nech b, c, d sú nezáporné konštanty. Riešenie rekurentnej rovnice*

$$T(n) = \begin{cases} bn & \text{pre } n = 1, \\ dT(n/c) + bn & \text{pre } n > 1, \end{cases}$$

je pre $n = c^k$

$$T(n) = \begin{cases} O(n) & \text{ak } d < c, \\ O(n \log n) & \text{ak } d = c, \\ O(n^{\log_c d}) & \text{inak.} \end{cases}$$

Dôkaz: Ak $n = c^k$, potom platí

$$\begin{aligned} T(n) &\leq bn + dT(n/c) \\ &\leq bn + bn\frac{d}{c} + bn\frac{d^2}{c^2} + \cdots + bn\left(\frac{d}{c}\right)^{\log_c n} \\ &= bn \sum_{i=0}^{\log_c n} r^i, \end{aligned}$$

kde $r = d/c$. Ak $r < 1$, potom rad $\sum_{i=0}^{\log_c n} r^i$ konverguje, čiže

$$T(n) \leq bn \sum_{i=0}^{\log_c n} r^i \leq \sum_{i=0}^{\infty} r^i \leq hn,$$

pre nejakú vhodnú kladnú konštantu h. Ak $r = 1$, potom

$$T(n) \leq bn(\log_c n + 1)$$

Ak $r > 1$, potom

$$T(n) \leq bn \sum_{i=0}^{\log_c n} r^i = bn \frac{r^{1+\log_c n} - 1}{r - 1} = bc^{\log_c n} \frac{\left(\frac{d}{c}\right)^{1+\log_c n} - 1}{\frac{d}{c} - 1} = O(d^{\log_c n}) = O(n^{\log_c d})$$

. □

Príklad: Sú dané dve n-bitové čísla x a y. Pokúsme sa nájsť efektívny algoritmus, ktorý tieto dve čísla vynásobí. Nech $x = a2^{n/2} + b$ a $y = c2^{n/2} + d$, kde a, b, c, d sú $n/2$ bitové čísla. Potom súčin $z = xy$ možno vypočítať takto:

4.5.1 Problém násobenia reťazca matíc

Príkladom problému, ktorý sa dá efektívne riešiť použitím metódy dynamického programovania je problém násobenia reťazca matíc.

Daných je n matíc $M_1, \ldots, M_n$, kde M_i je matica typu $r_{i-1} \times r_i$. Úlohou je vypočítať súčin týchto matíc s minimálnym celkovým počtom skalárnych násobení, pričom predpokladáme, že súčin matice typu $p \times q$ s maticou $q \times r$ potrebuje pqr skalárnych násobení (tj. matice násobíme klasickým spôsobom)[11].

Počet rôznych spôsobov, ako vypočítať súčin n matíc, tj. počet rôznych uzátvorkovaní, je $P(n)$, kde

$$P(n) = \begin{cases} 1 & \text{ak } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{inak} \end{cases}$$

Možno ukázať, že

$$P(n) = \frac{1}{n}\binom{2n-2}{n-1} = \Omega(4^n/n^{3/2})$$

Preskúmaním všetkých možností výpočtu súčinu n matíc vedie k neefektívnemu algoritmu s exponenciálnou zložitosťou. Pomocou dynamického programovania zostrojíme algoritmus so zložitosťou $O(n^3)$.

Nech sú dané čísla $r_0, \ldots, r_n$, kde $r_{i-1} \times r_i$ je typ matice M_i. Zavedieme pole $m[1 \ldots n, 1 \ldots n]$, kde $m[i,j]$ bude minimálny počet skalárnych násobení potrebných na výpočet súčinu matíc $M_i, \ldots, M_j$. V pomocnom poli $s[1 \ldots n, 1 \ldots n]$ budeme ukladať číslo $s[i,j] = k$, ktoré určuje, ako treba pri optimálnom násobení matíc reťazec uzátvorkovať (tj. určuje takéto uzátvorkovanie: $(M_i \times \cdots \times M_k) \times (M_{k+1} \times \cdots \times M_j)$, pričom reťazec $M_i, \ldots, M_k$ násobíme podľa hodnôt $m[i,k]$ a $s[i,k]$ a reťazec $M_{k+1}, \ldots, M_j$ podľa hodnôt $m[k+1,j]$ a $s[k+1,j]$).

Algoritmus 11

```
begin
    for i ← 1 to n do m[i,i] ← 0
    for l ← 1 to n − 1 do
        for i ← 1 to n − l do begin
            j ← i + l
            Nech k ∈ {i, i+1, ..., j−1} je taký index, pre ktorý je hodnota
                výrazu m[i,k] + m[k+1,j] + r_{i−1} r_k r_j minimálna
            m[i,j] ← m[i,k] + m[k+1,j] + r_{i−1} r_k r_j
            s[i,j] ← k
        end
end
```

Hodnoty tabuľky $m[1 \ldots n, 1 \ldots n]$ je možné vypočítať aj použitím metódy rozdeľuj a panuj. Procedúra RP(i, j) vypočíta hodnoty tabuľky $m[k,l]$ pre všetky $i \le k \le l \le j$, tj. pre všetky podreťazce reťazca matíc $M_i M_{i+1} \ldots M_j$. Celú tabuľku teda vypočítame príkazom RP(1,n).

Algoritmus 12

```
procedure RP(i,j)
begin
    if i = j then m[i,j] ← 0
    else m[i,j] ← min_{i≤k<j} {RP(i,k) + RP(k+1,j) + r_{i−1} r_k r_j}
    return m[i,j]
end
```

[11] Napríklad: nech M_1 je typu 10×30, M_2 je typu 30×5, M_3 je typu 5×100 a M_4 typu 100×10. Pri uzátvorkovaní $(M_1 \times (M_2 \times M_3)) \times M_4$ je počet násobení 55000 a pri uzátvorkovaní $(M_1 \times M_2) \times (M_3 \times M_4)$ vyžaduje riešenie úlohy iba 7000 operácií.

Nech $V(w, j) = \max_{S \subseteq \{1,2,\ldots,j\}} \{\sum_{i \in S} v_i | \sum_{i \in S} w_i \leq w\}$, pre $j = 1, 2, \ldots, n$ a $w = 0, 1, \ldots, W$. Ak optimálny výber objektov pre $V(w, j+1)$ obsahoval objekt s indexom $j+1$, po odobratí tohto objektu dostaneme výber s celkovou váhou o w_{j+1} menšou a s celkovou cenou o v_{j+1} menšou, pričom tento výber je zrejme optimálny pre $V(w - w_{j+1}, j)$. Ak optimálny výber pre $V(w, j+1)$ neobsahuje objekt s indexom $j+1$, potom $V(w, j+1) = V(w, j)$. Teda pre každé j a w platí

$$V(w, j+1) = \max\{V(w, j), v_{j+1} + V(w - w_{j+1}, j)\}.$$

Dynamickým programovaním možno riešiť 0-1 knapsack problém v čase $O(nW)$ a to postupným vypĺňaním tabuľky $V[0 \ldots W, 0 \ldots n]$ použijúc vzťah

$$V[w, j] = \begin{cases} \max\{V[w, j-1], v_j + V[w - w_j, j-1]\} & \text{ak } 0 < j \leq n \\ 0 & \text{ak } j = 0 \end{cases}$$

Výslednou hodnotou je číslo $V[W, n]$.

Cvičenia

Cvičenie 4.12 Je daná postupnosť čísel $a_1, \ldots, a_n$. Nájdite najdlhšiu rastúcu vybranú podpostupnosť tejto postupnosti (tj. čísla $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ také, že $a_{i_1} < a_{i_2} < \ldots < ai_k$ a k je najväčšie možné) (v čase $O(n^2)$ dynamickým programovaním, možno vylepšiť na $O(n \log n)$).

Cvičenie 4.13 Letecká spoločnosť prevádzkuje k liniek medzi mestami $1, \ldots, n$, pričom každá linka spája dve z týchto miest. Vyhrali ste letecký zájazd, pri ktorom vám letecká spoločnosť zaplatí vami vybraný okružný let postupne cez mestá $1, a_1, \ldots, a_{l-1}, n, a_{l+1}, \ldots, a_m, 1$, pričom $1 < a_1 < \ldots < a_{l-2} < n$ a $n > a_{l+1} > \ldots > a_m > 1$ a žiadne mesto (okrem 1) nenavštívite viackrát a prepravujete sa len linkami tejto leteckej spoločnosti. Nájdite takúto okružnú trasu, ktorá prechádza cez najväčší možný počet miest (časová zložitosť $O(n^2)$).

Cvičenie 4.14 Je daná postupnosť celých kladných čísel $V_1, \ldots, V_n$. Chceme z nej vybrať podpostupnosť s maximálnym súčtom takú, že z ľubovoľných troch za sebou idúcich členov pôvodnej postupnosti aspoň jeden vyberieme a aspoň jeden nevyberieme (časová zložitosť $O(n)$).

Cvičenie 4.15 Uvažujme nasledujúci spôsob kompresie: ak máme v postupnosti znakov n-krát sa za sebou opakujúcu podpostupnosť P, môžeme ju nahradiť postupnosťou $n[P]$. Napríklad `2[11[a]bc]` je kódom postupnosti: `aaaaaaaaaaabcaaaaaaaaaaabc`.

Je zadaná postupnosť znakov `a`...`z`. Nájdite algoritmus, ktorý nájde jeho najkratší možný komprimovaný zápis vyššieuvedenou metódou, pričom dĺžka zápisu je počet znakov vrátane zátvoriek a číslic.

4.6 Greedy algoritmy

Greedy algoritmy sa používajú na riešenie optimalizačných problémov. Globálne optimálne riešenia hľadajú (alebo vytvárajú) pomocou postupnosti lokálne optimálnych rozhodnutí, ktoré už ďalej nie sú revidované. Obvykle sú to iteratívne algoritmy, v ktorých lokálne optimálne rozhodnutia redukujú problémy na podproblémy menšieho rozsahu, v dôsledku čoho sú mnohé z týchto algoritmov rýchle.

Príklad: Dijkstrov algoritmus (pozri stranu 17), Kruskalov algoritmus (pozri stranu 15)

Príklad: Daných je n objektov s váhami $w_1, w_2, \ldots, w_n$ (kde w_i sú prirodzené čísla), cenami $v_1, v_2, \ldots, v_n$ a ďalej je dané prirodzené číslo W. Úlohou je vybrať časti niektorých objektov tak, aby celková cena vybraných častí bola najväčšia a zároveň aby ich celková váha neprekročila hodnotu W, tj. nájsť číslo

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \in <0,1>} \{\sum_{i=1}^{n} \alpha_i v_i | \sum_{i=1}^{n} \alpha_i w_i \leq W\}.$$

Tento problém sa nazýva racionálny knapsack problém a je ho možné riešiť pomocou greedy algoritmu v čase $O(n \log n)$ (a v čase $O(n)$ metódou rozdeľuj a panuj).

5 $\mathcal{NP}$-úplnosť

Doteraz sme sa zaoberali algoritmami, ktoré mali polynomiálnu časovú zložitosť, tj. na vstupoch rozsahu n potrebujú čas $O(n^k)$ pre nejaké k. Nie všetky problémy sa však dajú riešiť v polynomiálnom čase.

Príklad: Problém zastavenia Turingovho stroja nemožno algoritmicky riešiť. Existujú aj problémy, pre ktoré existuje algoritmické riešenie, ale neexistuje algoritmické riešenie v polynomiálnom čase.

Prakticky riešiteľné algoritmy sú také, ktoré možno riešiť algoritmami v polynomiálnom čase. Exponenciálne algoritmy sú totiž pre väčšie rozsahy vstupov z časového hľadiska nerealizovateľné (na rozdiel od polynomiálnych algoritmov s nízkym stupňom polynómu).

Poznámka: Hoci exponenciálna funkcia 2^n rastie rýchlejšie než ľubovoľná polynomiálna funkcia s premennou n, pre malé hodnoty n môže byť algoritmus s časovou zložitosťou $T(2^n)$ rýchlejší než mnohé polynomiálne algoritmy (napr. $2^n < n^{10}$ pre $n < 59$).

5.1 Triedy $\mathcal{P}$ a $\mathcal{NP}$

Pre niektoré problémy nie je známe, či je ich možné algoritmicky riešiť v polynomiálnom čase, alebo nie. Existujú niektoré triedy problémov, ktoré majú z tohto hľadiska špeciálny význam.

Triedu problémov, ktoré sa dajú riešiť deterministickým algoritmom v polynomiálnom čase, nazveme triedou $\mathcal{P}$ problémov. Triedu problémov, ktoré sa dajú riešiť nedeterministickým algoritmom v polynomiálnom čase nazveme triedou $\mathcal{NP}$ problémov.

Zjavne platí, že $\mathcal{P} \subseteq \mathcal{NP}$. V súčasnej dobe však nevieme povedať, či platí aj $\mathcal{P} = \mathcal{NP}$, alebo nie. Teda existujú úlohy v $\mathcal{NP}$, u ktorých nevieme nájsť polynomiálny deterministický algoritmus, ale súčasne ani nevieme dokázať, že taký algoritmus neexistuje. O niektorých problémoch však vieme povedať o niečo viac.

O probléme A z triedy $\mathcal{NP}$ hovoríme, že je $\mathcal{NP}$-úplný, ak pre ľubovoľný problém B z triedy $\mathcal{NP}$ platí, že je polynomiálne redukovateľný na tento problém. To znamená, že problém B možno riešiť tak, že nejakým polynomiálnym algoritmom pretransformujeme vstup pre problém B na vstup pre problém A, nájdeme riešenie problému A a potom toto riešenie opäť polynomiálnym algoritmom prevedieme na riešenie problému B.

Všimnime si, že problémy z triedy $\mathcal{NP}$-úplných majú jednu zaujímavú vlastnosť. Ak by sme totiž dokázali riešiť ľubovoľný problém z tejto triedy polynomiálnym deterministickým algoritmom, dokázali by sme aj všetky ostatné problémy z $\mathcal{NP}$ riešiť v polynomiálnom čase[12].

Niektoré $\mathcal{NP}$-úplné problémy sú veľmi podobné problémom, ktoré vieme riešiť deterministicky v polynomiálnom čase.

Príklad:

Det. polynomiálny čas	**$\mathcal{NP}$-úplný problém**
2-splniteľnosť	3-splniteľnosť
Hranové pokrytie grafu	Vrcholové pokrytie grafu
Najkratšie cesty v grafe	Najdlhšie cesty v grafe
Lineárne diofant. rovnice	Kvadratické diofant. rovnice
Racionálny knapsack problém	0-1 knapsack problém

Teraz zavedieme pojmy tried $\mathcal{P}$, $\mathcal{NP}$ a $\mathcal{NP}$-úplných problémov trochu formálnejšie — na Turingových strojoch. Turingov stroj dokáže vykonať v polynomiálnom čase každý výpočet, ktorý sa dá vykonať v polynomiálnom čase na bežnom počítači a má tú výhodu, že sa dá jednoducho a presne matematicky popísať.

Poznámka: Pre jednoduchosť budeme ďalej uvažovať iba polynómy s nezápornými koeficientami.

[12] Ak o niektorom probléme vieme, že patrí do triedy $\mathcal{NP}$-úplných problémov, stojí za zamyslenie to, či sa zaoberať hľadaním polynomiálneho riešenia takéhoto problému. Od roku 1971 sa to totiž ešte nikomu nepodarilo, navyše sa predpokladá, že taký algoritmus neexistuje. V praxi nemá väčšinou zmysel sa zaoberať písaním exponenciálneho algoritmu, preto schodná cesta v tomto prípade vedie cez nájdenie nejakého deterministického polynomiálneho aproximačného algoritmu.

5.2 $\mathcal{NP}$-úplné problémy

5.2.1 Booleovské výrazy

Booleovské výrazy obsahujúce booleovské premenné, logické spojky $\neg, \vee, \wedge, \Leftarrow, \Leftrightarrow$ a zátvorky (,) budeme kódovať tak, že jednotlivé premenné očíslujeme číslami $1, 2, \ldots$ a tieto premenné v booleovskom výraze nahradíme ich číslami v binárnom tvare.

Príklad: Booleovský výraz $(p \vee q) \Rightarrow (\neg p \vee q \wedge r)$ má kód $(1 \vee 10) \Rightarrow (\neg 1 \vee 10 \wedge 11)$.

Booleovský výraz je splniteľný, ak existuje aspoň jedno priradenie hodnôt 0,1 premenným také, že hodnota výrazu je 1.

Príklad: $(p \vee q) \wedge \neg r$ je splniteľný ($p = 1$, $q = 0$, $r = 0$). $(p \vee q) \wedge \neg p \wedge \neg q$ je nesplniteľný.

Definícia 5.5 *Nech SAT je množina všetkých splniteľných booleovských výrazov (SAT je jazyk nad abecedou* $\{\neg, \vee, \wedge, \Leftarrow, \Leftrightarrow, (,), 0, 1\}$*).*

Veta 5.6 (Cook, Levin) *SAT je $\mathcal{NP}$-úplný jazyk.*

Dôkaz: Čitateľ ľahko ukáže, že SAT patrí do množiny $\mathcal{NP}$ jazykov. Zostáva teda už len ukázať, že každý jazyk z množiny $\mathcal{NP}$ je polynomiálne transformovateľný na jazyk SAT.

Nech $L \in \mathcal{NP}$. Nech M je príslušný nedeterministický jednopáskový[13] Turingov stroj s polynomiálnou časovou zložitosťou $p(n)$, ktorý akceptuje L. Nech M má stavy $q_0, q_1, \ldots, q_s$ a páskové symboly $X_1, X_2, \ldots, X_m$. Nech w je vstup pre M, nech $|w| = n$. Nech q_s je jediný jeho koncový stav a nech po prejdení do q_s stroj M v ďalších krokoch v tomto stave zotrváva bez posunu hlavy.

Ak M akceptuje w, potom existuje postupnosť konfigurácií $Q_0, Q_1, \ldots, Q_q$, kde $q < p(n)$, Q_0 je počiatočná konfigurácia, Q_q je akceptujúca konfigurácia, $Q_{i-1} \vdash Q_i$ pre $1 \leq i \leq q$ a žiadna konfigurácia nezaberá viac ako $p(n)$ políčok pásky.

Skonštruujeme booleovský výraz w_0, ktorý bude splniteľný práve vtedy, ak existuje príslušná akceptujúca postupnosť konfigurácií (tj. ak Turingov stroj akceptuje w). Nasledujúce premenné použijeme vo výraze w_0 ako propozičné premenné:

$C_{i,j,t}$ $(1 \leq i \leq p(n),\ 1 \leq j \leq m,\ 0 \leq t \leq p(n))$ bude 1 práve vtedy, keď páska M bude pri výpočte v čase t na svojom i-tom políčku obsahovať symbol X_j,

$S_{k,t}$ $(0 \leq k \leq s,\ 0 \leq t \leq p(n))$ bude 1 práve vtedy, keď M bude pri výpočte v čase t v stave q_k,

$H_{i,t}$ $(1 \leq i \leq p(n),\ 0 \leq t \leq p(n))$ bude 1 práve vtedy, keď poloha hlavy M bude pri výpočte v čase t čítať políčko i.

Máme teda $O(p^2(n))$ propozičných premenných. Tieto premenné je teda možné reprezentovať postupnosťami dĺžky $c \log n$ (pre nejakú konštantu c) znakov $\{0, 1\}$.

Pre zjednodušenie si zaveďme predikát

$$U(x_1, x_2, \ldots, x_r) := (x_1 \vee x_2 \vee \ldots \vee x_r) \wedge \left(\bigwedge_{i,j,i \neq j} (\neg x_i + \neg x_j) \right),$$

ktorý je splnený, keď práve jedna premenná z $x_1, \ldots, x_r$ je 1. Všimnime si, že pokiaľ by sme formálne zapísali predikát U, jeho kód by mal pre r premenných dĺžku $O(r^3)$.

Je zrejmé, že postupnosť konfigurácií $Q_0, Q_1, \ldots, Q_{p(n)}$ je akceptujúca sekvencia práve vtedy, keď:

1. v každej konfigurácii hlava sníma práve jedno políčko pásky,
2. na každom políčku pásky sa nachádza naraz práve jeden symbol,
3. v každej konfigurácii sa stroj nachádza v práve jednom stave,

[13] k-páskový Turingov stroj s polynomiálnou časovou zložitosťou možno simulovať jednopáskovým Turingovým strojom pri polynomiálnom náraste časovej zložitosti

Príklad:

$(p \vee q) \wedge (q \vee \neg p \vee r)$	je	v 3-konjunktívnom normálnom tvare
$p \wedge q \vee r$	nie je	v konjunktívnom normálnom tvare
$p \wedge (q \vee r)$	je	v konjunktívnom normálnom tvare

Definícia 5.8 *Nech CSAT resp. k-SAT je množina splniteľných booleovských výrazov v konjunktívnom resp. v k-konjunktívnom normálnom tvare.*

Veta 5.9 *CSAT aj 3-SAT sú $\mathcal{NP}$-úplné jazyky.*

Poznámka: Dôkaz tejto vety je podobný ako dôkaz vety 5.6, stačí príslušné booleovské výrazy vytvárať v 3-konjunktívnom normálnom tvare.

Veta 5.10 *2-SAT patrí do triedy $\mathcal{P}$.*

5.2.2 $\mathcal{NP}$-úplné problémy na neohodnotených grafoch

Definícia 5.11 *Nech $G = (V, E)$ je neorientovaný graf. Vrcholové pokrytie grafu G je podmnožina $V' \subseteq V$ taká, že pre každú hranu $(v, w) \in E$ aspoň jeden z vrcholov v, w patrí do V'.*

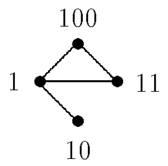
Hranové pokrytie grafu G je podmnožina $E' \subseteq E$ taká, že každý vrchol z V prislúcha niektorej hrane z E'.

Graf G je k-zafarbiteľný, ak možno jeho vrcholy zafarbiť k farbami tak, aby žiadne dva vrcholy spojené hranou nemali rovnakú farbu.

Graf G má k-kliku, ak v G existuje kompletný podgraf s k vrcholmi.

Graf $G = (V, E)$ budeme kódovať tak, že vrcholy grafu očíslujeme číslami $1, 2, \ldots, |V|$ a kód grafu G bude reťazec obsahujúci zoznam vrcholov a zoznam hrán, pričom čísla vrcholov sú v binárnom tvare. Dvojicu (k, G), kde k je nezáporné celé číslo budeme kódovať reťazcom

binárny zápis k; kód grafu G



obr. 10: Graf, ktorého kód je 1, 10, 11, 100, $(1, 10)$, $(1, 100)$, $(1, 11)$, $(100, 11)$

Definícia 5.12 *Nech VP je množina kódov dvojíc (k, G), kde $k \geq 0$ a G je graf majúci vrcholové pokrytie pozostávajúce z k vrcholov. Nech HP je množina kódov dvojíc (k, G), kde G je graf majúci hranové pokrytie pozostávajúce z k hrán. Nech F je množina kódov dvojíc (k, G), kde G je k-zafarbiteľný graf. Nech K je množina kódov dvojíc (k, G), kde G má k-kliku (tj. úplný podgraf s k vrcholmi). Nech HAM je množina kódov grafov G, ktoré majú hamiltonovskú kružnicu*[14].

Veta 5.13 *Jazyky VP, F, K a HAM sú $\mathcal{NP}$-úplné.*

Dôkaz: Dokážeme, že K je $\mathcal{NP}$-úplný jazyk, pri dôkazoch pre jazyky VP, F a HAM sa postupuje obdobne. Zrejme platí, že $K \in \mathcal{NP}$. Stačí teda pre nejaký $\mathcal{NP}$-úplný jazyk L_0 dokázať, že je polynomiálne transformovateľný na jazyk K. Z toho totiž vyplýva, že každý jazyk $L \in \mathcal{NP}$ je polynomiálne transformovateľný na K, lebo každý jazyk $L \in \mathcal{NP}$ je polynomiálne transformovateľný na L_0 a L_0 je transformovateľný na K.

[14] z formálneho hľadiska sú množiny VP, HP, F, K a HAM jazyky nad abecedou $\{0, 1, (,), ;, \text{"čiarka"}\}$

Obrátene, nech G' má hamiltonovskú kružnicu s cenou najviac $k = 0$. Potom takáto hamiltonovská kružnica musí mať cenu 0, lebo cena každej hrany je nezáporná, a teda nemôže obsahovať žiadnu hranu s cenou 1. Z konštrukcie grafu G' vyplýva, že takáto hamiltonovská kružnica je tiež hamiltonovskou kružnicou grafu G.

Preto kód grafu G patrí do HAM práve vtedy, keď kód dvojice $(0, G')$ patrí do POC. □

Definícia 5.18 *POC-OPT (problém obchodného cestujúceho — optimalizačná verzia): Pre daný kód kompletného ohodnoteného grafu treba nájsť hamiltonovskú kružnicu s minimálnou cenou.*

Je zrejmé, že problém POC je polynomiálne transformovateľný na problém POC-OPT (tzn. že ak by existoval deterministický polynomiálny algoritmus riešiaci POC-OPT, potom by tiež existoval deterministický polynomiálny algoritmus akceptujúci jazyk POC).

Platí to aj obrátene, lebo POC-OPT možno vypočítať pomocou algoritmu pre POC. Nech n je dĺžka kódu grafu G. Cena každej hamiltonovskej kružnice grafu G je medzi 0 a $2^n - 1$, lebo ceny hrán sú kódované binárne. Cenu najlacnejšej hamiltonovskej kružnice grafu G nájdeme binárnym prehľadávaním intervalu $0, \ldots, 2^n - 1$ pomocou algoritmu pre POC, tj. najprv zistíme, či $(2^{n-1}, G) \in$ POC, ak áno, potom zisťujeme, či $(2^{n-2}, G) \in$ POC, inak zisťujeme, či $(2^{n-1} + 2^{n-2}, G) \in$ POC, atď., až kým nezistíme cenu najlacnejšej hamiltonovskej kružnice grafu G. Nech jej cena je C. Potom príslušnú hamiltonovskú kružnicu nájdeme pomocou POC takto: zväčšujeme postupne ceny jednotlivých hrán z grafu G o 1 a zakaždým pomocou algoritmu pre POC zistíme, či v takto upravenom grafe existuje hamiltonovská kružnica s cenou najviac C. Ak áno, potom v grafe G existuje hamiltonovská kružnica s cenou C neobsahujúca vybranú hranu. V opačnom prípade je vybraná hrana súčasťou hľadanej hamiltonovskej kružnice s cenou C — v tomto prípade cenu tejto hrany znížime na pôvodnú hodnotu. Takto pokračujeme, až kým nepreskúmame všetky hrany[15].

Definícia 5.19 *$\mathcal{NP}$-optimalizačný problém A je daný cieľom (min, max), polynomiálne ohraničenou reláciou $R \subseteq \Sigma^* \times \Sigma^*$ (t.j. pre R existuje polynóm p taký, že ak $x, y \in R$, potom $|y| \leq p(|x|)$) a hodnotovou funkciou $m : \Sigma^* \times \Sigma^* \to N$, pričom R aj p sú vypočítateľné deterministicky v polynomiálnom čase, t.j. $\{x; y | (x, y) \in R\} \in P$ a pre $(x, y) \in R$ možno hodnotu $m(x, y)$ možno vypočítať deterministicky v polynomiálnom čase (t.j. v čase $q(|x| + |y|)$). Reláciu R môžeme chápať ako problém(vstup) $\times$ riešenie(výstup) a m je kvalita riešenia.*

Príklad: POC-OPT:

- cieľ je min
- $R = \{(x, y) | x$ je kompletný graf, y je hamiltonovská kružnica grafu $x\}$
- $m(x, y) =$ cena hamiltonovskej kružnice y grafu x

Definícia 5.20 *Pre $\mathcal{NP}$-optimalizačný problém (s cieľom min) môžeme definovať tri problémy:*

- *Konštrukčný problém : pre dané x zostroj (ak existuje) y také, že $(x, y) \in R \wedge m(x, y) = \min_{y' \in \Sigma^*} \{m(x, y')\}$*
- *Hodnotový problém : pre dané x a $k \in N$ zisti, či $(\exists y)((x, y) \in R \wedge m(x, y) \leq k)$*
- *Rozhodovací problém : pre dané x a $k \in N$ zisti, či $(\exists y)((x, y) \in R \wedge m(x, y) \leq k)$. Rozhodovací problém môžeme formulovať ako jazyk $L = \{x; \text{zápis } k \,|\, x \in \Sigma^* \wedge k \in N \wedge (\exists y)((x, y) \in R \wedge m(x, y) \leq k)\}$*

Poznámka: Pre $\mathcal{NP}$-optimalizačné problémy s cieľom max je možné definovať konštrukčný, hodnotový a rozhodovací problém analogicky.

[15] Všimnite si, že v prípade, že existuje kružnica s cenou najviac C, ktorá neobsahuje príslušnú hranu, **neznížime** cenu tejto hrany na pôvodnú hodnotu. Ak by totiž v grafe G existovalo viacero hamiltonovských kružníc s cenou C, nenašli by sme všetky hrany žiadnej z nich.

6 Aproximatívne algoritmy

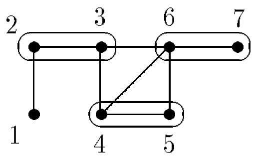
Definícia 6.1 *Problém VP-OPT (problém vrcholového pokrytia — optimalizačná verzia): Pre daný kód grafu G treba nájsť vrcholové pokrytie s minimálnym počtom vrcholov.*

Je zrejmé, že problém VP-OPT nie je ľahší než $\mathcal{NP}$-úplný problém VP. Preto nepoznáme žiadny deterministický polynomiálny algoritmus riešiaci problém VP-OPT.

Preto sa má v tomto prípade zmysel zaoberať aproximačnými algoritmami pre VP-OPT. Uvedieme aproximačný deterministický polynomiálny algoritmus, ktorý pre daný graf $G = (V, E)$ nájde vrcholové pokrytie $\tilde{V}$, pre ktoré platí, že $|\tilde{V}| \leq 2|V^*|$, kde V^* je vrcholové pokrytie grafu G s minimálnym počtom vrcholov.

Algoritmus 14

begin
  $\tilde{V} \leftarrow \emptyset$ *(1)*
  $E' \leftarrow E$ *(2)*
  while $E' \neq \emptyset$ **do begin** *(3)*
    nech (u, v) *je ľubovoľná hrana z* E' *(4)*
    $\tilde{V} \leftarrow \tilde{V} \cup \{u, v\}$ *(5)*
    odstráň z E' *každú hranu incidentnú s vrcholom* u *alebo* v *(6)*
  end
end



obr. 12: Príklad činnosti algoritmu pre VP-OPT, $\tilde{V} = \{2, 3, 4, 5, 6, 7\}$, $V^* = \{2, 4, 6\}$, vybrané hrany $(2, 3), (4, 5), (6, 7)$.

Veta 6.2 *Algoritmus 14 nájde v polynomiálnom čase vrcholové pokrytie $\tilde{V}$ grafu G, pričom platí $|\tilde{V}| \leq 2|V^*|$, kde V^* je optimálne vrcholové pokrytie.*

Dôkaz: $\tilde{V}$ je zrejme vrcholové pokrytie grafu G (z E' sú postupne odstraňované všetky hrany pokryté vrcholmi pridanými do $\tilde{V}$).

Dokážeme, že $|\tilde{V}| \leq 2|V^*|$. Nech $A \subseteq E$ je množina hrán, ktoré sú vybrané v riadku 4. Z riadku 6 vyplýva, že žiadne dve hrany z A nemajú spoločný vrchol. Keďže s každou vybranou hranou pribudnú do $\tilde{V}$ dva vrcholy (pozri riadok 5), platí $|\tilde{V}| = 2|A|$.

Keďže vrcholy z V^* pokrývajú všetky hrany z E (a teda aj z A) a žiadne dve hrany z A nemajú spoločný vrchol, musí platiť, že $|V^*| \geq |A|$, teda $2|V^*| \geq 2|A| = |\tilde{V}|$. □

Definícia 6.3 *Kompletný ohodnotený graf $G = (V, V \times V, c)$ spĺňa trojuholníkovú nerovnosť, ak pre každé tri vrcholy $u, v, w \in V$ platí*

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

.

- $0-1$ Knapsack je dobre aproximovateľný.
- Bin-packing je $\frac{3}{2}$ aproximovateľný (pozri cvičenie 6.1), ale nie je $\frac{3}{2}-\epsilon$ aproximovateľný (ak $\mathcal{P} \neq \mathcal{NP}$).
- Neaproximovateľné problémy (ak $\mathcal{P} \neq \mathcal{NP}$):
  - maximálna klika grafu
  - $0-1$ programovanie
  - najdlhšia cesta v grafe medzi dvoma vrcholmi
  - problém obchodného cestujúceho bez trojuholníkovej nerovnosti

Cvičenia

Cvičenie 6.1 Bin-packing: Daný je objem škatule S a N predmetov s objemami $a_1, \ldots, a_N$. Nech K^* je najmenší počet škatúľ, do ktorých je možné poukladať uvedené predmety tak, že v každej škatuli sa nachádzajú predmety s celkovým objemom nanajvýš S. Nájdite algoritmus, ktorý nájde nejaké zabalenie predmetov do škatúľ, pričom ak počet použitých škatúľ označíme K, musí platiť

a) $K \leq 2K^*$,

b) $K \leq \lceil \frac{3}{2}K^* \rceil$.