FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITA KOMENSKÉHO

Pavol Ďuriš

Tvorba efektívnych algoritmov

Máj 2009

Autor: Pavol Ďuriš
Názov: Tvorba efektívnych algoritmov
Vydavateľ: Knižničné a edičné centrum FMFI UK
Rok vydania: 2009
Miesto vydania: Bratislava
Vydanie: prvé
Počet strán: 46
Internetová adresa: http://www.fmph.uniba.sk/index.php?id=el_st_m
ISBN: 978-80-89186-50-1

Obsah

1 Vyhľadávanie, triedenie a súvisiace problémy

V tejto kapitole sa budeme venovať triedeniu a vyhľadávaniu. Oproti prednáške Älgoritmy a štruktúry údajovüvedieme niektoré ďalšie algoritmy (hľadanie k-teho najmenšieho prvku) a zavedieme niektoré nové dátové štruktúry (2-3 stromy, štruktúry pre UNION/FIND-SET problém). Takisto rozšírime vedomosti o zložitosti problému triedenia (dolný odhad priemerného prípadu).

1.1 Hľadanie k-teho najmenšieho prvku

Nech U je lineárne usporiadaná množina. Nad touto množinou majme danú n-prvkovú postupnosť S. Nie je ťažké napísať algoritmus, ktorý nájde minimálny (resp. maximálny) prvok takejto množiny v čase $O(n)$. Takisto nie je problémom v takom istom čase nájsť druhý najmenší prvok (na riešenie stačí pridať jednu premennú a pre každý prvok jedno porovnanie).

Vezmime si teraz o niečo všeobecnejšiu úlohu: dané je celé číslo k $(1 \leq k \leq n)$. Je potrebné nájsť v postupnosti S jej k-ty najmenší prvok.

Ak budeme postupovať v duchu predchádzajúcich úvah, dospejeme k nasledujúcemu algoritmu: v pamäti budeme uchovávať doteraz nájdených k najmenších prvkov. Postupne budeme prechádzať postupnosť a pre každý prvok pomocou k operácií zaktualizujeme uvedenú štruktúru. Takýmto spôsobom dostávame algoritmus o časovej zložitosti $O(k.n)$, a teda vo všeobecnosti až $O(n^2)$. Tento výsledok je neuspokojivý.

Ďalším prirodzeným riešením je utriediť celú postupnosť a potom sa jednoducho pozrieť na k-ty najmenší prvok. Takéto riešenie nám dáva časovú zložitosť $\Theta(n. \log n)$. Otázkou však zostáva, či nie je možné nájsť algoritmus s časovou zložitosťou porovnateľnou s hľadaním minimálneho (doteraz uvedené výsledky sú totiž rádovo horšie ako $O(n)$). Skutočne v ďalšom texte ukážeme, že existuje algoritmus s časovou zložitosťou $O(n)$.

Idea rýchleho algoritmu spočíva v použití metódy "Rozdeľuj a panuj". Problém o rozsahu n zredukujeme na jeden podproblém rozsahu $n/5$ a jeden podproblém rozsahu najviac $3n/4$.

Definícia 1.1 *Medián n-prvkovej postupnosti je jej $\lceil n/2 \rceil$-tý najmenší prvok.*

Algoritmus 1

procedure *SELECT(k,S)*
begin
  if $|S| < 50$ **then begin**
    utrieď S
    return *k-ty najmenší prvok v utriedenej postupnosti*
  end
  else begin
    rozdeľ S do $\lfloor |S|/5 \rfloor$ päťprvkových postupností
    utrieď päťprvkové postupnosti
    nech M je postupnosť mediánov päťprvkových postupností
    $m \leftarrow$ *SELECT*$(\lceil |M|/2 \rceil, M)$ (1)
    rozdeľ prvky z S do troch postupností S_1, S_2, S_3 tak, aby
      S_1 obsahovala všetky prvky z S menšie než m
      S_2 obsahovala všetky prvky z S rovné m
      S_3 obsahovala všetky prvky z S väčšie než m
    if $|S_1| \geq k$ **then return** *SELECT(k,S_1)* (2)
    else
      if $|S_1| + |S_2| \geq k$ **then return** m

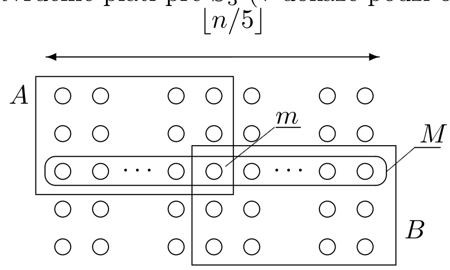
else return *SELECT*($k - |S_1| - |S_2|, S_3$) *(3)*
end
end

Lema 1.2 *Pre veľkosti množín* S_1 *a* S_3 *v algoritme 1 platí:* $|S_1| \leq 3n/4$, $|S_3| \leq 3n/4$.

Dôkaz: Utrieďme[1] jednotlivé päťprvkové postupnosti (každá z nich je utriedená podľa veľkosti prvkov) podľa ich mediánov — pozri obr. 1 (stĺpce predstavujú jednotlivé päťprvkové postupnosti). Obdĺžnik B obsahuje $3\lceil \lfloor n/5 \rfloor /2 \rceil$ prvkov a hodnota každého z nich je aspoň m. Preto

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

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



Obrázok 1: Odhad veľkostí $|S_1|, |S_3|$

V nasledujúcom tvrdení sa budeme zaoberať časovou zložitosťou uvedeného algoritmu.

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 $|S_1| \leq 3n/4$ a $|S_3| \leq 3n/4$ (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

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

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

Dôkaz korektnosti algoritmu prenechávame na čitateľa. □

V ďalšom uvedieme tvrdenia, ktoré ukazujú, že nie je možné zostrojiť algoritmus s rádovo 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.*

[1]Všimnite si, že algoritmus netriedi mediány päťprvkových postupností, namiesto toho hľadá prvok m ako medián množiny M

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?

Cvičenie 1.3 Uvažujte takýto algoritmus pre hľadanie k-teho najmenšieho prvku:

procedure *SEL(k,S)*
begin
  if $|S| < 50$ **then begin**
    utrieď S
    return *k-ty najmenší prvok v utriedenej postupnosti*
  end
  else begin
    $m \leftarrow$ *ľubovoľný prvok z S*
    rozdeľ prvky z S do troch postupností S_1, S_2, S_3 tak, aby
      S_1 obsahovala všetky prvky z S menšie než m
      S_2 obsahovala všetky prvky z S rovné m
      S_3 obsahovala všetky prvky z S väčšie než m
    if $|S_1| \geq k$ **then return** *SELECT(k,S_1)*
    else
      if $|S_1| + |S_2| \geq k$ **then return** m
      else return *SELECT($k - |S_1| - |S_2|, S_3$)*
  end
end

Aký je počet porovnaní algoritmu SEL v najhoršom a priemernom prípade?

Cvičenie 1.4 Sú dané dve polia $A[1..N]$, $B[1..N]$ rovnakej dĺžky N. Obe polia sú vzostupne utriedené. Nájdite čo najefektívnejší algoritmus, ktorý nájde medián pre postupnosť, ktorá by vznikla zlúčením oboch týchto polí ($O(\log n)$).

Cvičenie 1.5 V krajine je postavených N vrtných veží. Nech i-ta veža má súradnice (x_i, y_i) (možno predpokladať, že žiadne dve veže nemajú rovnakú x-ovú súradnicu). Inžinieri sa rozhodli postaviť dlhú rúru vedúcu od východu na západ a ku každej vrtnej veži postaviť prípojku. Nájdite čo najefektívnejší algoritmus, ktorý určí, na akú y-ovú súradnicu je potrebné rúru postaviť, aby súčet dĺžok prípojok k vežiam bol minimálny ($O(n)$).

1.2 Priemerný počet porovnaní triediacich algoritmov

Pri skúmaní počtu porovnaní triediacich algoritmov sme sa doteraz stretli s týmito tvrdeniami:

- Na utriedenie n prvkov **v priemernom prípade stačí** $O(n \log n)$ porovnaní (viď. vlastnosti algoritmov HEAPSORT a QUICKSORT).
- Na utriedenie n prvkov **v najhoršom prípade je potrebných** $\Omega(n \log n)$ (viď. rozhodovacie stromy) a **stačí** $O(n \log n)$ porovnaní (viď. vlastnosti algoritmu HEAPSORT).

V tejto kapitole sa budeme zaoberať priemerným prípadom triediacich algoritmov a stanovíme, aký počet porovnaní je **potrebných v priemernom prípade** na utriedenie n prvkov.

Definícia 1.7 *Striktne binárny strom je strom, v ktorom má každý vrchol okrem listov práve dvoch synov.*

Označenie: Nech D_T je suma hĺbok listov stromu T.
Nech $d(m) = \min\{D_T \mid T$ je striktne binárny strom s m listami$\}$

Lema 1.8 *Nech T_R je rozhodovací strom s m listami. Potom $D_{T_R} \geq m \log m$.*

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. Označme počet listov stromu T_1 i, 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, to isté 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:

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

$\square$

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

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

$\square$

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.

1.3 Algoritmy na dynamických množinách

Na dynamických množinách (dynamické preto, lebo povoľujeme aj operácie, ktoré menia tieto množiny) budeme uvažovať tieto základné operácie

MEMBER(a,S) zistí, či prvok a patrí do množiny S,

INSERT(a,S) do množiny S pridá prvok a,

DELETE(a,S) z množiny S odoberie prvok a,

MIN(S) nájde najmenší prvok množiny S,

UNION(S_1,S_2) vytvorí zjednotenie množín S_1 a S_2 (predpokladáme, že množiny S_1 a S_2 sú disjunktné),

FIND-SET(a) nájde množinu S, do ktorej patrí a.

Mnohé praktické problémy možno redukovať na podproblémy, ktoré možno abstraktne formulovať ako postupnosť uvedených základných operácií na nejakej dynamickej množine.

Príklad: Pri lexikálnej analýze kompilátory často používajú operácie MEMBER, INSERT; niektoré editory umožňujú kontrolu preklepov (operácie MEMBER, INSERT); mnohé greedy algoritmy používajú operácie UNION, FIND-SET.

Definícia 1.10 *Nech σ je konečná postupnosť základných operácií na dynamických množinách. Časová zložitosť postupnosti σ je množstvo času (vyjadrené ako funkcia dĺžky postupnosti σ), ktoré treba na vykonanie inštrukcií postupnosti σ.*

Poznámka: Postupnosť σ vykonávame on-line spôsobom, tzn. algoritmus sa nemôže "pozrieť"na j-tu operáciu v σ skôr, ako vykoná operácie $1, 2, \ldots, j - 1$.

[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 $(n, 0, 0, 0) \to (0, 1, 1, n - 2)$.

1.3.1 Realizácia slovníka hashovaním

Od slovníka požadujeme, aby čo najefektívnejšie dokázal realizovať ľubovoľnú postupnosť operácií skladajúcu sa z operácií MEMBER, INSERT a DELETE. Pomerne jednoduchým riešením problému je použitie hashovania.

Veta 1.11 *Ak hashovacia funkcia $h : U \to \{0, 1, \ldots, m-1\}$ zobrazuje U rovnomerne na množinu $\{0, 1, \ldots, m-1\}$, potom priemerná zložitosť postupnosti σ dĺžky $n \leq m$ je $O(n)$ (predpokladáme, že množina reprezentujúca slovník je na začiatku vykonania postupnosti σ prázdna).*

Dôkaz: Všetky (vzhľadom na hashovaciu funkciu) rôzne výpočty na n prvkových vstupných postupnostiach $a_1 \ldots a_n$ možno reprezentovať úplným m-árnym stromom výšky n (koreň je začiatok výpočtu, list koniec výpočtu). Každej hrane stromu prideľme cenu takto: Nech w je ľubovoľný vrchol hĺbky i $(0 \leq i \leq n-1)$ a nech tomuto vrcholu zodpovedá (vzhľadom na časť výpočtu koreň – vrchol w) stav, kde dĺžka j-teho zoznamu je l_j $(1 \leq j \leq m)$. Potom cena hrany z w do jeho j-teho syna je $c_j = l_j + 1$. Keďže $\sum_{j=1}^{m} l_j = i$, potom

$$\sum_{j=1}^{m} c_j = i + m$$

Počet vrcholov hĺbky i je m^i $(0 \leq i \leq n)$. Z každého vrcholu hĺbky i vychádza smerom k jeho synom m hrán a suma cien týchto hrán je $i + m$.

Každá z hrán vychádzajúca z niektorého vrcholu s hĺbkou i (smerom k jeho synom) leží na m^{n-i-1} cestách z koreňa k listom.

Nazvime cenou cesty z koreňa do listu súčet cien hrán ležiacich na tejto ceste.

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 $h(a_1) = h(a_2) = \ldots = h(a_n)$.

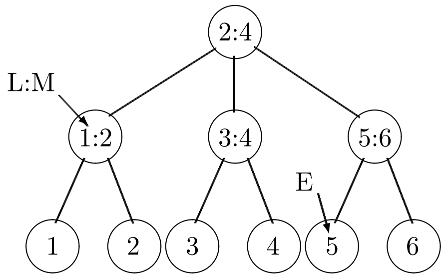
1.3.2 Realizácia slovníka pomocou 2-3 stromov

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.



Obrázok 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. Napíšeme 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.

Algoritmus 2

procedure *SEARCH(a,v)*
begin
    if *každý syn vrcholu v je list* **then return** v
    else begin
        $s_i \leftarrow$ *i-ty syn vrcholu v*
        if $a \leq L[v]$ **then return** *SEARCH(a,s_1)*
        else
            if *v má dvoch synov* **or** $a \leq M[v]$ **then return** *SEARCH(a,s_2)*
            else return *SEARCH(a,s_3)*
    end
end

Pomocou algoritmu 2 teraz ľahko zrealizujeme procedúru MEMBER. Procedúra zistí, či je prvok a v množine S reprezentovanej 2-3 stromom s koreňom v.

Algoritmus 3

procedure *MEMBER(a,v)*
begin
    $w \leftarrow$*SEARCH(a,v)*

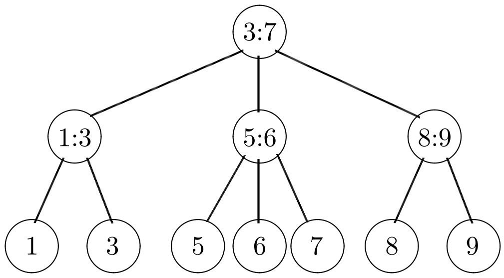
$l_i \leftarrow$ *i-ty syn vrcholu* w
if $E[l_i] = a$ *pre nejaké* i **then return** *"áno"*
else return *"nie"*

end

Lema 1.14 *Algoritmus 3 zistí, či 2-3 strom* T *s* n *listami obsahuje list s hodnotou* a *v najhoršom prípade v čase* $O(\log n)$.

Dôkaz: Lema je dôsledkom lemy 1.13. □

Operácia INSERT. Nech v je koreň 2-3 stromu reprezentujúceho množinu S. Nech prvok a nepatrí do množiny S.



Obrázok 3: 2-3 strom T reprezentujúci množinu $S = \{1, 3, 5, 6, 7, 8, 9\}$

Nech f je výsledkom procedúry SEARCH(a,v). Vytvoríme nový list s hodnotou a a pripojíme ho ako syna k vrcholu f tak, aby nebolo porušené usporiadanie hodnôt v listoch stromu. Môže nastať jedna z nasledujúcich možností:

1. f má troch synov. V tomto prípade sme získali 2-3 strom reprezentujúci $S \cup \{a\}$.

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.*

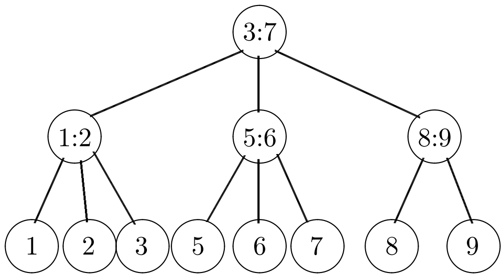
Dôkaz: Nech T je 2-3 strom s n listami. Z lemy 1.13 vyplýva, že výška stromu T je nanajvýš $\log n$. Preto nájdenie vrcholu $f =$SEARCH(a,v), kde v je koreň stromu T, potrebuje čas $O(\log n)$. Vsunutie nového listu do stromu potrebuje čas $O(1)$. Následná možná úprava na 2-3 strom, pri ktorej algoritmus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

Druhá časť lemy vyplýva priamo z realizácie algoritmu INSERT. □

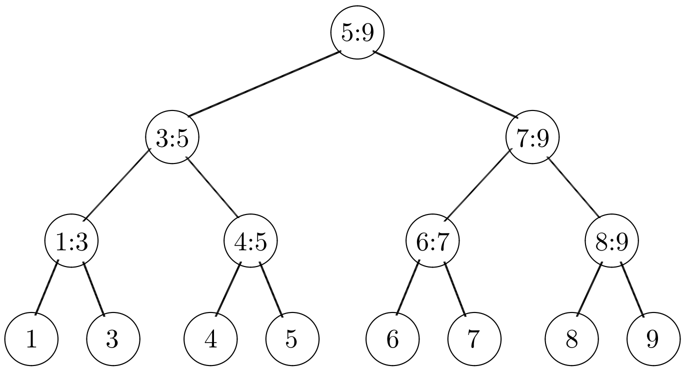
Operácia DELETE. Nech v je koreňom 2-3 stromu reprezentujúceho množinu S, nech $a \notin S$. Nech l je list s hodnotou a. Nech ďalej f je výsledkom procedúry SEARCH(a,v)[4].

Môže nastať jedna z nasledujúcich možností:

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

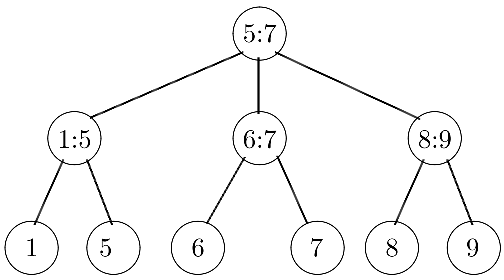


Obrázok 4: Výsledok operácie INSERT(2,v) pre strom T



Obrázok 5: Výsledok operácie INSERT(7,v) pre strom T

1. f má troch synov. Potom možno odstrániť l a skončiť.
2. f má dvoch synov l a s. Nech ďalej f má ľavého brata g (v prípade pravého brata postupujeme obdobne). Potom nastáva jedna z týchto možností:
   - g má troch synov. Potom odpojíme od g jeho najpravejšieho syna, pripojíme ho ku f ako najľavejšieho syna, odstránime l a skončíme.
   - g má dvoch synov. Potom odpojíme s od f a pripojíme s na g ako najpravejšieho syna. Potom odstránime l a pokračujeme rekurzívne smerom ku koreňu stromu odstraňovaním f.



Obrázok 6: Výsledok operácie DELETE(3,v) pre strom T

Poznámka: V algoritme pre DELETE takisto ako v predchádzajúcich prípadoch netreba zabudnúť na aktualizáciu hodnôt L a M.

Lema 1.16 *Algoritmu pre DELETE odstráni prvok z 2-3 stromu s n listami v čase $O(\log n)$. Naviac tento algoritmus upraví pôvodný 2-3 strom na 2-3 strom s $n-1$ listami.*

Dôkaz: Dôkaz je obdobný ako u lemy 1.15. □

Implementačné poznámky. Vrcholy 2-3 stromu obvykle implementujeme ako záznamy s týmito atribútmi: pointer na otca, pointre na synov, hodnoty L, M a E. Vstupnými parametrami pre operácie MEMBER, INSERT a DELETE sú potom hodnota prvku a a pointer ukazujúci na koreň 2-3 stromu T a ktorý po ukončení operácie bude ukazovať na nový vrchol stromu.

Veta 1.17 *Časová zložitosť postupnosti σ, ktorá sa skladá z n operácií typu MEMBER, INSERT, DELETE alebo MIN, je v najhoršom prípade $O(n \log n)$.*

Dôkaz: V priebehu vykonávanie operácií v σ počet listov v príslušných 2-3 stromoch neprekočí n. Keďže prvok s najmenšou hodnotou sa nachádza v najľavejšom liste 2-3 stromu, operáciu MIN možno vykonať v najhoršom prípade v čase $O(\log n)$. Ďalej tvrdenie vety vychádza z lemy 1.14, 1.15 a 1.16. □

1.3.3 UNION/FIND-SET problém

Majme množiny $S_i = \{i\}$ pre všetky $i = 1, 2, \ldots, n$. Úlohou bude v tomto prípade zostrojiť algoritmus, ktorý by (čo najefektívnejšie) vykonal ľubovoľnú postupnosť operácií typu UNION(S_i,S_j) a FIND-SET(l).

Príklad: Operácie UNION/FIND-SET možno použiť napríklad pri hľadaní súvislých komponentov grafu. Algoritmus s využitím týchto operácií by vyzeral asi takto:

> *pre každý vrchol $v \in V$ vytvor množinu $\{v\}$*
> *pre každú hranu $(u, v) \in E$:*
>   **if** *FIND-SET(u)$\neq$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 $p[1 \ldots n]$, $h[1 \ldots n]$ 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 *Časová zložitosť* n*-prvkovej postupnosti* σ *skladajúcej sa z operácií UNION a FIND-SET je v najhoršom prípade* $O(n \log n)$.

Dôkaz: Podľa lemy 1.19 má každý strom vytvorený počas vykonávania postupnosti σ výšku nanajvýš $\log_2 n + 1$. Preto je možné vykonať každú operáciu FIND-SET v σ v čase $O(\log n)$. Operáciu UNION možno vykonať v čase $O(1)$. □

Poznámka: V prípade, keď S_i nie sú tvaru $S_i = \{i\}$, ale napríklad S_i obsahujú reálne čísla, textové reťazce apod., možno ich prvky zotriediť a potom v čase $O(\log n)$ binárnym vyhľadávaním nájsť príslušné číslo množiny.

1.3.4 Zrýchlenie algoritmu pre UNION/FIND-SET problém

Nech j je vrchol stromu T s koreňom l. Modifikujeme algoritmus 5 (procedúru FIND-SET(j)) tak, aby každý vrchol vyskytujúci sa na ceste z vrcholu j do koreňa l bol odpojený od svojho otca a napojený priamo na koreň l.

Túto metódu nazývame metódou *kompresie cesty* a môžeme ju realizovať takto:

Algoritmus 6

procedure *FIND-SET*(j)

```
begin
    if p[j] ≠ j then p[j] ←FIND-SET(p[j])
    return p[j]
end
```

Modifikujme taktiež algoritmus 4 (procedúru UNION(l_i,l_j)) tak, aby strom reprezentujúci množinu $S_i \cup S_j$ bol vytvorený nie podľa kritéria výšok ale podľa počtu vrcholov stromov T_i a T_j[5].

Definícia 1.21 *Nech* $F(0) = 1$ *a nech* $F(i+1) = 2^{F(i)}$ *pre* $i \geq 0$. *Potom* $\log^* n := min\{k|F(k) \geq n\}$.

Poznámka: $\log^* n$ je extrémne pomaly rastúca funkcia, napríklad $\log^* n \leq 5$ pre všetky $n \leq 2^{65536}$.

Veta 1.22 *Časová zložitosť n-prvkovej postupnosti* σ *skladajúcej sa z n operácií UNION a FIND-SET je v najhoršom prípade* $O(n \log^* n)$, *ak použijeme modifikované algoritmy UNION a FIND-SET.*

Cvičenia

Cvičenie 1.10 Majme postupnosť k základných slovníkových operácií, pričom tieto operácie pracujú iba s číslami od 1 po n. V takomto prípade je možné udržiavať pole $A[1..n]$ tak, aby $A[i] = 1$ práve vtedy, keď $i \in S$. Toto pole je potrebné na začiatku inicializovať a tak tento algoritmus má časovú zložitosť $O(n+k)$. Je možné modifikovať tento algoritmus tak, aby jeho časová zložitosť bola $O(k)$ (tj. odstrániť inicializačnú fázu)?[6]

Cvičenie 1.11 Koľko existuje 2-3 stromov reprezentujúcich množinu čísel $\{1, \ldots, 6\}$?

Cvičenie 1.12 Pokúste sa detailne analyzovať riešenie UNION/FIND-SET problému, ak pri operácii UNION používame kritérium počtu vrcholov (tak ako v kapitole 1.3.4).

Cvičenie 1.13 Pokúste sa detailne analyzovať riešenie UNION/FIND-SET problému so skracovaním cesty (ale s pôvodnou operáciou UNION), ak predpokladáme, že najprv vykonáme všetky operácie UNION a potom všetky operácie FIND-SET.

Cvičenie 1.14 Majme N šúľkov dynamitu a postupnosť operácií SPOJ(i,j) (spojiť šúľky i a j zápalnou šnúrou) a ROZPOJ(i,j) (rozpojiť šúľky i, j, ak sú zápalnou šnúrou spojené). Dynamit možno odstreliť, ak medzi každými dvoma šúľkami vedie cesta po zápalných šnúrach. Nájdite čo najefektívnejší algoritmus, ktorý zistí, či po vykonaní operácie možno dynamit odpáliť, alebo nie.

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).

---

[5] $h[l_i]$ a $h[l_j]$ budú teraz obsahovať namiesto výšok stromov ich počty vrcholov

[6] Hint: Pomocou druhého poľa sa pokúste sa rozlíšiť nenainicializovaný prvok poľa a nainicializovaný.

1.4 Ďalšia literatúra

Referencie

[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

2 Grafové algoritmy

Množstvo praktických problémov možno sformulovať v pojmoch teórie grafov. Z tohto hľadiska má táto časť teórie algoritmov mimoriadne veľký praktický význam. V tejto kapitole rozdiskutujeme niektoré zo základných problémov, ktoré majú riešenie v polynomiálnej časovej zložitosti (minimálna kostra, najkratšie cesty).

2.1 Najlacnejšia kostra grafu

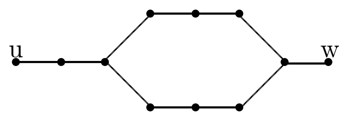
Definícia 2.1 *Nech $G = (V, E)$ je neorientovaný súvislý graf s ohodnotenými hranami, (tj. pre G je daná cenová funkcia $h : E \rightarrow R$; R je množina reálnych čísel).*

1. *Kostra grafu G je ľubovoľný neorientovaný strom (V, T), $T \subseteq E$, spájajúci všetky vrcholy z V (t.j. ľubovoľné dva vrcholy z V sú spojené cestou v strome (V, T)).*
2. *Cena kostry je $\sum_{e \in T} h(e)$.*
3. *Kostrový les pre graf G je ľubovoľná množina stromov $\{(V_1, T_1), \ldots, (V_k, T_k)\}$, $k \geq 1$ taká, že $V = \cup_{i=1}^{k} V_i$, $V_i \cap V_j = \emptyset$ pre $i \neq j$, v každom strome (V_i, T_i) sú spojené všetky vrcholy z V_i a $T_i \subseteq E \cap (V_i \times V_i)$ pre každé i (každý strom (V_i, T_i) je kostra grafu $(V_i, E \cap (V_i \times V_i))$).*

Lema 2.2 *Nech $G = (V, E)$ je súvislý neorientovaný graf a nech $S = (V, T)$ je kostra grafu G. Potom:*

1. *Pre všetky $u, w \in V$ je cesta medzi u a w v S jediná.*
2. *Po pridaní ľubovoľnej hrany z $E - T$ do S vznikne jediná kružnica.*

Dôkaz: Časť 1 vyplýva z toho, že keby boli v S dve cesty medzi u a w, potom by bola v S kružnica.



Obrázok 7: Ak sú medzi u a w dve cesty, existuje v grafe kružnica

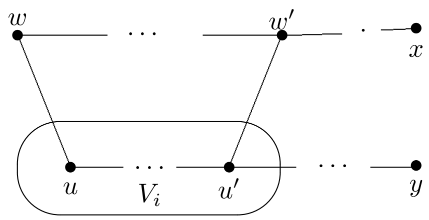
Časť 2. Keďže S je kostra (t.j strom spájajúci všetky vrcholy), existuje medzi ľubovoľnými vrcholmi jediná cesta (pozri časť 1) a preto po pridaní ľubovoľnej hrany z $E - T$ musí vzniknúť jediná kružnica. □

Lema 2.3 *Nech $G = (V, E)$ je súvislý neorientovaný graf a h je cenová funkcia na hranách E. Nech $\{(V_1, T_1), \ldots, (V_k, T_k)\}$, $k > 1$ je kostrový les pre graf G. Nech $H = \cup_{i=1}^{k} T_i$. Nech (u, w) je najlacnejšia hrana z $E - H$ taká, že $\exists i$, $1 \leq i \leq k$,$u \in V_i$ a $w \notin V_i$. Potom existuje kostra grafu G obsahujúca všetky hrany z $H \cup \{(u, w)\}$, ktorej cena nie je väčšia než cena najlacnejšej kostry grafu G obsahujúcej všetky hrany z H.*

Dôkaz: Nech $S = (V, T)$ je ľubovoľná najlacnejšia kostra grafu G obsahujúca hrany z H. Ak T obsahuje hranu (u, w), potom lema 2.3 platí.

Nech teda $(u, w) \notin T$. Z lemy 2.2 časť 2 vyplýva, že pridanie hrany (u, w) do T vytvorí jedinú kružnicu (pozri obr. 2.1). Táto kružnica musí obsahovať nejakú hranu (u', w') takú, že $u' \in V_i$ a $w' \notin V_i$. Podľa predpokladu $h(u,w) \le h(u',w')$, lebo $(u',w') \notin \cup_{i=1}^{k} T_i = H$.

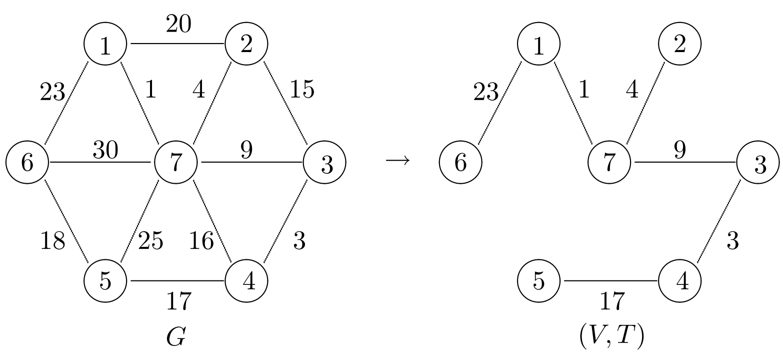
Nech $S' = (V, T')$, kde $T' = (T \cup \{(u,w)\}) - \{(u',w')\}$. S' nemá kružnicu, lebo jediná kružnica bola prerušená odstránením hrany (u',w'). Naviac, všetky vrcholy vo V sú v grafe S' spojené, lebo existuje cesta medzi u' a w' v S' (cesta medzi vrcholmi x a y, ktora v S viedla cez hranu (u',w'), vedie v S' cez vrcholy $w', \ldots, w, u, \ldots, u'$, viď obr. 2.1). 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*$(u) \ne$ *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



Obrázok 8: Príklad Kruskalovho algoritmu

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

Dôkaz:

Správnosť programu. Indukciou na počet vykonaných cyklov v riadkoch (5) až (8) (t.j. na počet hrán pridaných do T) možno dokázať, že po vykonaní l cyklov ($l = 0, 1, 2, \ldots, |V| - 2$) sú splnené predpoklady lemy 2.3 (pre $k = |V| - l$ - každá z množín V_i je niektorá množina FIND-SET(v) pre $v \in V$). Vykonanie jedného cyklu totiž spôsobí (volaním procedúry UNION) nahradenie množín V_i a V_j množinou $V_i \cup V_j$ práve vtedy, keď hrana (u, w), pre ktorú platí $V_i =$ FIND-SET$(u) \neq$ FIND-SET$(w) = V_j$ spája stromy (V_i, T_i) a (V_j, T_j). Teda nový kostrový les (pre lemu 2.3) možno dostať z kostrového lesa $\{(V_1, T_1), \ldots, (V_k, T_k)\}$ nahradením stromov (V_i, T_i) a (V_j, T_j) stromom $(V_i \cup V_j, T_i \cup T_j \cup \{(u, q)\})$. Preto z lemy 2.3 vyplýva, že algoritmus nájde najlacnejšiu kostru grafu G.

Časová zložitosť. Inicializácia v riadkoch (1) a (2) potrebuje čas $O(|V|)$ a triedenie v riadku (3) potrebuje čas $O(|E| \log |E|)$. V riadkoch (5) až (8) sa vyskytne najviac $O(|E|)$ operácií FIND-SET a UNION, ktorých vykonanie vyžaduje čas najviac $O(|E| \log |E|)$ resp. $O(|E| \log^* |E|)$ (pozri vetu 1.22 o časovej zložitosti UNION, FIND-SET problému). Algoritmus vykoná príkaz v riadku (7) práve $(|V| - 1)$krát, pričom vykonanie jedného príkazu potrebuje čas $O(1)$. Teda celková zložitosť algoritmu je v najhoršom prípade $O(|E| \log |E|)$, lebo $|V| - 1 \leq |E|$ pre súvislé grafy. □

Poznámka: Existuje iný algoritmus pre najlacnejšiu kostru grafu so zložitosťou $O(|E| + |V| \log |V|)$, ktorý je výhodný pre husté grafy (t.j. grafy s veľkým počtom hrán).

Cvičenia

Cvičenie 2.1 Nech je daný graf $G = (V, E)$, $V = \{1, 2, \ldots, n\}$ a cenová funkcia h taká, že

$$h(u, v) = \begin{cases} \text{cena hrany } (u, v), & \text{ak } (u, v) \in E, \\ \infty & \text{inak.} \end{cases}$$

Nech $h(u, v) > 0$ pre všetky $u, v \in V$. Uvažujme nasledovný algoritmus:

begin
    $q \leftarrow 0$
    $S \leftarrow \{v_0\}$
    $D[v_0] \leftarrow 0$
    pre každý vrchol $v \in V \setminus \{v_0\}$: $D[v] \leftarrow h(v_0, v)$
    while $S \neq V$ **do begin**
        vyber $w \in V \setminus S$ *taký, že hodnota* $D[w]$ *je minimálna*
        $S \leftarrow S \cup \{w\}$
        $q \leftarrow q + D[w]$
        pre každý $v \in V \setminus S$:
            $D[v] \leftarrow min\{D[v], h(w, v)\}$
    end
    return q
end

Dokážte, že tento algoritmus počíta cenu najlacnejšej kostry grafu G[7].

[7] Všimnite si nápadnú podobnosť s algoritmom 8

Cvičenie 2.2 Odhadnite časovú zložitosť algoritmu z cvičenia 2.1 a porovnajte ju s časovou zložitosťou algoritmu 7. Kedy je výhodnejšie použiť algoritmus z cvičenia 2.1 a kedy algoritmus 7?

Cvičenie 2.3 V cvičení 2.1 sme predpokladali, že všetky hrany majú kladné ohodnotenia. Je možné tento algoritmus použiť aj pre hrany, ktoré majú záporné ohodnotenia? Ak nie, je možné ho upraviť tak, aby sa dal použiť?

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 $i = 1, 2, \ldots, k$ a $v_i \neq v_j$ pre $i \neq j$. Cena cesty $P = [v_0, v_1, \ldots, v_k]$ je $\sum_{i=1}^{k} h(v_{i-1}, v_i)$. Označme cenu cesty P 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 ciesty 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 $(v_i, vi + 1) \notin E$, 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 $D[v]$.

2. Ak $v \in V \setminus S$, potom $D[v]$ je cena najlacnejšej cesty z v_0 do v spomedzi ciest, ktoré celé s výnimkou vrcholu v ležia v S.

Platnosť invariantu dokážeme indukciou vzhľadom na $|S|$.

Pre $|S| = 1$ (tj. pri prvom prechode) má najlacnejšia cesta z v_0 do v_0 cenu 0 a cesta z v_0 do v celá s výnimkou vrcholu v ležiaca v množine S pozostáva z hrany (v_0, v).

Nech ďalej invariant platí pre $|S| = k$. Nech w je vrchol, ktorý vyberieme na základe podmienky v riadku (5). Najprv sporom ukážeme, že $D[w]$ je cena najlacnejšej cesty z v_0 do w.

Nech teda existuje cesta P z v_0 do w, kde $|P| < D[w]$. Podľa indukčného predpokladu je $D[w]$ cena najlacnejšej cesty z v_0 do w spomedzi takých ciest, ktoré celé okrem vrcholu w ležia v S. Preto musí na ceste P existovať vrchol (rôzny od w), ktorý nepatrí do S. Nech v je prvý takýto vrchol. Označme Q úsek cesty P od v_0 po v. S výnimkou vrcholu v ležia všetky vrcholy cesty Q v množine S. Potom ale podľa indukčného predpokladu musí platiť $D[v] \leq |Q|$. Súčasne, keďže ceny hrán sú nezáporné, platí $|Q| \leq |P| < D[w]$ a teda $D[v] < D[w]$, čo je v spore s podmienkou výberu vrcholu w v riadku (5). Preto $D[w]$ musí byť cena najlacnejšej cesty z v_0 do w. Zvyšné časti tvrdenia invariantu sú zrejmé.

V každom kroku cyklu pridáme do množiny S práve jeden vrchol a teda po konečnom počte krokov cyklus skončí. Z platnosti invariantu po skončení cyklu priamo vychádza správnosť algoritmu 8. $\square$

Poznámka: Algoritmus pre riešenie problému 2 môžeme použiť aj pre riešenie problému 1. Navyše nie je známy asymptoticky rýchlejší algoritmu pre riešenie problému 1.

Poznámka: Je známy algoritmus pre riešenie problému 2 so zložitosťou $O(|V|.|E|)$, pričom neuvažujeme obmedzenie ohodnotení hrán na nezáporné čísla. Algoritmus zistí

- či existuje v grafe cyklus zápornej ceny dosiahnuteľný z vrcholu v_0,
- ak taký cyklus neexistuje potom pre každý vrchol v vypočíta cenu najlacnejšej cesty z v_0 do v.

Poznámka: Algoritmus 8 možno upraviť tak, aby bolo možné zrekonštruovať nájdené najlacnejšie cesty. Pre každý vrchol v si budeme v $P[v]$ pamätať číslo vrcholu, ktorý mu predchádza na doteraz nájdenej najlacnejšej ceste[8]. Po skončení algoritmu teda pre ľubovoľný vrchol v najlacnejšou cestou z v_0 do v bude cesta $(v_0, \ldots, P[P[v]], P[v], v)$.

[8] Je dobré si uvedomiť, že ak najlacnejšia cesta z vrcholu v_0 do v prechádza vrcholom w, potom časť tejto cesty z v_0 do w je najlacnejšou cestou z v_0 do w.

Na začiatku je potrebné položiť pre každé v $P[v] = v_0$. Ak modifikujeme $D[v]$ v riadku (8) algoritmu 8, je potrebné modifikovať príslušným spôsobom aj pole P, tj. riadok (8) nahradíme takto:

if $D[w] + h(w,v) < D[v]$ **then begin**
  $D[v] \leftarrow D[w] + h(w,v)$
  $P[v] \leftarrow w$
end

2.2.2 Floyd–Warshallow algoritmus

V tejto časti ukážeme algoritmus, ktorý rieši problém 3 v čase $O(|V|^3)$.

Je daný orientovaný graf $G = (V,E)$ s cenami hrán z R, pričom sa v ňom nenachádza cyklus zápornej ceny. Nech je graf G reprezentovaný incidenčnou maticou $W = (w_{ij})$, pričom ak $(i,j) \in E$, potom w_{ij} je cena hrany (i,j), ak $(i,j) \notin E$ tak $w_{ij} = \infty$, ďalej pre každé i $w_{ii} = 0$.

Výstupom algoritmu bude matica $C^{(n)} = (c_{ij}^{(n)})$, kde $c_{ij}^{(n)}$ je cena najlacnejšej cesty z vrcholu i do vrcholu j.

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 $p_{ij}^{(k)}$. $p_{ij}^{(k)}$ 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 najkratšiu 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.

Cvičenie 2.8 Uvažujme ohodnotenie hrán grafu G také, že platí ak $(u, v) \in E$ potom $0 \leq h(u, v) \leq 1$. Pri takomto ohodnotení *spoľahlivosť cesty* v grafe G je súčin ohodnotení jednotlivých hrán na tejto ceste. Napíšte algoritmy, ktoré nájdu najspoľahlivejšie cesty v grafe z vrcholu v_0 do všetkých ostatných vrcholov, resp. z každého do každého vrcholu grafu G.[9]

Cvičenie 2.9 Podobne ako v predchádzajúcom cvičení nájdite algoritmy pre tzv. najširšiu cestu. *Šírka cesty* je maximum ohodnotení hrán na tejto ceste.

Cvičenie 2.10 Daných je N letísk svojimi súradnicami, ďalej je daný dolet lietadla t (po preletení vzdialenosti t musí lietadlo nutne pristáť na niektorom letisku). Ďalej sú dané dve letiská s a t. Medzi každými dvoma letiskami, ktorých vzdialenosť je nanajvýš rovná doletu lietadla, lietadlo letí po priamke. Napíšte algoritmus, ktorý nájde trasu pre lietadlo:

a) s najmenším počtom medzipristátí

b) s najmenšou celkovou vzdialenosťou

Cvičenie 2.11 Daných je N miest. Medzi týmito mestami premáva M autobusov. Autobusy premávajú vždy iba priamo z jedného mesta do niektorého iného mesta. U každého autobusu vieme: z ktorého mesta vychádza, čas odchodu, do ktorého mesta prichádza, čas príchodu. Cesta žiadneho autobusu netrvá viac ako 24 hodín.

Napíšte algoritmus, ktorý pre zadaný rozpis autobusov zistí, ako sa najrýchlejšie možno dostať zo zadaného mesta s do iného zadaného mesta t (nezabudnite na čakacie doby na spoje).

Cvičenie 2.12 Na sídlisku Číselníkovo je N križovatiek očíslovaných od 1 po N. Križovatky sú pospájané ulicami rôznej dĺžky (pod ulicou rozumieme úsek cesty neprerušený križovatkou). Popri každej ulici stoja smetiaky. Smetiari, ktorí majú depo na križovatke 1, majú k dispozícii jediné smetiarske auto, ktorým každý deň musia vyprázdniť všetky smetiaky v Číselníkove.

Nájdite algoritmus, ktorý určí najkratšiu možnú trasu smetiarskeho auta v Číselníkove, pričom auto vychádza z križovatky číslo 1, prejde všetky ulice a vráti sa späť na križovatku číslo 1.

[9] Pouvažujte nad vlastnosťami logaritmu.

2.3 Ďalšia literatúra

Referencie

[PLE83] Plesník J.: *Grafové algoritmy*, VEDA 1983

[KUČ83] Kučera L.: *Kombinatorické algoritmy*, SNTL Praha 1983

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 rekurentý vzťah

$$T(n) = 8T(n/2) + \Theta(n^2).$$

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

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

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

□

Veta 3.2 (Strassen) *Na vynásobenie dvoch matíc typu $n \times n$ stačí $O(n^{\log_2 7})$ aritmetických operácií.*

Dôkaz: Nech A a B sú dve matice typu $n \times n$, nech $n = 2^k$. Rozdeľme každú z matíc A a B na štyri podmatice typu $\frac{n}{2} \times \frac{n}{2}$. Teda

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

Podľa lemy 3.1 možno všetky podmatice C_{ij} vypočítať pomocou 7 súčinov a 18 súčtov matíc typu $\frac{n}{2} \times \frac{n}{2}$. Rekurzívnym aplikovaním tohoto algoritmu možno vypočítať súčin dvoch matíc typu $n \times n$ s použitím $T(n)$ jednoduchých aritmetických operácií, kde

$$T(n) \leq 7T(\frac{n}{2}) + 18(\frac{n}{2})^2,$$

pre $n \geq 2$. Preto $T(n) = O(7^{\log_2 n}) = O(n^{log_2 7})$.

Ak n nie je mocnina čísla 2, potom rozšírime obe matice A a B tak, aby boli typu $2^k \times 2^k$, kde $n < 2^k \leq 2n$. Celkový počet operácií postačujúci na vykonanie algoritmu popísaného vyššie na takto rozšírených maticiach bude $O((2^k)^{\log_2 7}) = O((2n)^{\log_2 7}) = O(n^{\log_2 7})$. □

Poznámka: Strassenova metóda násobenia matíc je pre malé ($n \leq 45$) alebo riedke matice nepraktická. Vieme síce túto metódu implementovať tak, že čas potrebný na násobenie dvoch matíc typu $n \times n$ je nanajvýš $cn^{\log_2 7} \approx cn^{2.81}$, ale c je značne veľká konštanta. Pre riedke matice existuje špeciálny algoritmus lepší než Strassenov.

Poznámka: Existujú asymptoticky rýchlejšie ale značne komplikovanejšie algoritmy než Strassenov. V roku 1990 mal najrýchlejší algoritmus časovú zložitosť $O(n^{2.376})$. Najlepší získaný dolný odhad zložitosti násobenia matíc je $\Omega(n^2)$.

3.1.1 Násobenie booleovských matíc

Definícia 3.3 *Nech $A = (a_{ij})$, $B = (b_{ij})$ sú booleovské matice typu $n \times n$. Booleovský súčin matíc A a B je booleovská matica $C = (c_{ij})$ typu $n \times n$, kde*

$$c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}.$$

V nasledujúcom texte opíšeme algoritmus pre booleovské násobenie matíc.

Algoritmus 10

1. *Strassenovým algoritmom (pozri vetu 3.2) vypočítame celočíselný súčin matíc A a B. Výslednú maticu označme C'*

2. Kedže $a_{ik} \wedge b_{kj} = 0 \Leftrightarrow a_{ik}b_{kj} = 0$, *tak* $c_{ij} = 0 \Leftrightarrow c'_{ij} = 0$. *Preto výslednú booleovskú maticu bude platiť*

$$c_{ij} = \begin{cases} 0 & \text{, ak c'(i,j)=0,} \\ 1 & \text{inak} \end{cases}$$

Poznámka: Strassenov algoritmus nemožno použiť priamo na výpočet booleovského súčinu matíc, keďže pre booleovské matice nie je definovaný rozdiel matíc (resp. opačná matica), ale tieto sa v Strassenovom algoritme používajú (pozri dôkaz vety 3.2).

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 dvoch 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 $M(2m) \geq 2^{2+\varepsilon} M(m)$.*

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 [AHUK6].
2. Nájdeme LUP dekompozíciu matice A v čase $O(M(n))$ (viď. 1). Potom vyriešime systém rovníc $Ly = b$ a nakoniec vyriešime systém rovníc $UPx = y$. Oba tieto systémy možno riešiť spätnou substitúciou v čase $O(n^2)$. Teda celkový čas potrebný na vyriešenie systému $Ax = b$ je $O(M(n)) + O(n^2) = O(M(n))$, lebo $M(n) \geq 2^{2+\varepsilon} M(n/2) \geq 2^{2 \log_2 n} M(1) = n^2 M(1)$.
3. Tvrdenie vyplýva z 1 a z toho, že A^{-1}=$(LUP)^{-1}$=$P^{-1}U^{-1}L^{-1}$ (matice P^{-1}, U^{-1} a L^{-1} možno vypočítať v čase $O(n^2)$).
4. Tvrdenie vyplýva z 1 a z toho, že $\det(A)$=$\det(LUP)$=$\det(L)\det(U)\det(P)$ ($\det(P) = \pm 1$, pričom znamienko možno zistiť v čase $O(n^2)$ podľa parity permutácie, $\det(L) = 1$ a $\det(U)$ je súčin prvkov na diagonále, ktorý vieme vypočítať v čase $O(n)$).

□

Dôsledok 3.8 *Tvrdenia vety 3.7 platia pre nejaký čas $M(n) \leq cn^{2.81}$.*

Dôkaz: Tvrdenie vyplýva z vety 3.2. □

Cvičenia

Cvičenie 3.6 Nájdite algoritmus pre LUP dekompozíciu s časovou zložitosťou $O(n^3)$.

Cvičenie 3.7 Môže mať singulárna matica LUP dekompozíciu?

3.3 Ďalšia literatúra

Referencie

[MÍK85] Míka S.: *Numerické metody algebry*, SNTL 1985

[AHUK6] Aho A. V., Hopcroft J. E., Ullman J. D.: *The Design and Analysis of Computer Algorithms*, kapitola 6, Addison-Wesley 1974

4 Metódy tvorby efektívnych algoritmov

Táto kapitola sa bude 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. Tento, kto navrhuje algoritmus riešiaci danú úlohu, mal by pokračovať v skúmaní problému z rôznych pohľadov, až kým si nie je istý, že získal najlepší algoritmus pre jeho potreby.

Voľba vhodnej štruktúry údajov. Spôsob organizácie dát pri výpočte algoritmu často výrazne vplýva na jeho efektívnosť. Preto voľbou vhodnej reprezentácie získavame obvykle 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 jednotlivé podštruktúry (prípadne podvýpočty) mali približne rovnakú veľkosť.

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 použijeme ten istý princíp vo väčšom rozsahu: ak nevieme presne určiť, ktoré menšie problémy riešiť, jednoducho ich vyriešime všetky a uložíme si výsledky, aby mohli byť použité pri výpočte väčších problémov.

Greedy algoritmy. Pri hľadaní otimálneho 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 skúmame problém z rôznych pohľadov a snažíme sa nájsť stále efektívnejší algoritmus a takto nájdený algoritmus nám poskytuje horný odhad zložitosti problému. Na druhej strane sa snažíme nájsť čo najlepší dolný odhad. Pokiaľ sa tieto dva odhady nestretnú, je obvykle možné zlepšiť buď dolný odhad, alebo nájsť efektívnejší algoritmus.

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 $\Omega(n \log n)$.

4.2 Voľba vhodnej štruktúry údajov

Abstraktný dátový typ je abstrakcia nad dátovými štruktúrami, kde neuvažujeme skutočné uloženie dát, ale iba to, aké operácie sa budú na štruktúre vykonávať.

Príklad: Slovník je abstraktný dátový typ, u ktorého považujeme operácie MEMBER, INSERT, DELETE.

Pri tvorbe algoritmu musíme rozhodnúť, akými dátovými štruktúrami budeme realizovať jednotlivé abstraktné dátové typy potrebné v algoritme. Pri tom musíme brať ohľad na to, aby najpoužívanejšie operácie boli realizované čo najefektívnejšie.

Príklad: Realizácia slovníka:

Implementácia	**MEMBER**	**INSERT**	**DELETE**
Pole	$O(n)$	$O(1)$	$O(n)$
Utriedené pole	$O(\log n)$	$O(n)$	$O(n)$
2-3 stromy	$O(\log n)$	$O(\log n)$	$O(\log n)$

Príklad: Prioritná fronta je abstraktný dátový typ, od ktorého požadujeme operácie INSERT, MIN, DELETE_MIN.

Implementácia	**INSERT**	**MIN**	**DELETE_MIN**
Pole	$O(1)$	$O(n)$	$O(n)$
Utriedené pole	$O(n)$	$O(1)$	$O(1)$
Halda	$O(\log n)$	$O(1)$	$O(\log n)$
2-3 stromy	$O(\log n)$	$O(\log n)$	$O(\log n)$

Cvičenia

Cvičenie 4.4 Ukážte, ako možno miernou modifikáciou 2-3 stromu dosiahnuť realizáciu prioritnej fronty, v ktorej sa dá operácia MIN vykonať v čase $O(1)$ a aby časová zložitosť ostatných operácií zostala zachovaná.

Cvičenie 4.5 Na obrázku je pohľad na sídlisko zpredu. Jednotlivé domy sa na obrázku javia ako obdĺžniky, pričom poznáme ich výšku a x-ovú súradnicu ľavého a pravého okraja. Spodnú stranu majú všetky obdĺžniky na tej istej priamke. Zostrojte algoritmus, ktorý určí postupnosť úsečiek tvoriacich "hornýöbrys sídlika (siluetu).

Cvičenie 4.6 Daná postupnosť slov nad $\{0, 1\}$. Jednotlivé slová sú na vstupe oddelené medzerami. Úlohou je zistiť, ktoré slovo sa vyskytuje v postupnosti najviac krát. Pokúste sa nájsť algoritmus s časovou zložitosťou $O(n)$, kde n je súčet dĺžok všetkých slov.

4.3 Princíp vyváženosti

Pri návrhu algoritmov sa často stretneme s prípadmi, keď sa výpočet rozdeľuje na niekoľko podúloh, alebo sa nejaká dátová štruktúra rozdeľuje na menšie podštruktúry. Efektívnosť takýchto algoritmov možno často zvýšiť tým, že sa snažíme, aby medzi jednotlivými podobjektami (či už podvýpočtami alebo podštruktúrami) bola istá vyváženosť.

Príklad: Algoritmus QUICKSORT, ktorý vyberá pivotný prvok náhodne, má v priemernom prípade časovú zložitosť $O(n \log n)$. V najhoršom prípade (keď za pivotný prvok vyberieme vždy minimum) však tento algoritmu má časovú zložitosť $O(n^2)$. Ak však za pivotný prvok volíme napríklad medián (viď. strana 4), čím zaistíme, že pole sa rozdelí na dve rovnaké časti (s rozdielom nanajvýš jeden prvok), dostávame aj v najhoršom prípade časovú zložitosť $O(n \log n)$.

Príklad: Výška binárneho prehľadávacieho stromu je v priemernom prípade $\log n$ a vyhľadávanie v takomto strome má teda časovú zložitosť $O(\log n)$. V najhoršom prípade však výška takéhoto stromu môžu byť až n a teda časová zložitosť vyhľadanie prvku v najhoršom prípade je $O(n)$. 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 9), AVL stromy, červeno-čierne (RB) stromy, stromy pre UNION/FIND-SET problém (viď. strana 14), binárne prehľadávacie stromy, algoritmus SELECT pre hľadanie k-teho najmenšieho prvku (viď. strana 4).

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

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

□

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:

$u \leftarrow (a + b)(c + d)$
$v \leftarrow ac$
$w \leftarrow bd$
$z \leftarrow v2^n + (u - v - w)2^{n/2} + w$

Čitateľ ľahko ukáže na základe predchádzajúcej úvahy, že dve n-bitové čísla možno vynásobiť v čase $T(n)$, kde $T(n) = 3T(n/2) + tn$ pre nejaké $t \geq 0$ a každé n, ktoré je mocninou dvojky. Podľa vety 4.1 teda $T(n) = O(n^{\log_2 3}) = O(n^{1.59})$.

Príklad: Metódu Rozdeľuj a panuj využívame aj pri konštrukcii týchto algoritmov: Strassenovo násobenie matíc (pozri strana 26), hľadanie k-teho najmenšieho prvku (pozri strana 4), quicksort, binárne vyhľadávanie.

Poznámka: Bez dôkazu uveďme ešte jedno tvrdenie, ktoré je o niečo všeobecnejšie, ako tvrdenie 4.1.

Veta 4.2 *Nech $a \geq 1$, $b > 1$ sú konštanty, nech f je funkcia a nech $T(n)$ je definovaná rekurentne ako nezáporná funkcia*

$$T(n) = aT(n/b) + f(n).$$

Potom $T(n)$ môže byť asymptoticky ohraničená takto:

- *ak $f(n) = O(n^{\log_b a - \varepsilon})$ pre nejaké $\varepsilon > 0$, potom $T(n) = \Theta(n^{\log_b a})$,*
- *ak $f(n) = \Theta(n^{\log_b a})$, potom $T(n) = \Theta(n^{\log_b a} \log n)$,*
- *ak $f(n) = \Omega(n^{\log_b a + \varepsilon})$ pre nejaké $\varepsilon > 0$ a ak $af(n/b) \leq cf(n)$ pre nejaké $0 < c < 1$ a pre všetky dostačne veľké n, potom $T(n) = \Theta(f(n))$.*

Cvičenia

Cvičenie 4.7 Nájdite algoritmus na nájdenie konvexného obalu množiny bodov metódou "Rozdeľuj a panuj"s časovou zložitosťou $O(n \log n)$[10].

[10]Hint: zotrieďte body podľa x-ovej súradnice a rozdeľte úlohu tak, aby ste dostali dva menšie disjunktné konvexné obaly.

Cvičenie 4.8 Riešte cvičenie 4.5 metódou "Rozdeľuj a panuj".

Cvičenie 4.9 Je daná postupnosť (kladných aj záporných) čísel $a_1 \dots a_n$. Napíšte program, ktorý nájde súvislú podpostupnosť tejto postupnosti s najväčším možným súčtom (metódou "Rozdeľuj a panuj"v čase $O(n \log n)$).

Cvičenie 4.10 Pokúste sa nájsť lineárny algoritmus pre riešenie úlohy z predchádzajúceho cvičenia.

Cvičenie 4.11 V rovine je daných n bodov. Nájdite algoritmus, ktorý nájde medzi nimi dvojicu bodov s najmenšou vzdialenosťou ($O(n \log n)$).

4.5 Dynamické programovanie

Dynamické programovanie, podobne ako metóda "Rozdeľuj a panuj", zostrojí riešenie problému pomocou riešení podproblémov. Na rozdiel od "Rozdeľuj a panuj"metóda dynamického programovania rieši problém "zdola-nahor"(bottom-up) a to tak, že postupuje od menších podproblémov k väčším. Medzivýsledky zapisujeme do tabuľky, čím možno zabezpečiť, že každý podproblém je riešený práve raz. V niektorých aplikáciách iných metód ("Rozdeľuj a panujälebo backtracking) môže totiž dochádzať k viacnásobnému riešeniu niektorých podproblémov, čo zapríčiní spravidla horšiu časovú zložitosť.

Poznámka: Dynamické programovanie nemusí viesť k efektívnemu algoritmu, ak nepotrebujeme pre výpočet celkového problému poznať riešenia všetkých podproblémov.

Príklad: Floyd-Warshallow algoritmus (pozri stranu 22), riešenie problému násobenia reťazca matíc, 0-1 knapsack problému (viď. nižšie).

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 $M_1, \dots, M_n$, 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 všetkách 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 $r_0, \dots, r_n$, kde $r_{i-1} \times r_i$ je typ matice M_i. Zavedieme pole $m[1 \dots n, 1 \dots n]$, kde $m[i, j]$ bude minimálny počet skalárnych násobení potrebných na výpočet súčinu matíc $M_i, \dots, M_j$. V pomocnom poli $s[1 \dots n, 1 \dots n]$ budeme

[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.

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
            min ← i; minh ← m[i,i] + m[i+1,j] + r_{i−1} r_i r_j
            for k ← i + 1 to j − 1 do begin
                h ← m[i,k] + m[k+1,j] + r_{i−1} r_k r_j
                if h < minh then begin
                    minh ← h
                    min ← k
                end
            end
            m[i,j] ← minh
            s[i,j] ← min
        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 $i \leq k \leq l \leq j$, tj. pre všetky podreťazce reťazca matíc $M_i M_{i+1} \ldots M_j$. Celú tabuľku teda vypočítame pomocou RP(i,j).

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
```

Lema 4.3 *Nech $T(m)$ je čas výpočtu procedúry RP(i,j) pre $m = j - i + 1$. Platí $T(m) \geq 2^{m-1}$.*

Dôkaz: Matematickou indukciou. Pre $m = 1$ platí $T(1) \geq 1$, ďalej

$$
\begin{aligned}
T(m) &= T(j-i+1) \geq \sum_{k=i}^{j-1} (T(k-i+1) + T(j-k)) = |l := k-i+1| = \\
&= \sum_{l=1}^{m-1} (T(l) + T(m-l)) = \\
&= 2 \sum_{l=1}^{m-1} T(l) \geq 2 \sum_{l=1}^{m-1} 2^{l-1} = 2(2^{m-1} - 1) \geq 2^{m-1}
\end{aligned}
$$

□

Z neefektívnej procedúry RP ($\Omega(2^n)$) možno ľahko vyrobiť efektívnu procedúru tak, že si budeme pamätať, či už boli jednotlivé podproblémy vyriešené alebo nie (aby nedochádzalo k ich viacnásobnému riešeniu).

Algoritmus 13

> **procedure** *RPM(i,j)*
> **begin**
>   **if** *procedúra RPM ešte nebola volaná s parametrami* i *a* j **then begin**
>   **if** $i = j$ **then** $m[i,j] \leftarrow 0$
>   **else** $m[i,j] \leftarrow \min_{i \le k < j}\{RPM(i,k) + RPM(k+1,j) + r_{i-1}r_k r_j\}$
>   **end**
>   **return** $m[i,j]$
> **end**

Lema 4.4 *Procedúra RPM(1,n) vypočíta hodnoty tabuľky* $m[1\dots n, 1\dots n]$ *v čase* $O(n^3)$.

Dôkaz: Nazvime *lacným* volaním procedúry RPM(i,j) také volanie, že príslušnú hodnotu už nemusíme počítať (tj. procedúra RPM už bola volaná aspoň raz s týmito parametrami). Inak je volanie procedúry *drahé.*

Pre výpočet RPM(1,n) platí:

- Pre každú dvojicu i,j ($1 \le i \le j \le n$) sa vyskytne práve jedno drahé volanie RPM(i,j), pričom ak nepočítame čas potrebný na vykonanie ďalších (rekurzívnych) volaní RPM, potrebuje takéto volanie čas $O(n)$.
- Celkový počet lacných volaní je $O(n^3)$, lacné volanie je totiž vždy vyvolané nejakým drahým volaním a každé drahé volanie vyvoláva najviac $2n-2$ volaní (lacných aj drahých). Každé lacné volanie vyžaduje čas $O(1)$.

Teda celkový čas potrebný na všetky volania je $O(n^3)$. □

Ukázali sme si dva spôsoby ako efektívne riešiť problém násobenia reťazca matíc: pomocou dynamického programovania a pomocou metódy rozdeľuj a panuj so zapamätaním medzivýsledkov. Obidva algoritmy majú zložitosť $O(n^3)$. Použitie metódy rozdeľuj a panuj zavádza do riešenia rekurziu, preto má riešenie pomocou dynamického programovania určité výhody.

4.5.2 0-1 knapsack problém

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

$$\max_{S \subseteq \{1,2,\dots,n\}} \{\sum_{i \in S} v_i | \sum_{i \in S} w_i \le w\}$$

.

Nech $V(w,j) = \max_{S \subseteq \{1,2,\dots,j\}} \{\sum_{i \in S} v_i | \sum_{i \in S} w_i \le w\}$, pre $j = 1,2,\dots,n$. 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, pričom tento výber je zrejme optimálny pre $V(w - w_{j+1}, j)$. Ak optimálny výber pre

$V(w, j+1)$ nebsahuje objekt s indexom $j+1$, potom $V(w, j+1) = V(w, j)$. Teda pre každé j 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 $V[0 \dots W, 0 \dots n]$ 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 $a_1, \dots, a_n$. Nájdite najdlhšiu rastúcu vybranú podpostupnosť tejto postupnosti (tj. čísla $1 \leq i_1 < i_2 < \dots < i_k \leq n$ také, že $a_{i_1} < a_{i_2} < \dots < ai_k$ 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 $1, \dots, n$, 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 $1, a_1, \dots, a_{l-1}, n, a_{l+1}, \dots, a_m, 1$, pričom $1 < a_1 < \dots < a_{l-2} < n$ a $n > a_{l+1} > \dots > a_m > 1$ 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 $V_1, \dots, V_n$. 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 20), Kruskalov algoritmus (pozri stranu 18)

Príklad: Daných je n objektov s váhami $w_1, w_2, \dots, w_n$ (kde w_i sú prirodzené čísla), cenami $v_1, v_2, \dots, v_n$ 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

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

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

Cvičenia

Cvičenie 4.16 Nájdite algoritmus, ktorý určí vyplatenie sumy na najmenší možný počet platidiel v sústave slovenských platidiel. Možno použiť greedy algoritmus?

Cvičenie 4.17 Je možné predchádzajúce cvičenie riešiť greedy algoritmom v každom systéme platidiel? Ak áno, dokážte, ak nie, nájdite kontrapríklad.

Cvičenie 4.18 Máme k dispozícii tlačiareň, ktorá si vie v pamätať tvary k znakov. Pomocou operácie *Download*(i, x) je možné na i-tu pozíciu v pamäti tlačiarne $(1 \leq i \leq k)$ zapísať tvar znaku x. Tlačiareň môže tlačiť len tie znaky, ktoré má uložené v pamäti, pričom na začiatku tlačenia je pamäť tlačiarne prázdna. Daný je text, ktorý je potrebné na tlačiarni vytlačiť. Nájdite spôsob, ako to spraviť použitím najmenšieho možného počtu operácií *Download*. Dokážte správnosť vášho algoritmu.

Cvičenie 4.19 Určite podmienku použiteľnosti greedy algoritmu na vyplácanie peňazí v systéme s troma druhmi platidiel.

4.7 Ďalšia literatúra

Referencie

[SAD88] Sadgewick R.: *Algorithms*, Addison-Wesley 1988

[PRE85] Preparata F. P., Shamos M. I.: *Computational Geometry (an Introduction)*, Springer Verlag 1985

[BE192] Bentley J.: *Perly programovania*, Alfa Bratislava 1992

[BE288] Bentley J.: *More Programming Pearls, Confessions of a Coder*, Addison-Wesley 1988

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 $O(2^n)$ 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 v reálnych číslach

[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.

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

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

Definícia 5.1 *Trieda $\mathcal{P}$ je množina jazykov L takých, pre ktoré existuje polynóm $p(n)$ s nezápornými koeficiantami a k-páskový deterministický Turingov stroj M s časovou zložitosťou $p(n)$, ktorý akceptuje jazyk L.*

Trieda $\mathcal{NP}$ je množina jazykov L takých, pre ktoré existuje polynóm $p(n)$ a k-páskový nedeterministický Turingov stroj M s časovou zložitosťou $p(n)$, ktorý akceptuje jazyk L.

Definícia 5.2 *Jazyk $L \subseteq \Sigma^*$ je polynomiálne transformovateľný na jazyk $L_0 \subseteq \Sigma_0^*$, ak existuje jednopáskový deterministický Turingov stroj M s polynomiálnou časovou zložitosťou $p(n)$, ktorý slovo $x \in \Sigma^*$ pretransformuje na slovo $y \in \Sigma_0^*$, pričom platí, že $x \in L$ práve vtedy $y \in L_0$.*

Definícia 5.3 *Jazyk L_0 je $\mathcal{NP}$-úplný, ak $L_0 \in \mathcal{NP}$ a každý jazyk z $\mathcal{NP}$ je polynomiálne transformovateľný na L_0.*

Veta 5.4 *Nech L je ľubovoľný $\mathcal{NP}$-úplný jazyk. $L \in \mathcal{P}$ práve vtedy, keď $\mathcal{P} = \mathcal{NP}$.*

Dôkaz: Nech $\mathcal{P} = \mathcal{NP}$. Potom zrejme L patrí do $\mathcal{P}$. Nech $L \in \mathcal{P}$ a nech $L' \subseteq \Sigma^*$ je ľubovoľný jazyk z $\mathcal{NP}$. Potom existujú polynómy $p_1(n)$, $p_2(n)$, deterministický Turingov stroj M_1 s časovou zložitosťou $p_1(n)$ a deterministický Turingov stroj M_2 s časovou zložitosťou $p_2(n)$ také, že M_1 transformuje L' na L a M_2 akceptuje L. Nech M_3 je deterministický Turingov stroj, ktorý na vstup $x \in \Sigma^*$ najprv simuluje výpočet stroja M_1 na vstupe x (výstupom stroja M_1 nech je slovo y, potom $|y| \leq p_1(|x|)$) a potom M_3 simuluje výpočet stroja M_2 na vstupe y. M_3 akceptuje L'. Čas výpočtu stroja M_3 na vstupe x je $p_1(|x|) + p_2(|y|) \leq p_1(|x|) + p_2(p_1(|x|))$. Keďže $p_1(n)$ aj $p_2(n)$ sú polynómy, aj $p_1(n) + p_2(p_1(n))$ je polynóm. Preto $L' \in \mathcal{P}$ a teda $\mathcal{P} = \mathcal{NP}$. $\square$

5.2 $\mathcal{NP}$-úplné problémy

5.2.1 Booleovské výrazy

Booleovské výrazy obsahujúce booleovské premenné, logické spojky $\neg, \vee, \wedge, \Leftarrow, \Rightarrow$ 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 $\{\neg, \vee, \wedge, \Leftarrow, \Rightarrow, (,), 0, 1\}$).*

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

Dôkaz: Čitateľ ľahko ukáže, že SAT patrí do množiny $\mathcal{NP}$ problémov. Zostáva teda už len ukázať, že každý problém z množiny $\mathcal{NP}$ je polynomiálne transformovateľný na problém 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 $q_1, q_2, \ldots, q_s$ 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,

$S_{k,t}$ $(1 \leq k \leq s,\ 0 \leq t \leq p(n))$ 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

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

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. medzi konfiguráciami Q_i, Q_{i+1} je zmena v nanajvýš jednom polísku pásky a to v tom políčku, ktoré bolo v konfigurácii Q_i snímané hlavou
4. zmena stavu, polohy hlavy a obsahu pásky medzi stavmi Q_i, Q_{i+1} sa riadi prechodovou funkciou stroja M
5. Q_0 je počiatočná konfigurácia
6. $Q_{p(n)}$ je akceptujúca konfigurácia

Teraz zostrojíme booleovské výrazy $A, \ldots, G$, ktorých splniteľnosť je postupne ekvivalentná uvedeným podmienkam $1, \ldots, 7$.

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

1. Nech A_t je ekvivalentné podmienke, že v konfigurácii Q_t je snímané práve jedno políčko pásky, t.j. $A_t = U(H_{1,t}, H_{2,t}, \ldots, H_{p(n),t})$ a súčasne $A = \prod_{t=1,\ldots,p(n)} A_t$.
2. Nech $B_{i,t}$ je ekvivalentné podmienke, že v konfigurácii Q_t je na i-tom políčku pásky práve jeden symbol, t.j. $B_{i,t} = U(C_{i,1,t}, C_{i,2,t}, \ldots, C_{i,m,t})$ a súčasne $B = \prod_{i,t=1,\ldots,p(n)} B_{i,t}$.
3. Nech C_t je ekvivalentné podmienke, že v Q_t sa stroj nachádza práve v jednom stave, t.j. $C_t = U(S_{1,t}, \ldots, S_{s,t}$ a súčasne $C = \prod_{t=1,\ldots,p(n)} C_t$.
4. Nech D_t je ekvivalentné podmienke 4 pre konfigurácie Q_{t-1}, Q_t, t.j. $D_t = \prod_{i,j}(C_{i,j,t-1} \equiv C_{i,j,t}) + H_{i,t})$ a súčasne $D = \prod_{t=1,\ldots,p(n)} D_t$.
5. Nech $E_{i,j,k,t}$ je splnené práve, keď nastane jedna z týchto možností:
   (a) i-te políčko na páske neobsahuje symbol j v čase t,
   (b) hlava nesníma políčko i v čase t,
   (c) M nie je v stave k v čase t,
   (d) nasledujúca konfigurácia Q_{t+1} vznikla z Q_t prechodom podľa prechodovej funkce δ stroja M.

   Teda:
   $$E_{i,j,k,t} = \neg C_{i,j,t} + \neg H_{i,t} + \neg S_{k,t} + \sum_l (C_{i,j_l,t+1} S_{k_l,t+1} Hi + m_l, t+1),$$
   pričom l prebieha cez všetky trojice $(q_{k_l}, X_{j_l}, m_l) \in \delta(q_k, X_j)$ $(m_l \in \{-1, 0, +1\})$. Potom teda $E = \prod_{i,j,k,t} E_{i,j,k,t}$.
6. Zostrojíme F, ktoré zodopovedá podmienke 6 (tj. Q_0 je počiatočná konfigurácia):
   $$F = S_{1,0} H_{1,0} \prod_{1 \le i \le n} C_{i,j_i,0} prod_{n+1 \le i \le p(n)} C_{i,1,0},$$
   kde vstupné slovo je $w = X_{j_1} X_{j_2} \ldots X_{j_n}$ a X_1 je znak blank.
7. Zostrojíme G, ktoré zodpovedá podmienke 7 (tj. $Q_{p(n)}$ je akceptujúca konfigurácia): $G = S_{s,p(n)}$.

Položme teraz $w_0 = ABCDEFG$; z doteraz uvedeného je zrejmé, že w_0 je splniteľné práve vtedy keď existuje akceptujúca postupnosť konfigurácií a teda keď $w \in L$.

Všimnime si ďalej, že zápis w_0 obsahuje nanajvýš $O(p^4(n))$ symbolov a (triviálne), že w_0 možno vytvoriť pre konkrétny vstup w v polynomiálnom čase. Ukázali sme teda, že ľubovoľný jazyk $L \in \mathcal{NP}$ je polynomiálne transformovateľný na SAT a teda SAT je $\mathcal{NP}$-úplný. □

Definícia 5.7 *Booleovský výraz je v konjunktívnom normálnom tvare, ak je tento výraz v tvare súčinu súčtu literálov, kde literál je booleovská premenná, alebo jej negácia. Ak každý z týchto súčtov má najviac k literálov je v k-konjunktívnom normálnom tvare.*

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 je jazyk 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 vrchlov v, w patrí do V'.*

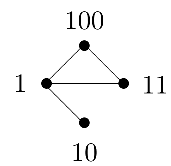
Hranové pokrytie grafu G je podmnožina $E' \subseteq E$ taká, že každý vrchol z V prislúcha niektorej hrane z E'.

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

$$\text{binárny zápis } k; \text{kód grafu } G$$



Obrázok 9: Graf, ktorého kód je 1, 10, 11, 100, $(1, 10)$, $(1, 100)$, $(1, 11)$, $(100, 11)$

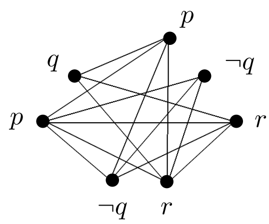
Definícia 5.12 *Nech VP je množina kódov dvojíc (k, G), kde $k \geq 0$ a G je graf majúci vrcholové pokrytie pozostávajúce z k vrcholov. Nech HP je množina kódov dvojíc (k, G), kde G je graf majúci hranové pokrytie pozostávajúce z k hrán. Nech F je množina kódov dvojíc (k, G), kde G je k-zafarbiteľný 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 transfomovateľný na K.

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

Zvolíme L_0 = CSAT. Nech $B = B_1 \wedge B_2 \wedge \ldots \wedge B_k$ je booleovský výraz v konjunktívnom normálnom tvare. Pre B zostrojíme graf G, ktorý bude mať toľko vrcholov, koľko je výskytov literálov v B. Hranou budú spojené každé dva vrcholy, ktoré odpovedajú literálom vyskytujúcim sa v rôznych podvýrazoch B_i a B_j, $i \neq j$, pričom tieto literály nie sú navzájom komplementárne (tj. α a $\neg\alpha$).



Obrázok 10: Graf priradený formule $B = (p \vee q) \wedge (\neg q \vee r) \wedge (p \vee \neg q \vee r)$

Dokážeme teraz, že B je splniteľný práve vtedy, keď G obsahuje k-kliku. Nech B má hodnotu 1 pre nejaké priradenie hodnôt 0 a 1 booleovským premenným. Potom v každom B_i vyberme práve jeden literál s hodnotou 1. Vrcholy grafu G zodpovedajúce týmto výskytom literálov sú navzájom pospájané hranami, lebo sa vyskytujú v rôznych podvýrazoch B_i a žiadne dva z nich nie sú komplementárne (inak by nemohli mať obidva hodnotu 1). Teda G obsahuje k-kliku. Podobne sa dá dokázať, že ak G obsahuje k-kliku, potom B je splniteľný.

Aby bol dôkaz tvrdenia kompletný, stačí už len ukázať, že kód výrazu B možno v polynomiálnom čase (vzhľadom na jeho dĺžku) tranformovať na kód dvojice (k, G). To je ale zrejmé z konštrukcie grafu G pre daný výraz B. □

Veta 5.14 *Jazyk HP patrí do triedy $\mathcal{P}$.*

5.2.3 Problém obchodného cestujúceho

Definícia 5.15 *Nech (V, E) je graf a nech $c : E \to N_0$ je ľubovoľná funkcia. Trojicu $G = (V, E, c)$ budeme nazývať ohodnotený graf.*

Ohodnotený graf (V, E, c) budeme kódovať podobne ako graf (V, E), ale s tým rozdielom, že v zozname hrán budú namiesto dvojíc $(i, j) \in E$ trojice $(i, j, c(i, j))$, kde $(i, j) \in E$ a čísla $c(i, j)$ budú kódované binárne. Dvojicu (k, G), kde k je nezáporné celé číslo a G je graf s ohodnotenými hranami budeme kódovať reťazcom

$$\text{binárny zápis } k; \text{kód grafu } G$$

.

Definícia 5.16 *Nech POC (problém obchodného cestujúceho — rozhodovacia verzia) je množina kódov dvojíc (k, G), kde $k \geq 0$ a G je kompletný ohodnotený graf, pričom graf G obsahuje hamiltonovskú kružnicu s cenou najviac k.*

Veta 5.17 *Jazyk POC je $\mathcal{NP}$-úplný.*

Dôkaz: Je zrejmé, že POC $\in \mathcal{NP}$. Dokážeme, že jazyk HAM je polynomiálne transformovateľný na POC. Keďže HAM je $\mathcal{NP}$-úplný, bude aj jazyk POC $\mathcal{NP}$-úplný.

Pre graf $G = (V, E)$ zostrojíme dvojicu (k, G'), kde $k = 0$ a $G' = (V, V \times V, c)$ je kompletný ohodnotený graf, pričom

$$c(i, j) = \begin{cases} 0 & \text{ak } (i, j) \in E \\ 1 & \text{inak} \end{cases}$$

Dokážeme, že G má hamiltonovskú kružnicu práve vtedy, keď G' má hamiltonovskú kružnicu s cenou najviac $k = 0$.

Nech G má hamiltonovskú kružnicu. Potom zrejme G' má hamiltonovskú kružnicu s cenou 0.

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 *Nech POC-OPT (problém obchodného cestujúceho — optimalizačná verzia) je pre daný kód kompletného ohodnoteného grafu 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).

Obrátene, 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: zväčujeme 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].

5.3 Ďalšia literatúra

Referencie

[ULM76] Aho A. V., Hopcroft J. E., Ullman J. D: *The Design and Analysis of Computer Algorithms*, Addison-Wesley 1976

[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

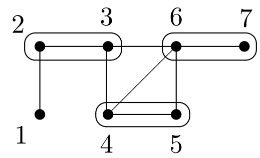
Definícia 6.1 *Problém VP-OPT (problém vrcholového pokrytia — optimalizačná verzia) je pre daný kód grafu G nájsť vrcholové pokrytie s minimálnym počtom vrcholov.*

Ľahko ukážeme, že problém VP-OPT je $\mathcal{NP}$-úplný (podobným spôsobom ako u problému POC-OPT). Preto nepoznáme žiadny deterministický polynomiálny algoritmus, riešiaci tento problém.

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



Obrázok 11: 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*

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

.

V prípade, že kompletný ohodnotený graf spĺňa trojuholníkovú nerovnosť, poznáme tiež deterministický polynomiálny aproximačný algoritmus pre POC-OPT. Tento algoritmus nájde hamiltonovskú kružnicu $\tilde{H}$, pre ktorú platí $c(\tilde{H}) \leq 2c(H^*)$, kde H^* je hamiltonovská kružnica s minimálnou cenou a $c(H)$ je cena hamiltonovskej kružnice H.

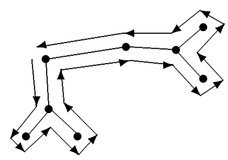
Algoritmus 15

1. *Nájdi najlacnejšiu kostru K grafu G*
2. *Algoritmom pre prehľadávanie do hĺbky prehľadávaj kostru K a vždy, keď je navštívený vrchol, ktorý predtým ešte nebol navštívený, zaraď ho do tvoriacej sa hamiltonovskej kružnice $\tilde{H}$.*

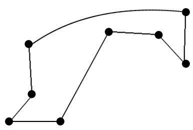
Veta 6.4 *Algoritmus 15 nájde v polynomiálnom čase hamiltonovskú kružnicu $\tilde{H}$, pre ktorú platí $c(\tilde{H}) \leq 2c(H^*)$, kde H^* je hamiltonovská kružnica s minimálnou cenou.*

Dôkaz: Keďže K je najlacnejšia kostra grafu G, musí platiť $c(K) \leq c(H^*)$, lebo inak by sme po odstránení ľubovoľnej hrany z H^* dostali lacnejšiu kostru než K.

Cena cesty pri prehľadávaní kostry K je $p(K) = 2c(K)$, lebo po každej hrane kostry K prejde algoritmus práve dvakrát.



Obrázok 12: Cesta pri prehľadávaní K



Obrázok 13: Nájdená hamiltonovská kružnica

Z trojuholníkovej nerovnosti vyplýva, že $c(\tilde{H}) \leq p(k) = 2c(K) \leq 2c(H^*)$. □

Pre grafy nespĺňajúce trojuholníkovú nerovnosť predchádzajúce tvrdenie neplatí.

Veta 6.5 *Ak by pre nejaké $\alpha > 1$ existoval deterministický polynomiálny algoritmus, ktorý by pre každý kompletný ohodnotený graf G našiel hamiltonovskú kružnice $\tilde{H}$ takú, že $c(\tilde{H}) \leq \alpha c(H^*)$, kde H^* je najlacnejšia hamiltonovská kružnica grafu G, potom by existoval deterministický polynomiálny algoritmus riešiaci POC-OPT.*

Cvičenia

Cvičenie 6.1 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) $K \leq \lceil \frac{3}{2} K^* \rceil$.