Tvorba efektívnych algoritmov

RNDr. Pavol Ďuriš, CSc.
Katedra informatiky MFF UK
e-mail: duris@fmph.uniba.sk

December 1997

Abstrakt

Tento text vznikol ako materiál ku prednáške "Tvorba efektívnych algoritmov" pre 3. ročník štúdia informatiky na Matematicko-fyzikálnej fakulte Univerzity Komenského. Táto prednáška voľne nadväzuje na prednášku "Algoritmy a štruktúry údajov" (2. ročník), a preto sa aj tento text často odkazuje na znalosti z tejto prednášky. Nemá slúžiť ani ako náhrada za skriptá, ani ako náhrada za prednášku. Predstavuje len mierne prepracovanú verziu poznámok k tejto prednáške.

Jednotlivé kapitoly obvykle nepodávajú ucelený pohľad na problematiku. Ťažiskom celého textu je kapitola 4, ktorá podáva stručný prehľad jednotlivých metód tvorby efektívnych algoritmov. Vhodné príklady čitateľ nájde m.i. v kapitolách 1 až 3. Kapitoly 1 až 3 súčasne tvoria vhodné rozšírenie poznatkov oproti prednáške "Algoritmy a štruktúry údajov". Kapitola 5 podáva úvod do $\mathcal{NP}$-úplných problémov (je čiastočne doplnená o materiály z prednášky "Teória zložitosti" z bloku "Matematické metódy"). Na záver, v kapitole 6, sa čitateľ môže oboznámiť so stručným úvodom do tzv. aproximatívnych algoritmov.

Autori textu nezodpovedajú za prípadné chyby, ktoré sa v ňom nachádzajú.

1 Vyhľadávanie, triedenie a súvisiace problémy

V tejto kapitole sa budeme venovať triedeniu a vyhľadávaniu. Oproti prednáške "Algoritmy a štruktúry údajov" uvedieme niektoré ďalšie algoritmy (hľadanie k-teho najmenšieho prvku) a zavedieme niektoré nové dátové štruktúry (2-3 stromy, štruktúry pre UNION/FIND-SET problém). Takisto si 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 s časovou zložitosťou $\Theta(k.n)$, a teda vo všeobecnosti až $\Theta(n^2)$. 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 existuje algoritmus s časovou zložitosťou porovnateľnou s hľadaním minimálneho prvku (doteraz uvedené výsledky sú totiž asymptoticky 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 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: $|S_1| \leq 3n/4$, $|S_3| \leq 3n/4$.*

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 postupnosti, ktorá by vznikla zlúčením oboch týchto polí (časová zložitosť $O(\log N)$).

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 potrubie vedúce od východu na západ a ku každej vrtnej veži postaviť prípojku. Nájdite čo najefektívnejší algoritmus, ktorý určí, na akú y-ovú súradnicu je potrebné potrubie postaviť, aby súčet dĺžok prípojok k vežiam bol minimálny (časová zložitosť $O(n)$).

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 $d(m) = \min\{D_T \mid T \text{ je striktne binárny strom s } m \text{ listami}\}$

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

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 odstráni 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, t.j. algoritmus sa nemôže "pozrieť" na j-tu operáciu v σ skôr, ako vykoná operácie $1, 2, \ldots, j-1$.

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.

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.

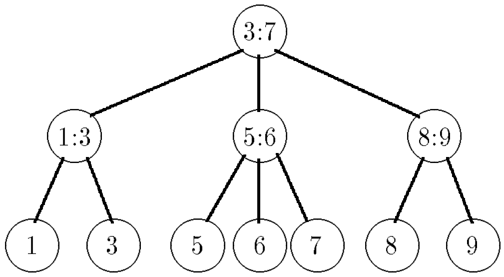
Algoritmus 3

> **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.

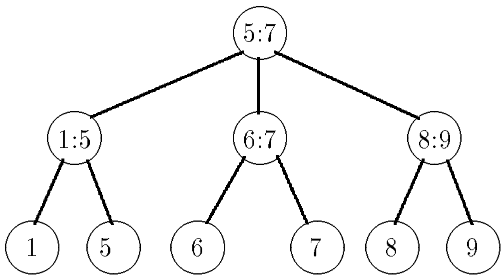


obr. 3: 2-3 strom T reprezentujúci množinu $S = \{1, 3, 5, 6, 7, 8, 9\}$

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 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.



obr. 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, ktorý po ukončení operácie bude ukazovať na koreň výsledného 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 vykonávania 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 $(u, v) \in E$:

Dôkaz: Podľa lemy 1.19 má každý strom vytvorený počas vykonávania postupnosti σ výšku najvýš $\log_2 m$. Preto je možné vykonať každú operáciu FIND-SET v σ v čase $O(\log m)$. 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 j (pre procedúru FIND-SET(j)).

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 $p[j] \neq j$ **then** $p[j] \leftarrow$*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), ale neskracujeme cesty.

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.

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

[6] Hint: Pomocou druhého poľa sa pokúste rozlíšiť neinicializovaný prvok poľa a inicializovaný.

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 (najlacnejšia kostra, najlacnejšie cesty).

2.1 Najlacnejšia kostra grafu

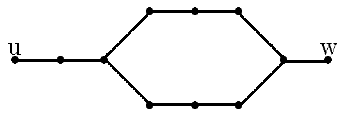
Definícia 2.1 *Nech $G = (V, E)$ je neorientovaný súvislý graf s ohodnotenými hranami, (tj. pre G je daná cenová funkcia $h : E \to R$; R je množina reálnych čísel).*

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.



obr. 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. 8). Táto kružnica musí obsahovať nejakú hranu (u', w') takú, že $u' \in V_i$ a $w' \notin V_i$. Podľa predpokladu $h(u, w) \leq h(u', w')$, 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, ktorá v S viedla cez hranu (u', w'),

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 $(V_i \cup V_j, T_i \cup T_j \cup \{(u, w)\})$. 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 práve $|V| - 1$ operácií UNION, ktorých vykonanie vyžaduje čas najviac $O(|V| + |E| \log |V|)$ (pozri vetu 1.20). 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].

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?

[7] Všimnite si nápadnú podobnosť s algoritmom 8

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 $w \in V \setminus S$ 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.

Teda tvrdenie 1 zostáva v platnosti aj po pridaní vrcholu w do S. Z riadkov (7) a (8) vyplýva, že aj tvrdenie 2 zostane v platnosti aj po pridaní w do S.

V každom kroku cyklu pridáme do množiny S práve jeden vrchol a teda po konečnom počte krokov cyklus skončí. Správnosť algoritmu 8 vychádza priamo z platnosti invariantu po skončení cyklu. □

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 nájsť 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 bude teda najlacnejšou cestou z v_0 do v cesta $(v_0, \ldots, P[P[v]], P[v], v)$.

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 uvedieme 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 $w_{ij} = \infty$ a ďalej $w_{ii} = 0$ pre každé i.

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

[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.

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.

2.3 Ďalšia literatúra

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

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

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

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

[9] Pouvažujte nad vlastnosťami logaritmu.

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/rozdielov 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

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

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 doplňme obe matice A a B nulami 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 $C' = (c'_{ij})$*

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

$$c_{ij} = \begin{cases} 0 & \text{, ak } c'_{ij} = 0, \\ 1 & \text{inak} \end{cases}$$

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).

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 $O(n)$).

$\square$

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

Dôkaz: Z dôkazu vety 3.2 vyplýva že čas potrebný na výpočet súčinu dvoch matíc typu $n \times n$ (označme ho $M(n)$) Strassenovou metódou nie je menší, než čas potrebný na výpočet 7 súčinov (nejakých) dvoch matíc typu $\frac{n}{2} \times \frac{n}{2}$. Teda $M(n) \geq 7M(\frac{n}{2}) = 2^{2+\varepsilon} M(\frac{n}{2})$ pre $\varepsilon = \log_2 7 - 2 > 0$. $\square$

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

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

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

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

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 spredu. 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 "horný" obrys sídliska (siluetu). Použite modifikáciu prioritnej fronty, ktorá umožňuje efektívnu realizáciu operácie DELETE.

Cvičenie 4.6 Daná postupnosť slov z $\{0,1\}^+$. 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 2), 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 $O(n)$.

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

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 (pozor, $a+b$ a $c+d$ môžu byť $(\frac{n}{2}+1)$-bitové čísla). 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 22), hľadanie k-teho najmenšieho prvku (pozri strana 2), 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 dostatočne 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].

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

Cvičenie 4.9 Je daná postupnosť (kladných aj záporných) čísel $a_1 \ldots a_n$. 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 (časová zložitosť $O(n \log n)$).

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 panuj" alebo 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 19), riešenie problému násobenia reťazca matíc, 0-1 knapsack problému (viď. nižšie).

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

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 $T(1) \geq 1$. Nech platí $T(l) \geq 2^{l-1}$ pre $l < m$.

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

□

Z neefektívnej procedúry RP s časovou zložitosťou $\Omega(2^n)$ možno ľahko vyrobiť efektívnu procedúru RPM 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 $m[i,j] \leftarrow \min_{i \leq k < j}\{RPM(i,k) + RPM(k+1,j) + r_{i-1}r_kr_j\}$
  end
  return $m[i,j]$
end

Lema 4.4 *Procedúra RPM(1,n) vypočíta hodnoty tabuľky $m[1\ldots n, 1\ldots n]$ 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 ($1 \leq i \leq j \leq n$) 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 $w_1, w_2, \ldots, w_n$ (kde w_i sú prirodzené čísla), cenami $v_1, v_2, \ldots, v_n$ a ďalej je dané prirodzené číslo W. Úlohou je vybrať niektoré z objektov tak, aby celková cena vybraných objektov bola najväčšia a zároveň aby ich celková váha neprekročila hodnotu W, tj. nájsť číslo

$$\max_{S \subseteq \{1,2,\ldots,n\}}\{\sum_{i \in S} v_i | \sum_{i \in S} w_i \leq W\}.$$

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 $\mathit{Download}(i, x)$ 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

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

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

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

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

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

[SED88] Sedgewick R.: *Algorithms*, Addison-Wesley 1988

Definícia 5.1 *Trieda $\mathcal{P}$ je množina jazykov L takých, pre ktoré existuje polynóm $p(n)$ 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 vstupe $x \in \Sigma^*$ najprv simuluje výpočet stroja M_1 na vstupe x (výstupom stroja M_1 nech je slovo y, zrejme $|y| \leq p_1(|x|)$) a potom M_3 simuluje výpočet stroja M_2 na vstupe y. Teda 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 $\mathcal{P} = \mathcal{NP}$. □

Je dôležité si uvedomiť, že Turingove stroje sú schopné (napriek svojej jednoduchosti a nedokonalosti) riešiť v polynomiálnom čase práve všetky tie problémy, ktoré sú riešiteľné v polynomiálnom čase bežnými počítačmi, ba dokonca dokonalejšími výpočtovými modelmi – tzv. idealizovanými reálnymi počítačmi.

Idealizovaný reálny počítač R má neohraničenú pamäť, má bežné inštrukcie (napr. ako PC) a na vstupe dĺžky n je dĺžka pamäťového slova (aj registrov) $p(n)$, kde $p(n)$ je nejaký polynóm.

Pri takýchto počítačoch však treba dávať pozor na pretečenie: počítač R nemôže na vstupe dĺžky n vykonať program

$x \leftarrow 2$
for $i \leftarrow 1$ **to** n **do** $x \leftarrow x^2$

lebo $x = 2^{2^n}$ po vykonaní programu a dĺžka zápisu takéhoto x je $2^n > p(n)$ (pre dosť veľké n).

Dá sa dokázať, že ak R akceptuje jazyk L v polynomiálnom čase, potom $L \in P$. Idea dôkazu spočíva v tom, že Turingov stroj môže simulovať vykonanie inštrukcií počítača R typu "zápis do/čítanie z pamäte" v polynomiálnom čase, lebo R použije počas výpočtu najviac polynomiálne veľa rôznych adries a Turingov stroj si môže na jednej zo svojich pások vytvárať zoznam týchto adries a hodnôt uložených na nich. Je zrejmé, že vykonanie ostatných inštrukcií počítača R môže Turingov stroj simulovať tiež v polynomiálnom čase.

Ľahko sa dá dokázať, že platí aj obrátené tvrdenie.

Podobne možno definovať aj nedeterministický idealizovaný počítač NR, ktorý je schopný vykonať inštrukciu goto$(L_1, L_2, \ldots, L_k)$. Pre NR platia podobné tvrdenia ako pre R. Teda triedy $\mathcal{P}$ a $\mathcal{NP}$ môžeme tiež definovať pomocou počítačov R a NR.

Cvičenia

Cvičenie 5.1 Majme dvojice pozícia; ťah rozumne kódované pomocou 0 a 1, kde pozícia je šachová pozícia a ťah je šachový ťah (pozícia môže byť aj nezmyselná, ťah nemusí súvisieť s danou pozíciou. Uvažujme jazyk SACH ako takú podmnožinu dvojíc pozícia; ťah, že pozícia na šachovnici je zmysluplná a ťah z danej pozície je súčasťou víťaznej stratégie ťahajúceho hráča. Platí SACH $\in \mathcal{NP}$?

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

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

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 nech $A = \bigwedge_{t=1,\ldots,p(n)} A_t$.
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 nech $B = \bigwedge_{i,t=1,\ldots,p(n)} B_{i,t}$.
3. Nech C_t je ekvivalentné podmienke, že v Q_t sa stroj nachádza práve v jednom stave, t.j. $C_t = U(S_{0,t}, \ldots, S_{s,t})$ a nech $C = \bigwedge_{t=1,\ldots,p(n)} C_t$.
4. Nech D_t je ekvivalentné podmienke 4 pre prechod z konfigurácie Q_{t-1} do konfigurácie Q_t, t.j. nech $D_t = \bigwedge_{i,j} ((C_{i,j,t-1} \Leftrightarrow C_{i,j,t}) \vee H_{i,t})$ a nech $D = \bigwedge_{t=1,\ldots,p(n)} D_t$.
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 funkcie δ stroja M.

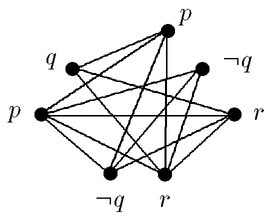
   Teda:
   $$E_{i,j,k,t} = \neg C_{i,j,t} \vee \neg H_{i,t} \vee \neg S_{k,t} \vee \bigvee_l (C_{i,j_l,t+1} \wedge S_{k_l,t+1} \wedge H_{i+m_l,t+1}),$$
   pričom l prebieha cez všetky trojice $(q_{k_l}, X_{j_l}, m_l) \in \delta(q_k, X_j)$ $(m_l \in \{-1, 0, +1\})$. Potom teda $E = \bigwedge_{i,j,k,t} E_{i,j,k,t}$.
6. Zostrojíme F, ktoré zodpovedá podmienke 6 (tj. Q_0 je počiatočná konfigurácia):
   $$F = S_{0,0} \wedge H_{1,0} \wedge \bigwedge_{1 \le i \le n} C_{i,j_i,0} \wedge \bigwedge_{n+1 \le i \le p(n)} C_{i,1,0},$$
   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 $w_0 = A \wedge B \wedge C \wedge D \wedge E \wedge F \wedge G$; 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.*

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 $\neg\alpha$).



obr. 11: 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) transformovať 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 Optimalizačné versus rozhodovacie problémy

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.

Veta 5.21 *Nech A je ľubovoľný $\mathcal{NP}$-optimalizačný problém s cieľom min, reláciou R a hodnotovou funkciou m, ktorého rozhodovací problém $L = \{x; zápis\ k\ |x \in \Sigma^* \wedge k \in N \wedge (\exists y)((x,y) \in R \wedge m(x,y) \leq k)\}$ je $\mathcal{NP}$-úplný. Nech L možno akceptovať deterministicky v čase $t(n)$. Potom konštrukčný problém pre A možno riešiť deterministicky v čase $r(n)t(s(n))$ pre vhodné polynómy r a s.*

Poznámka: Nie je je známe, či táto veta platí aj bez predpokladu, že L je $\mathcal{NP}$-úplný.

Dôkaz: V prvom kroku pre dané x nájdeme (ak existuje) hodnotu $k = \min_{y \in \Sigma^*}\{m(x,y)|(x,y) \in R\}$. Nech q je polynóm ohraničujúci čas výpočtu funkcie m. Po $q(|x| + |y|)$ krokoch výpočtu môže mať vypočítaná hodnota $m(x,y)$ v tvare binárneho zápisu najviac $2^{q(|x|+|y|)}$ číslic, t.j. $m(x,y) \leq 2^{q(|x|+|y|)} \leq 2^{q'(|x|)}$ pre vhodné q' (lebo $|y| \leq p(|x|)$). Binárnym prehľadávaním intervalu $\left\langle 0, 2^{q'(|x|)} \right\rangle$ možno nájsť číslo k použitím algoritmu na rozpoznávanie L (zisťujeme, či reťazec (x; zápis k) patrí do L).

Pre dané x a k zostrojíme niektoré optimálne riešenie y, pre ktoré platí $(x,y) \in R \wedge m(x,y) = k$. Nech $L' = \{x; \text{zápis k}; z|x \in \Sigma^* \wedge k \in N \wedge \text{ pre } x \text{ existuje } y \text{ také, že } (x,y) \in R \wedge m(x,y) \leq k \wedge z \text{ je prefix reťazca } y\}$. Jazyk L' patrí do triedy $\mathcal{NP}$(nedeterministicky si pre x a k zostrojíme y a deterministicky overíme, že z je prefix y, $(x,y) \in R$, $m(x,y) \leq k$. Keďže L je $\mathcal{NP}$-úplný, musí byť L' polynomiálne transformovateľný na L, t.j. každý reťazec (x; zápis k; z) je polynomiálne transformovateľné na nejaké w, pričom platí (x; zápis k; z) $\in L' \Leftrightarrow w \in L$.

Hľadaný reťazec y zostrojíme postupným predlžovaním už nájdeného prefixu. Ak $u0 =$ (x; zápis k; $z0$) $\in L'$ alebo $u1 =$ (x; zápis k; $z1$) $\in L'$ (t.j. ak $u0' \in L$ resp. $u1' \in L$, čo sa dá zistiť pomocou algoritmu pre rozpoznávanie L), tak $z0$, resp. $z1$ je dlhší nájdený prefix L. $\square$

Podobné dôkazy majú aj nasledujúce vety:

Veta 5.22 *Nech A je ľubovoľný $\mathcal{NP}$-optimalizačný problém, ktorého rozhodovací problém L možno akceptovať deterministicky v čase $t(n)$ (L nemusí byť $\mathcal{NP}$-úplný). Potom hodnotový problém pre A možno riešiť deterministicky v čase $r(n)t(s(n))$ pre vhodné polynómy r a s.*

Veta 5.23 *Nech A je ľubovoľný $\mathcal{NP}$-optimalizačný problém. Nech L_0 je ľubovoľný $\mathcal{NP}$-úplný problém, ktorý možno deterministicky akceptovať v čase $t(n)$. Potom konštrukčný problém pre A možno riešiť deterministicky v čase $r(n)t(s(n))$ pre vhodné polynómy r a s.*

Dôsledok 5.24 *Ak $\mathcal{P} = \mathcal{NP}$, potom konštrukčný problém každého $\mathcal{NP}$-optimalizačného problému možno riešiť deterministicky v polynomiálnom čase.*

5.3 Ďalšia literatúra

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

[PAP94] Papadimitriou C. H.: Computational Complexity, Addison-Wesley 1994

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.

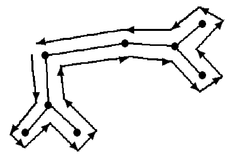
Algoritmus 15

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 $\tilde{H}$.*

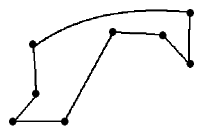
Veta 6.4 *Nech G je kompletný ohodnotený graf spĺňajúci trojuholníkovú nerovnosť. Algoritmus 15 nájde v polynomiálnom čase hamiltonovskú kružnicu $\tilde{H}$ grafu G, pre ktorú platí $c(\tilde{H}) \leq 2c(H^*)$, kde H^* je hamiltonovská kružnica s minimálnou cenou.*

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.

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



obr. 13: Cesta pri prehľadávaní K



obr. 14: Nájdená hamiltonovská kružnica

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

Pre grafy nespĺňajúce trojuholníkovú nerovnosť platí nasledujúce tvrdenie.

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, (t.j. potom by platilo $\mathcal{P} = \mathcal{NP}$).*

Definícia 6.6 *Nech A je ľubovoľný $\mathcal{NP}$-optimalizačný problém s cieľom min (max), reláciou R a hodnotovou funkciou m. Nech $m^*(x) = min\{m(x,y)|(x,y) \in R\}$. Problém A je α aproximovateľný, $\alpha \geq 1$, ak existuje deterministický polynomiálny algoritmus M, ktorý pretransformuje vstup x na výstup y, pričom $(x,y) \in R$ a platí $\alpha m^*(x) \geq m(x, M(x))$ pre skoro všetky (t.j. až na konečný počet) x (obdobne sa dá zadefinovať aproximovateľnosť pre cieľ max).*

Problém A je dobre aproximovateľný, ak je α aproximovateľný pre každé $\alpha > 1$.

Problém A je neaproximovateľný, ak nie je α aproximovateľný pre žiadne $\alpha > 1$.

Poznámka: Ak $\mathcal{P} = \mathcal{NP}$, definícia je bezpredmetná (pozri dôsledok 5.24), ale z praktického hľadiska má význam aproximovať polynomiálny algoritmus napr. algoritmom s nižším stupňom polynómu.

Veta 6.7 *Ak $\mathcal{P} \neq \mathcal{NP}$, potom existujú $\mathcal{NP}$-optimalizačné problémy A, B a C také, že*

- *A je dobre aproximovateľný, ale jeho rozhodovací problém nepatrí do $\mathcal{P}$*
- *B je α aproximovateľný pre nejaké $\alpha > 1$, ale nie je dobre aproximovateľný*
- *C je neaproximovateľný*

Príklad: Aproximovateľnosť niektorých $\mathcal{NP}$-optimalizačných problémov:

RNDr. Pavol Ďuriš, CSc.
Tvorba efektívnych algoritmov
(materiály k prednáške)

Spracovali Bronislava Brejová, Martin Gažák a Tomáš Vinař
Sadzba programom LaTeX

Verzia 2.0
Neprešlo jazykovou úpravou