Homework 1

Jakub Senko, Štefan Uherčík

19. marca 2014

Príklad 1

Nech $X = [x_1, x_2, x_3, \ldots, x_n]$ je pole cisel dlzky $n, n > 0$ a plati ze $\forall x, y \in X : x \neq y$. Kazdemu $x_i \in X$ je priradene cislo w_i, pre ktore plati:

$$\begin{aligned} w_i &> 0 \\ \sum_{i=0}^{n} w_i &= 1 \end{aligned} \tag{0.1}$$

Optimalny prvok postupnosti (OPP) je cislo x_k pre ktore plati:

$$\begin{aligned} \sum_{x_i < x_k} w_i &< \frac{1}{2} \\ \sum_{x_i > x_k} w_i &\leq \frac{1}{2} \end{aligned} \tag{0.2}$$

Problémom je návrh algoritmu, ktorý nájde optimálny prvok s časovou zložitosťou $\Theta(n)$, a poskytnutie dôkazu jeho korektnosti a zložitosti.

Navrhované riešenie je modifikovaný algoritmus *Quick Select* (QS) ktorý rieši problém nájdenia k-teho najmenšieho prvku v poli čísel. Tento algoritmus má obecne zložitosť $\mathcal{O}(n^2)$, avšak pomocou metódy *Median of Medians* (MoM) je možné nájsť dostatočne dobrý pivot na to, aby mal algoritmus vždy lineárnu zložitost. QS a MoM sú popísané v nasledujúcom texte iba neformálne, pretože sú to známe algoritmy a ich presný popis je dostupný v materiáloch predmetu a v článku od BLUM et al. [1] Zadaná úloha

je vyriešená redukciou problému nájdenia OPP na problém riešený algoritmom QS + MoM, s využitím už dokázaných znalostí o ich zložitosti. Táto redukcia je vykonatelná v linárnom čase. Výsledná zlozitost je teda $\mathcal{O}(n)$.

Quick Select

Vráti k-ty najmenší prvok poľa v časti ohraničenej indexami $left$ a $right$ (vrátane). Predpokladáme že pole je neprázdne a indexy platné.

```
1: function QUICK_SELECT(list, k, left, right)
2:     if left = right then
3:         return list[left]
4:     end if
5:     pivotIndex ← RANDOM_BETWEEN(left, right)
6:     pivotIndex ← PARTITION(list, left, right, pivotIndex)
7:     if k = pivotIndex then
8:         return list[k]
9:     else if k < pivotIndex then
10:        return QUICK_SELECT(list, k, left, pivotIndex − 1)
11:    else
12:        return QUICK_SELECT(list, k, pivotIndex + 1, right)
13:    end if
14: end function
```

QS funguje podobne ako *Quick Sort*. V každom volaní sa zvolí náhodný pivot v ohraničenej časti pomocou funkcie RANDOM_BETWEEN, a následne funkcia PARTITION v lineárnom čase rozdelí podzoznam do dvoch častí. V spodnej sú prvky menšie ako pivot a v druhej väčšie alebo rovné. Analogicky ako pri triedení funkcia patrí do $\mathcal{O}(n^2)$.

Median Of Medians

Pomocou tejto funkcie je možné deterministicky vybrať dobrý pivot tak, aby po nahradení RANDOM_BETWEEN touto procedúrou mal výsledný algoritmus QS + MoM vždy lineárnu zložitosť.

```
1: function MEDIAN_OF_MEDIANS(list, left, right)
2:     groups ← SPLIT_INTO_GROUPS_OF(list, left, right, 5)
3:     medians ← NEW_LIST
4:     for all group ∈ groups do
5:         median ← MEDIAN_OF_5(group)
6:         ADD(medians, median)
7:     end for
```

```
    size ← SIZE(medians)
    return QUICK_SELECT(medians, n/2, 1, size)
end function
```

Rozdelenie do skupín po 5 je $\mathcal{O}(n)$ ale nájdenie mediánu z konštantného počtu prvkov je $\mathcal{O}(1)$.

Zložitosť QS + MoM

Nahradenie funkcie RANDOM_BETWEEN spôsobi problém pretože MOM vracia hodnotu a QS očakáva index, ale index môžeme získať lineárnym vyhľadanim prvku v poli, čo nezmení výslednú triedu zložitosti. Takto upravený algoritmus bude obsahovať dve rekurzívne volania. Jedno vo funkcii MoM (zložitosť je $\frac{n}{5}$ pretože sme získali pole mediánov o veľkosti $\frac{n}{5}$) a druhé priamo v QS (riadok 10 alebo 12). Druhé volanie má zložitosť $\frac{7}{10}n$ čo zodpovedá hornému odhadu počtu prvkov ktoré su väčšie alebo naopak menšie ako medián mediánov, teda pivot (viz slidy). Ostatné operácie sú konštantné alebo lineárne. Nasledujúca rekurentná rovnica vyjadruje celkovú zložitosť algoritmu (kde c je konštanta):

$$\begin{aligned} T(n) &= T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + c.n \\ T(n) &= 10.c.n \in \Theta(n) \end{aligned} \tag{0.3}$$

Transformácia na algoritmus FIND_OPTIMAL_ELEMENT

Redukcia spočíva v doplňujúcej operácii, ktorá pre obidve partície zoznamu rozdeleného podľa pivota spočíta súčet ohodnotení pre všetky prvky. Tieto operácie su znovu vykonateľné v lineárnom čase. Výsledné súčty určia na ktorej partícii sa algoritmus rekurzívne zavolá, prípadne ak platí podmienka pre OPP, výpočet skončí. Môžeme odstrániť parameter k ktorý je po tejto úprave nepotrebný (vždy existuje unikátny OPP). Výsledný algoritmus (FOE) je nasledovný:

```
1:  function FIND_OPTIMAL_ELEMENT(list, left, right)
2:      if left = right then
3:          return list[left]
4:      end if
5:      pivotIndex ← MEDIAN_OF_MEDIANS_INDEX(list, left, right)
6:      pivotIndex ← PARTITION(list, left, right, pivotIndex)
7:      sumBottom ← SUM_W(list, 0, pivotIndex − 1)
8:      sumTop ← SUM_W(list, pivotIndex + 1, length(list))
9:      if sumBottom ≥ 1/2 then
10:         return FIND_OPTIMAL_ELEMENT(list, left, pivotIndex − 1)
11:     else
```

```
12:         return FIND_OPTIMAL_ELEMENT(list, pivotIndex + 1, right)
13:     end if
14: end function
15:
16: function FIND_OPTIMAL_ELEMENT_START(list)
17:     return FIND_OPTIMAL_ELEMENT(list, 1, lenght(list))
18: end function
```

Korektnosť FOE

Vstupná podmienka $\phi(\langle X \rangle)$ je určená vzťahom (0.1).
Výstupná podmienka $\psi(\langle X \rangle, x_k)$ požaduje, že výstupom algoritmu je optimalny prvok postupnosti x_k podľa (0.2).
Využijeme dôkaz matematickou indukciou. Základom indukcie je nasledujúca myšlienka: Počas priebehu algoritmu (na začiatku) je vstupné pole $X = [x_0, x_1, x_2, \ldots, x_n], n > 0$ rozdelené na tri sekcie (platí že $left < right$):

$$\begin{aligned} X_{left} &= [x_1, x_2, x_3, \ldots, x_{left-1}] \\ X_{mid} &= [x_{left}, \ldots, x_{right}] \\ X_{right} &= [x_{right+1}, \ldots, x_n] \end{aligned} \tag{0.4}$$

Tvrdím, že pre každé volanie funkcie FOE platí, že:

$$\begin{aligned} \forall x_i \in X_{left}, \forall x_j \in X_{mid}, x_i < x_j \wedge \sum w_i < \frac{1}{2} \\ \forall x_j \in X_{mid}, \forall x_k \in X_{right}, x_j < x_k \wedge \sum w_k \leq \frac{1}{2} \end{aligned} \tag{0.5}$$

Pokiaľ dokážeme, že veľkosť X_{mid} v každom rekurzívnom volaní funkcie klesne, dostaneme sa postupne k strednej časti ktorá bude obsahovať práve jeden prvok. Pretože preň budú platiť vyššie uvedené podmienky, bude sa jednať o hľadaný OPP.
Zaklad indukcie:
Pri zavolaní funkcie FIND_OPTIMAL_ELEMENT_START (iniciálne volanie FOE) je rozdelenie na tri sekcie nasledovné:

$$\begin{aligned} X_{left} &= [] \\ X_{mid} &= [x_1, x_2, \ldots, x_{n-1}, x_n] \\ X_{right} &= [] \end{aligned} \tag{0.6}$$

Triviálne tvrdenie platí.
Indukčný krok:
Predpokladáme že pre k-te rekurzívne volanie tvrdenie platí na začiatku funkcie. Analyzujme prebeh funkcie aby sme ukázali že bude platiť aj pri nasledujúcom rekurzívnom volaní $k+1$ a zároveň $|X_{mid_k}| > |X_{mid_{k+1}}|$. V prvom kroku zvolíme pivot a získame jeho index. Funkcia vráti hodnotu, pre ktorú platí: $left \leq pivotIndex \leq right$. Poznamenávam že vlastná implementacia tejto funkcie nie je podstatná pre korektnosť algortimu

ale len pre jeho efektivitu. V druhom kroku sa volá funkcia PARTITION, ktora rozdelí X_{mid_k} na dve partície:

$$\begin{aligned} X_{lower} &= [x_{left}, \ldots, x_{pivotIndex-1}] \\ X_{upper} &= [x_{pivotIndex+1}, \ldots, x_{right}] \end{aligned} \tag{0.7}$$

Pre ktoré platí:

$$\forall x \in X_{lower}, x < x_{pivotIndex} \forall x \in X_{lower}, x \geq x_{pivotIndex} \tag{0.8}$$

Dalšie dve operácie spočítajú sumu ohodnotení všetkých prvkov v zozname naľavo a potom napravo od pivota. V tomto momente sú už k dispozícii všetky informácie a nasleduje podmienené vetvenie, ktoré rekurzívne zavolá FOE. Ak platí že $sumBottom \geq \frac{1}{2}$ potom bude pri dalšom rekurzívnom volaní nové rozdelenie nasledujuce ($\sqcup$ je zjednotenie zoznamov):

$$\begin{aligned} X_{left_{k+1}} &= X_{left_k} \\ X_{mid_{k+1}} &= X_{lower_k} \\ X_{right_{k+1}} &= [x_{pivotIndex}] \sqcup X_{upper_k} \sqcup X_{right_k} \end{aligned} \tag{0.9}$$

V tomto prípade platí ze všetky prvky v $X_{left_{k+1}}$ sú menšie než všetky v $X_{mid_{k+1}}$ triviálne, pretože $X_{left_{k+1}} = X_{left_k}$ a $X_{mid_{k+1}} \sqsubset X_{mid_k}$ a teda to vyplýva priamo z indukčnej hypotézy. To isté platí pre sumu $X_{left_{k+1}}$. $X_{right_{k+1}}$ sa skladá z pivota a X_{upper_k} ktoré su väčšie ako prvky v $X_{mid_{k+1}}$ čo priamo vyplýva zo špecifikácie pre PARTITION funkciu. Pre X_{right_k} to vyplýva z indukčnej hypotézy. To že je súčet menší ako $\frac{1}{2}$ vyplýva zo zadania, ktoré požaduje, aby bol súčet ohodnotení rovný jednej. Kedže z podmienky ktorá spôsobila vetvenie platí ze $sumBottom \geq \frac{1}{2}$ a $X_{right_{k+1}}$ tvorí zvyšok zoznamu, potom daná suma musi byt mensia ako $\frac{1}{2}$. Indukcia platí.
V druhom pripade je nove rozdelenie nasledujuce:

$$\begin{aligned} X_{left_{k+1}} &= X_{left_k} \sqcup X_{lower_k} \sqcup [x_{pivotIndex}] \\ X_{mid_{k+1}} &= X_{upper_k} \\ X_{right_{k+1}} &= X_{right_k} \end{aligned} \tag{0.10}$$

Platí že $sumBottom < \frac{1}{2}$, ale kedže X_{right} sa nemení, jej súčet je podľa indukčnej hypotézy $\leq \frac{1}{2}$. Analogicky ako pri predchádzajúcej možnosti, indukčný krok a teda celá indukcia platí.

Konvergencia FOE

Dôkaz konvergencie spočíva v tom, ze časť X_{mid} sa v každom kroku zmenší minimálne o prvok, ktorý bol v predchadzajúcom kroku pivotom. Týmto spôsobom sa v konečnom počte krokov veľkosť X_{mid} zredukuje na jediný prvok. V tomto prípade sa výpočet dostane do podmienej vetvy na riadku 2, a kedže pre tento prvok musí platiť indukčný krok a teda aj výstupná podmienka, algoritmus v konečnom čase vráti správny výsledok. □

Literatúra

[1] BLUM, Manuel, Robert W. FLOYD, Vaughan PRATT, Ronald L. RIVEST a Robert E. TARJAN. Time bounds for selection. Journal of Computer and System Sciences. 1973, vol. 7, issue 4, s. 448-461. DOI: 10.1016/S0022-0000(73)80033-9. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0022000073800339

Príklad 2

Riešenie tohto algoritmu môžeme zjednodušiť na hľadania najväčšieho prvku v poli

```
1: function SEARCH_ACE(matrix, startX, startY, size)
2:     midX ← startX + ⌊size/2⌋
3:     midY ← startY + ⌊size/2⌋
4:     maxX ← 1
5:     maxY ← 1
6:     for i = startX to startX + size − 1 do
7:         if matrix[i][midY] > matrix[maxX][maxY] then
8:             maxX ← i
9:             maxY ← midY
10:        end if
11:    end for
12:    for i = startY to startY + size − 1 do
13:        if matrix[miX][i] > matrix[maxX][maxY] then
14:            maxX ← midX
15:            maxY ← i
16:        end if
17:    end for
18:    if IS_ACE(maxX, maxY) then
19:        return matrix[maxX][maxY]
20:    end if
21:    if SELECT_QUADRANT(maxX, maxY) = 1 then
22:        SEARCH_ACE(matrix, startX, startY, ⌊size/2⌋)
23:    else if SELECT_QUADRANT(maxX, maxY) = 2 then
24:        SEARCH_ACE(matrix, midX, startY, ⌈size/2⌉)
25:    else if SELECT_QUADRANT(maxX, maxY) = 3 then
26:        SEARCH_ACE(matrix, startX, midY, ⌈size/2⌉)
27:    else if SELECT_QUADRANT(maxX, maxY) = 4 then
28:        SEARCH_ACE(matrix, midX, midY, ⌈size/2⌉)
29:    end if
30: end function
31:
32:
33: function IS_ESO(x,y)
34:                                    ▷ Implementacia vynechana, popis v texte
35: end function
36:
37:
38: function SELECT_QUADRANT(x,y)
39:                                    ▷ Implementacia vynechana, popis v texte
40: end function
```

Hlavnou funkciou tohto algoritmu je searchAce. V tejto funkcii najprv vyberieme stredný stĺpec matice a a jej stredný riadok. Tento stĺpec a riadok spolu vytvárajú kríž, ktorý rozdeľuje maticu na rovnaké, resp. skoro rovnaké kvadranty (pri sudom n).
V tomto kríži nájdeme maximálnu hodnotu - zložitosť tejto časti algoritmu je 2n
Pred dôkazom korektnosti doplníme korektnosť funkcií, ktoré sme priamo nedefinovali:
IS_ACE Táto funkcia má za úlohu zistiť, či je prvok na zadaných súradniciach v matici eso. Na toto stačia maximálne 4 porovnania so susednými prvkami (ak tieto prvky existujú). Výsledná zložitosť patrí do $\mathcal{O}(1)$.
SELECT_QUADRANT Táto funkcia vráti číslo z množiny {1,2,3,4}, ktoré reprezentuje 1 zo štyroch možných kvadrantov matice, na ktorých sa funkcia SEARCH_ACE rekurzívne zavolá. Vstupné parametre sú súradnice maximálneho prvku na *kríži*. Funkcia vyhodnotí v ktorom kvadrante sa nachádza najväčší zo štyroch susediacich prvkov (nazveme ho *smerodajný prvok*. Tento najväčší prvok sa musí nachádzať mimo strednu kríža, v opačnom prípade, funkcia IS_ACE vráti true a k tejto funkcii sa vykonávanie nedostane. Túto funkcionalitu je možné implementovať pomocou konštantného počtu porovnaní, čo znamená že výsledná zložitosť patrí do $\mathcal{O}(1)$.
Korektnosť
Parciálna korektnosť. Vstupná aj výstupná podmienka sú devinované v zadaní TODO. Dôkaz korektnosti algoritmu vykonáme matematickou indukciou vzhľadom na počet rekurzívnych volaní. Ak argument funkcie SEARCH_ACE je štvorcová podmatica pôvodnej matice obsahujúca eso , potom kvadrant(podmatica), ktorý algoritmus vyberie a rekurzívne zavolá, tiež obsahuje eso.
bázový krok: funkciu SEARCH_ACE zavoláme na zadanej matici M nasledovným spôsobom SEARCH_ACE(M,1,1,$|M|$), M triviálne obsahuje eso. indukčný krok: Vnútri kvadrantu existuje prvok, ktorý je väčší ako všetky prvky na jeho hranici, konkrétne prvok, na základe ktorého bol kvadrant vybraný funkciou SELECT_QUADRANT. V závislosti na vybranom kvadrante môže byť časť tejto hranice súčasťou kvadrantu, tvrdenie ale platí stále pre zvyšné prvky.
Lemma 1: Vzhľadom k tomu, že matica je zaplnená rôznymi prirodzenými číslami, teda obsahuje aj maximálny prvok a tento prvok je s určitosťou eso.
Keby sa maximálny prvok mohol nachádzať na hranici v rámci kvadrantu mohli by sme ho označiť za eso, ale v skutočnosti by mohol susediť s prvkom mimo kvadrantu, ktorý by bol väčší a teda by sa o eso nejednalo. Maximálny prvok v kvadrante je väčší ako ktorýkoľvek prvok na hranici kvadrantu, pretože maximálny prvok musí byť väčší alebo rovný ako *smerodajný prvok*. To znamená, že kvadrant, nad ktorým zavoláme rekurzívne funkciu SEARCH_ACE obsahuje eso, čo znamená, že indukčný krok platí.
Konvergentnosť. Algoritmus skončí v konečnom počte rekurzívnych volaní, pretože veľkosť podmatice (size) sa v každom volaní zmenší $\lceil \frac{size}{2} \rceil$. V prípade, že size dosiahne hodnotu 1, algoritmus skončí, pretože metóda IS_ACE vráti určite true.
Zložitosť Na výpočet zložitosti využijeme Master Theorem. Zložitosť použitého algoritmu je zadaná ako počet operácii porovnania. Tieto závisia na veľkosti strany vstupnej matice. V každom volaní funkcie sa rekurzívne tento parameter zmenší na polovicu. Naviac určenie kvadrantu do ktorého sa algoritmus rekurzívne zavolá vyžaduje nájdenie maximálneho prvku z dvoch polí dĺžky n. A nakonie sa vykoná jedna s funkcii IS_ACE alebo

SELECT_QUADRANT, ktoré majú konštantnú zložitosť c. Zložistosť algoritmu je teda:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + 2.n + c \\ T(n) &\in \mathcal{O}(n) \end{aligned} \tag{0.11}$$

0.1 Bla Bla

Príklad 3

Príklad 4

Tvrdenie 1: Ľubovoľná postupnosť n operácií INSERT a MIN-ALL má zložitosť O(n).

Uvažujme prirodzené čísla n,k a l, pre ktoré platí n=k+l (n vyjadruje počet operácií)

$$k = \begin{cases} \frac{n}{2} & \text{ak } n \text{ je párne} \\ \frac{n-1}{2} & \text{ak } n \text{ is nepárne} \end{cases}$$

$$l = \begin{cases} \frac{n}{2} & \text{ak } n \text{ je párne} \\ \frac{n-1}{2} + 1 & \text{ak } n \text{ is nepárne} \end{cases}$$

Uvažujme k oprácií INSERT, každá z týchto operácií vloží do zoznamu rovnaké prirodzené číslo z. Po poslednej z týchto operácií bude mať zoznam dĺžku k.

Cena týchto operácií dohromady je k.

Po týchto operáciách nasleduje l operácií MIN-ALL. Všetky čísla v zozname sú rovnaké, teda všetky čísla v ňom sú minimálne. Znamená to, že pri žiadnom z volaní operácie MIN-ALL sa dĺžka zoznamu nezmení.
Cena týchto operácií bude

$$l * k = \begin{cases} \frac{n}{2} * \frac{n}{2} = \frac{n^2}{4} & \text{ak } n \text{ je párne} \\ \frac{n-1}{2} * (\frac{n}{2} + 1) = \frac{n^2}{4} + \frac{n}{4} - \frac{1}{2} & \text{ak } n \text{ is nepárne} \end{cases}$$

Z predošlého tvrdenia vyplýva, že špecifikovaná postupnosť operácií bude minimimálne v zložitostnej triede O(n), teda tvrdenie **neplatí**.

Tvrdenie 2: Ľubovoľná postupnosť n operácií INSERT a MIN-ONE má zložitosť O(n).
Príklad riešime pomocou metódy účtov, kredity pre jednotliv0 operácie stanovíme nasledovne

Operácia	Cena	Kredit
INSERT	1	2
MIN-ONE	|S|	1

Platí, že vždy počas výpočtu je veľkosť zoznamu rovná počtu kreditov na účte, teda počet kreditov nikdy nebude menší ako 0. Celkový kredit po vykonaní n operácií bude menší alebo rovný 2n, teda tvrdenie **platí**.

Tvrdenie 3: Ľubovoľná postupnosť n operácií INSERT a DELETE má zložitosť O(n).

Uvažujme prirodzené čísla n,k a l, pre ktoré platí n=k+l (n vyjadruje počet operácií
Hodnotu čísel k a l stanovíme rovnako, ako pri tvrdení 1.
Uvažujme k oprácií INSERT, každá z týchto operácií vloží do zoznamu rovnaké prirodzené číslo z. Po poslednej z týchto operácií bude mať zoznam dĺžku k.

Cena týchto operácií dohromady je k.

Po týchto operáciách nasleduje l operácií DELETE(y), pričom platí, že y≠z. To má za dôsledok, že po žiadnej z týchto operácií sa dĺžka zoznamu nezmení. Cena týchto operácií bude rovnaká, ako v tvrdení 1. Tvrdenie preto **neplatí**.
Tvrdenie 4: Ľubovoľná postupnosť n operácií INSERT a DELETE taká, že pri každom volaní sa operácia DELETE volá s iným parametrom i, má zložitosť má zložitosť O(n).

Uvažujme prirodzené čísla n,k a l, pre ktoré platí n=k+l (n vyjadruje počet operácií). Hodnotu čísel k a l stanovíme rovnako, ako pri tvrdení 1.
Uvažujme k oprácií INSERT, každá z týchto operácií vloží do zoznamu rovnaké prirodzené číslo z. Po poslednej z týchto operácií bude mať zoznam dĺžku k.

Cena týchto operácií dohromady je k.

Špecifikujeme množinu M o veľkosti l, v ktorej sa nachádzajú prirodzené čísla odližné od z. Vykonáme l operácií DELETE, pričom pri každej jej volaní predložíme ako parameter iný prvok z množiny M. To bude mať za následok, že veľkosť zoznamu sa nezmení. Cena týchto operácií bude rovnaká, ako v tvrdení 1. Tvrdenie preto **neplatí**.

Príklad 5