IV003 - SADA 1

Řešitelé: Tomáš Skopal (374549)
Vojtěch Bělovský (374032)

Příklad 1

Při řešení příkladu vycházíme z algoritmu SELECT, probíraného na první přednášce. Po prvním zavolání algoritmu SELECT máme ukazatel na prvek x, který rozděluje posloupnost na menší a větší prvky než je on sám. V dalším kroku sečteme hodnoty w menších prvků:

$$\sum_{x_i<x} w_i = w$$

a) Pokud je $w < ½$ a zároveň $w+w_x \geq ½$ našli jsme optimální prvek a tedy $x_k = x$. (w_x je hodnota w asociovaná k mediánu x)
b) Pokud je $w > ½$ pak víme, že optimální prvek je v levé části posloupnosti (tedy ta část přes, kterou jsme aktuálně dělali součet). Nazvěme ji S_l.
c) Pokud je součet $w < ½$ a neplatí, že $w+w_x \geq ½$ pak víme, že optimální prvek je v pravé části posloupnosti, nazvěme ji S_p.

Pokud nastane situace c) uložíme si hodnotu w a zavoláme SELECT na posloupnosti prvků > x (S_p). Získáme nový medián y. Sečteme hodnoty w_i v prvcích menších než je nový medián:

$$w + \sum_{x_i>x}^{y} w_i = w_1$$

Kde w znamená součet, který jsme měli uložený z minulého kroku a w_1 je nový součet, přes všechny prvky menší než je medián y. Podíváme se, která ze situací a)b)c) nastala a podle toho rekurzivně postupujeme.

Pokud nastane situace b) znovu zavoláme SELECT na posloupnosti prvků < x (S_l). Najdeme medián a rekurzivně postupujeme, dokud nenastane situace a).

<u>Korektnost algoritmu</u>:

Při volání algoritmu SELECT dostaneme vždy prvek (medián), jež rozděluje posloupnost na prvky menší a větší. Mohou pak nastat jen 3 situace uvedené výše, tedy a), b) nebo c). Z podmínek vidíme, že nikdy nevynecháme žádná čísla, protože v situaci c) máme jistotu, že hledaný optimální prvek se nachází v části posloupnosti S_p, protože všechna čísla v S_l jsou menší než medián a víme, že součet hodnot w je menší než 1/2. Naopak pokud platí podmínka b), tak podle definice optimálního prvku musí takový prvek ležet v levé části posloupnosti (S_l). Protože v každém kroku rekurze zmenším posloupnost, dostanu se až do bodu, kdy mi zbyde jeden prvek a ten musí splňovat podmínku a) a tedy být optimálním prvkem. Tím také ukazujeme, že algoritmus je vždy konečný, protože v každém kroku zmenšíme velikost prohledávané posloupnosti.

<u>Složitost algoritmu</u>:

Složitost každého volání SELECT je lineární (s odvoláním se na přednášky p. Černé). Sečtení hodnot w v posloupnosti S_l je také lineární. Tedy nový krok rekurze probíhá v čase Θ(n). Zároveň vždy postupujeme do S_l nebo S_p v závislosti na součtu hodnot w, takže opačnou část posloupnosti v dalších krocích neuvažujeme. Z toho odvodíme rekurentní rovnici pro výslednou složitost algoritmu.

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

IV003 - SADA 1

Řešitelé: Tomáš Skopal (374549)
Vojtěch Bělovský (374032)

Příklad 2

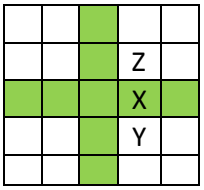
Nejdříve uvedeme hledání esa v 1D poli, které obsahuje n prvků. Princip nám poté poslouží při hledání esa v 2D poli, tedy v matici. Při nejjednodušším hledání esa v 1D poli postupujeme lineárně po jednotlivých prvcích a nejvýše po n krocích nalezneme eso. To je však pomalé a proto urychlíme postup použitím rekurze. Pokud se podíváme na okolí čísla x v 1D poli na indexu ⌈n/2⌉, můžou být okolní čísla buď obě menší (nalezli jsme eso) nebo obě větší nebo jedno větší a jedno menší. Pokud číslo x není esem, podíváme se na sousední prvky a stejný postup opakujeme na tu stranu pole, na které se nachází větší prvek než je x. Časová složitost je dána rekurentně rovnicí

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

tedy podle Master Theoremu O(log n).

Algoritmus je korektní, protože se vždy zanoří do té části pole, kde se nachází větší prvek a tím jednou musí najít ten který má oba sousedy menší (eso).

Princip aplikujeme na matici a to následovně. Najdeme výše uvedeným algoritmem nejvyšší prvek v prostředním sloupci a řádku matice.



Obrázek 1.1

Prozkoumáme jeho okolí. Pokud by podle obrázku 1.1 největším prvkem v prohledávaném řádku a sloupci bylo číslo X, pak porovnáme prvky Z a Y (okolí bodu X) a rekurzivně algoritmus voláme na submatici n/2 * n/2 kde se nachází MAX (X, Y). Do nové submatice (T(n/2)) zahrnujeme i hranici, tedy prvky, které tvořily prostřední sloupec a řádek aktuálně prohledávané matice (T(n)) (zvýrazněné na obrázku 1.1).

Časová složitost:

$$T(n) = T\left(\frac{n}{2}\right) + O(n)$$

Z rovnice podle Master Theoremu určíme koeficienty: a=1, b=2, d=1. Při dosazení zjistíme, že složitost výše popsaného algoritmu je O(n). Tedy jsme splnili zadání.

Korektnost:

Při prohledávání prostředního řádku a sloupce matice najdeme vždy ten největší prvek. Pokud se v jeho okolí nenachází větší prvky, našli jsme eso, pokud je některý z prvků Y a Z větší než aktuální maximum pak rekurzivně postupujeme do submatice, která obsahuje větší z prvků. Máme při tom

jistotu, že hranice nové (poloviční) matice jsou tvořeny pouze menšími prvky než je Y potažmo Z. Tedy eso, jelikož jde o největší prvek ve svém okolí, se musí nacházet v zmenšené části. Konečnost algoritmu je dána stálým zmenšováním matice, až zbyde pouze jeden prvek.

Pozn.: Pravděpodobně by algoritmus byl ještě rychlejší, protože nový krok rekurze probíhá ve složitosti O(2*log(n)), ale pro jednodušší počítání jsme uvažovali tuto složitost jako O(n).

IV003 - SADA 1

Řešitelé: Tomáš Skopal (374549)
Vojtěch Bělovský (374032)

Příklad 3

Implementace operací:

1) MINIMUM – V důsledku toho, že v každém vnitřním uzlu je uložena hodnota small[x] můžeme prohledáváním do hloubky vrátit ukazatel na nejmenší prvek. V každém uzlu (kořenem počínaje) určíme syna s nejmenší hodnotou small[x] a takto rekurzivně postupujeme až do listu. List pak obsahuje nejmenší prvek.
   Hloubka stromu vzhledem k počtu prvků je log(n), tedy výše uvedené prohledání do hloubky má složitost O(log n).

IV003 - SADA 1

Řešitelé: Tomáš Skopal (374549)
Vojtěch Bělovský (374032)

Příklad 4

Korektnost tvrzení:

1) INSERT + MIN-ALL:
   Tvrzení **není** korektní. Existuje totiž posloupnost n operací INSERT a MIN-ALL, které mají kvadratickou složitost. Mějme libovolné číslo i, které budeme n-krát pomocí operace INSERT(S, i) vkládat do seznamu S. Pokud potom budeme volat pouze operaci MIN-ALL(S) bude složitost této operace rovna velikosti seznamu, tedy n. V takovémto případě neexistuje konstanta, kterou bychom mohli shora omezit složitost MIN-ALL(S), proto bude amortizovaná složitost n operací INSERT a MIN-ALL alespoň kvadratická.

2) INSERT + MIN-ONE:
   Tvrzení **je** korektní. Operaci INSERT dám 2 kredity a operaci MIN-ONE dám 1 kredit. Princip je v tom že každý insert vloženému prvku dá 1 kredit na jeho vymazání. MIN-ONE má jeden kredit proto, že můžeme volat MIN-ONE na jednoprvkový seznam, takže potřebujeme 1 kredit, aby se tato operace vždy mohla zaplatit. Vidíme že operacím INSERT a MIN-ONE stačí konstantní počet kreditů, aby se tyto operace navzájem zaplatily, z toho plyne, že jsou omezitelné konstantou, proto můžeme tvrdit, že amortizovaná složitost n operací INSERT a MIN-ONE je v O(n).

Operace	Cena	Kredity	Výsledné kredity
Insert(x)	1	2	1
Insert(x)	1	2	2
Insert(x)	1	2	3
Insert(x)	1	2	….
Insert(x)	1	2	n
Min_One	n	1	n-n+1

3) INSERT + DELETE:
   Tvrzení **není** korektní. Řešení je podobné jako v části 1). Pomocí operace INSERT vložíme do seznamu n-krát číslo i. Když potom budeme volat pouze operaci DELETE s parametrem i, nikdy se žádný prvek nesmaže a složitost operace DELETE bude n. V takovém případě nelze operaci DELETE omezit shora žádnou konstantou, takže amortizovaná složitost n operací INSERT a DELETE je kvadratická.

4) INSERT + DELETE + podmínka na různost *i*:
   Tvrzení **je** korektní. Operaci insert přiřadím 3 kredity a operaci DELETE žádný. Tím, že se DELETE volá pokaždé s jiným parametrem, máme jistotu že pokud vložíme do seznamu S prvek i, pak k jeho smazání dojde nejpozději při druhé operaci DELETE, jež následují v posloupnosti operací za vložením prvku i. Proto každému prvku i stačí přiřadit 3 kredity při jeho vložení - jeden na vložení, druhý na “falešné” smazání (volání DELETE s parametrem i), třetí na jeho opravdové smazání (pokud budeme volat dále operaci DELETE, musíme ji volat s

jiným parametrem než i, proto si můžeme být jisti že prvek již smazán bude a my máme ještě jeden kredit abychom tuto operaci zaplatili).

Operace	Cena	Kredity	Výsledné kredity
Insert(x)	1	3	2
Insert(x)	1	3	4
Insert(x)	1	3	6
Insert(x)	1	3	….
Insert(x)	1	3	2n
Delete(x)	n	0	n
Delete(y)	n	0	0