

Příklad 1

Při řešení příkladu vycházíme z algoritmu SELECT, probíraného na první přednášce. Po prvním zavolání algoritmu SELECT máme ukazatel na prvek x , který rozděljuje 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 $w < \frac{1}{2}$ a zároveň $w + w_x \geq \frac{1}{2}$ našli jsme optimální prvek a tedy $x_k = x$.
- b) Pokud je $w > \frac{1}{2}$ pak víme, že optimální prvek je v levé části posloupnosti (tedy ta část přes, kterou jsme aktuálně dělali součet).
- c) Pokud je součet $w < \frac{1}{2}$ a neplatí, že $w + w_x \geq \frac{1}{2}$ pak víme, že optimální prvek je v pravé části posloupnosti.

Pokud nastane situace c) uložíme si hodnotu w a zavoláme SELECT na posloupnosti prvků $> x$. Od nového mediánu y sečteme hodnotu w_1 , která tentokrát bude rovna

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

Podíváme se, která ze situací a)b)c) nastala a podle toho rekurzivně postupujeme.

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

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 $\lfloor n/2 \rfloor$, 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.

			Z	
			X	
			Y	

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 $\text{MAX}(X, Y)$. 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 $O(2 \cdot \log(n))$, ale pro jednodušší počítání jsme uvažovali tuto složitost jako $O(n)$.

Příklad 3

Implementace operací:

- 1) MINIMUM – V důsledku toho, že v každém vnitřním uzlu je uložena hodnota $\text{small}[x]$ můžeme prohledáváním do hloubky vrátit ukazatel na nejmenší prvek. V každém uzlu (kořenem počínaje) určíme syna s nejmenší hodnotou $\text{small}[x]$ 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 $O(\log n)$.

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 i 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 omezené 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