IV003 - SADA 1

Řešitelé: Tomáš Skopal (374549)

Vojtěch Bělovský (374032)

Příklad 1

Díval jsem se na ty sumy a fakt tam jsou všechna x<xk . Zatím nevím jak z toho ven, ale je pravda, že jsem nad tím moc nepřemýšlel. Když se podíváš do diskuse, tak tam hned na začátku tipek píše ať si projdeme sešit a slidy a že jsme to řešení někde už viděli. Teď mě tak napadlo, jestli by to nešlo vytvořit nějakou haldou nebo B+ stromem. Máš mít složitost O(n) což ti přesně vychází na projití toho pole a když budeš každý prvek brát a skládat do nějaké haldy (fibonacciho možná) tak by to nějak mohlo vyjít, ale je to jen takový nástřel.

IV003 - SADA 1

Řešitelé: Tomáš Skopal (374549)

Vojtěch Bělovský (374032)

Příklad 2

Nejdříve uvedeme hledání esa v 1D poli. 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 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, na které se nachází větší prvek než je x. Časová složitost je dána rekurentně rovnicí

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 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:

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

IV003 - SADA 1

Řešitelé: Tomáš Skopal (374549)

Vojtěch Bělovský (374032)

Příklad 4

Korektnost tvrzení:

• 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 nkrát 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.

Tvrzení **není** korektní. Operaci insert přiřadím 2 kredity a operaci Min-All dám počet kreditů roven počtu minimálního prvku v poli, tedy count(min), maximálně tedy |S|.

Insert má dva kredity, protože 1 spotřebuje a 1 vloží na účet pro případnou následující operaci mazání, která při průchodu přes prvek 1 kredit spotřebovává. Min-All při každém průchodu přes minimální prvek 1 kredit spotřebuje, ale prvek nesmaže, proto mu musí nastavit kredit zpátky. (tohle tady vubec nemusi byt ;-) )

Pokud uvažujeme takový případ, kdy bude pole tvořeno ze stejných čísel, pak Min-All nesmaže nikdy žádné číslo, protože jsou všechna minimální. Z toho vyplývá, že složitost Min-All v takovém případě bude na n operacích O(n2) ( ty nevis jak velky je seznam, protoze jsi na zacatku nerekl kolik tam je cisel, takze muzes tvrdit akorat ze slozitost je O(n\*|S|, kde |S| je velikost seznamu), ta uvaha je dobra, ale technicky neni dobre napsana.). A Celková složitost varianty Insert+MinAll bude O(n2+2n), což je spor.

• INSERT + MIN-ONE:

Tvrzení **je** korektní. Operaci INSERT dám 2 kredity a operaci MIN-ONE dám 1 kredit. Pricip 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 konstatní počet kreditů, aby se tyto operace navzájem zaplatily, z toho plyne že jsou omezitelné konstatnou, proto můžeme tvrdit že amortizovaná složitost n operací INSERT a MIN-ONE je v O(n).

(místo na obrázek - pošlu ti jak vypadá)

Tvrzení **je** korektní. Operaci insert přiřadím 2 kredity a operaci Min-All dám počet kreditů roven počtu minimálního prvku v poli, tedy count(min), maximálně tedy 1 (to není pravda, předtím než volám count min tam může být víc než jeden minimální prvek, takže by podle tvé definice mohl mít Min-All až n kreditů, což je ve sporu s tím co chceš ukázat, a to je to že jde omezit konstantou, ale n není konstanta). Princip je obdobný jako v případě 1). Tedy časová složitost je maximálně rovna součtu kreditů, což je 2n za operaci Insert a n za operaci Min-All. 2n+n=3n což spadá do časové složitosti O(n).

• INSERT + DELETE:

Tvrzení není korektní. Řešení je podobné jako v části a). Pomocí operace INSERT vložíme do seznamu nkrát 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.

Tvrzení **není** korektní. Operaci insert přiřadím 2 kredity a operaci Delete dám počet kreditů roven počtu prvku *i* v poli, tedy count(i), maximálně tedy |S|. Princip je obdobný jako v případě 1).

Pokud uvažujeme takový případ, kdy bude pole tvořeno ze stejných čísel *x* a zároveň budeme operaci Delete volat s parametrem *y != x*, pak se nesmaže nikdy žádné číslo ( je to přesně naopak, smažou se všechna čísla ). Z toho vyplývá, že složitost Delete v takovém případě bude na n operacích O(n2). A Celková složitost varianty Insert+Delete bude O(n2+2n), což je spor.

• 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).

Tvrzení **je** korektní. Operaci insert přiřadím 3 kredity a operaci Delete žádný. Tím, že se Delete volá pokaždé s jiným parametrem, máme jistotu, že maximálně může proběhnout dvakrát po sobě, aby došlo k nějakému mazání prvků ( tuhle větu moc nechápu.. ale předpokládám že tím asi myslís že nejpozdeji při druhém volání DELETE už se musí nějaký prvek smazat..? ). Při dalším (třetím po sobě jdoucím) volání operace Delete už v poli nebou žádné prvky a tím se nespotřebují žádné kredity. 3 kredity v operaci Insert přesně pokrývají tuto situaci (1 kredit při vložení a dva kredity pro dvě po sobě jdoucí operace Delete). Další Insert opět vloží kredity na zásobník. Součtem kreditů (3n) tedy dostáváme složitost O(n). - jakože tohle bych řekl že máš dobře, ale možná bych to trochu přepsal.. nějak jednoznačněji, docela se v tom ztrácím, myslím tím ten příklad jako celek.