Za domácí úkol máte příklady 7 a 8. Důkladně si přečtěte zadání!

Věta 1 (Linearita střední hodnoty)**.** Nechť X a Y jsou náhodné veličiny a α a β jsou reálná čísla. Potom $E[\alpha X + \beta Y] = \alpha E[X] + \beta E[Y]$.

Lemma 1 (O džbánu)**.** Čekáme-li na náhodný jev, který nastane s pravděpodobností p, dočkáme se ve střední hodnotě po $1/p$ nezávislých pokusech.

Příklad 1. Mějme posloupnost matic $A_1, \ldots, A_n$, kterou si chceme uložit tak, abychom rychle uměli odpovídat následující dotazy: Pro dané indexy i a j urči součin matic $A_i \cdots A_j$. Součin matic pomalá operace, takže chceme ukládat pomocné informace takové, že k vyhodnocení dotazu potřebujeme co nejmenší počet součinů matic. Mohli bychom si pamatovat součin matic $A_i \cdots A_j$ pro všechny indexy $i \leq j$, ale takových dvojic je $\binom{n+1}{2}$ a tolik paměti nemáme. Kolik paměti potřebujete, abyste dokázali dotaz vyhodnotit jen na jeden součin dvou matic? Kolik součinů potřebujete, pokud smíte použít jen $\mathcal{O}(n)$ paměti? Můžete předpokládat, že jednu matici lze uložit v $\mathcal{O}(1)$ paměti. Nezapomeňte, že násobení matic je sice asociativní, ale není komutativní ani nelze matice dělit.

Příklad 2. Jaká je průměrná časová složitost bublinkového třídění? Přesněji: pokud dostaneme na vstupu náhodnou permutaci čísel 1 až n, jaká bude střední hodnota doby výpočtu? Předpokládáme, že algoritmus se zastaví, jakmile během jednoho průchodu nic neprohodí.

Příklad 3. Určete časovou složitost v nejhorším případě následujících variant třídících algoritmů založených na hledání souvislých rostoucích podposloupností a jejich slévání.

1. Nejprve jedním průchodem rozdělme vstupní posloupnost na maximální souvislé rostoucí podposloupnosti a poté opakujme následující postup. Mějme k rostoucích podposloupností a pro všechna $i = 1, \ldots, \lfloor k/2 \rfloor$ slijme $(2i-1)$-tou a $(2i)$-tou podposloupnost. Pokud i je liché, tak poslední podposloupnost nezměníme.
2. Nejprve najdeme první maximální rostoucí podposloupnost, tj. maximální k takové, že prvních k prvků je v rostoucím pořadí. Poté opakovaně hledáme další maximální rostoucí podposloupnost a sléváme s první.
3. Rostoucí podposloupnosti (resp. indexy hrajních prvků) bude postupně dávat do zásobníku. V každé fázi algoritmu dáme na vrchol zásobníku ze vstupní posloupnosti další maximální souvislou rostoucí podposloupnost a aplikujeme následující pravidla, dokud je možné některé aplikovat. Nechť k je vždy aktuální počet podposloupností na zásobníku, jejichž délka je značena $l_1, \ldots, l_k$, kde l_1 je délka podposloupnosti na vrcholu zásobníku.
   - Jestliže $k \geq 2$ a $l_2 \leq l_1$, tak slijeme podposloupnosti 1 a 2.
   - Jestliže $k \geq 3$ a $l_3 \leq l_1 + l_2$, tak slijeme podposloupnost 2 s kratší z 1 a 3.

   Všimněme si, že vždy sléváme dvě sousední podposloupnosti a slitou vždy vracíme na tu pozici zásobníku, kde se nacházeli původní dvě podposloupnosti a jedno uvolněné místo vynecháme. Poté, co projdeme celou vstupní posloupnost, tak ještě dotřídíme podposloupnosti na zásobníku postupným sléváním dvou podposloupností na vrcholu zásobníku.

Příklad 4. Jak rychle umíte setřídit n čísel z rozsahu $0, \ldots, n^3 - 1$.

Příklad 5. Quicksort má očekávanou složitost $\mathcal{O}(n \log n)$, ale v nejhorším případě může výpočet trvat $\mathcal{O}(n^2)$. Jak jednoduše upravit quicksort, aby jeho složitost byla $\mathcal{O}(n \log n)$ i v nejhorším případě, ale nedošlo k výraznému zpomalení v průměrném případě?

Příklad 6. Uvažujme hešovaní s řešením kolizí pomocí lineárního přidávání, tj. hešovací funkce nám dá pozici a pokud je obsazená, tak se podíváme na další, dokud nenajdeme první volnou pozici. Jak v této tabulce můžeme prvky mazat (a přitom nerozbít hledání prvků)?

Příklad 7. Mějme množinu přirozených čísel a číslo x. Chceme zjistit, zda množina obsahuje dvojici prvků se součtem x.

Příklad 8. Představme si situaci, že máme n prvků s tak dlouhými klíči, že se nám všechny nevejdou do paměti a musíme si vystačit jen s m-bitovým polem $B[1..m]$. Použijeme hešovací funkci, která klíče mapuje to pole B. Při vkládání prvku x nastavíme $B[h(x)] = 1$. Při dotazu, zda byl prvek vložen, otestujeme, zda $B[h(x)] = 1$. Pokud je x jeden z n vložených prvků, tak určitě dotaz odpoví správně. Pokud se ale zeptáme na prvek x, který nebyl vložen, tak se muže stát, že byl vložen prvek y takový, že $h(x) = h(y)$. Spočítejte, s jakou pravděpodobností se to pro dané n a m stane.