5. QuickSort a složitost třídění *(Michal Staša, Jan Návrat)*

Dostaneme posloupnost, jejíž prvky dovedeme porovnávat, a snažíme se co nejefektivněji posloupnost setřídit. Ukážeme si třídící algoritmus QuickSort (pro přátele QSort nebo QS) založený na metodě Rozděl a panuj:

Algoritmus: (QuickSort)

1. Pokud $|X| \leq 1$, vrátíme X.
2. Vybereme pivota $p \in X$.⟨1⟩
3. $M \leftarrow \{x \in X; x \leq p\}$.
4. $P \leftarrow \{x \in X; x = p\}$.
5. $V \leftarrow \{x \in X; x \geq p\}$.
6. Vrátíme $\mathit{Quicksort}(M) \,.\, P \,.\, \mathit{Quicksort}(V)$. ⟨2⟩

Rozbor: Všimněme si, že QS se určitě zastaví a také že vydá správný výsledek. To můžeme ověřit například indukcí podle $|X|$.

Podobně jako u vybírání k-tého nejmenšího prvku v minulých přednáškách i zde časová složitost závisí hlavně na volbě pivota. Kdybychom za pivota zvolili medián, vyšla by časová složitost stejně jako u MergeSortu:

$$T(n) = 2T(n/2) + O(n) = O(n \log n).$$

Pokud naopak budeme volit pivota nešikovně, dostaneme:

$$T(n) = T(n-1) + \Theta(n) = \Theta(n^2).$$

V ideálním případě bychom tedy chtěli za pivota zvolit medián, avšak jeho přímým výpočtem bychom algoritmus příliš zpomalili. Používá se proto mnoho způsobů, jak vybrat rychle pivota blízkého mediánu. Často používanou metodou je náhodný výběr, v praxi realizovaný nějakým pseudonáhodným generátorem. Ukážeme, že v tomto případě je QS v průměrném případě rychlý:

Věta: QS s náhodnou volbou pivota má časovou složitost v průměru $\mathcal{O}(n \log n)$.

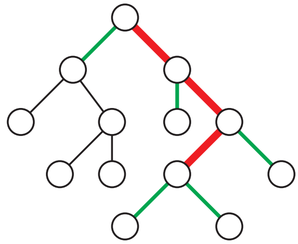
Poznámka: Stejně jako u výběru k-tého nejmenšího prvku bychom také mohli ukázat, že QS s pevnou volbou pivota spuštěný na náhodnou permutaci má tutéž časovou složitost. Detaily nicméně vynecháme.

Důkaz věty:

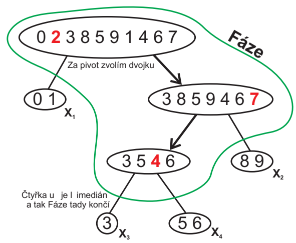
Důkaz provádíme rozdělením algoritmu na fáze. Rekurzivní volání QSortu zle zobrazit jakožto strom, přičemž fází rozumíme cestu ve stromu volání, která sleduje větší díl a končí, když se povede vybrat za pivota lžimedián.

⟨1⟩ později upřesníme, jak

⟨2⟩ Operátor „.“ značí zřetězení seznamů



Důkaz rozdělením na fáze



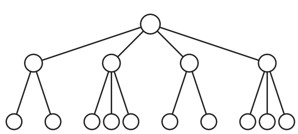
Fáze

Každá fáze přitom rozdělí vstup na disjunktní části $X_1, \ldots, X_k$ $(k \geq 2)$ a pivoty, kteří je oddělují. Označme si n za velikost vstupu (počet prvků vstupní posloupnosti) a n_i za velikost i-té části. Nahlédneme, že platí $\sum_i n_i \leq n$. Velikost každé části je navíc nejvýše $3/4 \cdot n$ (na konci fáze to platí proto, že jsme zvolili lžimedián, předtím jsme vždy oddělili menší z částí, čili nejvýše $n/2$).

Jedna iterace algoritmu trvá $\mathcal{O}(n)$ a jelikož lžimedián vybereme s pravděpodobností alespoň $1/2$, je v jedné fázi v průměru $\mathcal{O}(1)$ iterací a celá fáze proto v průměru trvá čas $\mathcal{O}(n)$. Z toho dostaneme následující rekurenci pro průměrnou časovou složitost celého algoritmu:

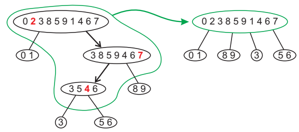
$$\mathbb{E}T(n) = \sum_i \mathbb{E}T(n_i) + \mathcal{O}(n).$$

Tento typ rekurence jsme ještě nepotkali a Kuchařková věta[3] na ni nezabere, ovšem můžeme si pomoci jednoduchou úvahou. Představme si, že v našem stromu rekurzivních volání zkomprimujeme každou fázi do jednoho vrcholu. Tím vznikne strom, který odpovídá algoritmu, jenž v jedné iteraci v průměrně lineárním čase rozdělí vstup na několik částí a rekurzivně se na ně zavolá.



Komprimovaný Strom

[3] MasterTheorem



Způsob vytvoření komprimovaného stromu

Nový strom má logaritmickou hloubku, protože na každé další hladině jsou délky posloupností nejvýše 3/4 délek z předchozí hladiny. Navíc součet délek přes každou hladinu je maximálně n. Proto na každé hladině trávíme čas v průměru $\mathcal{O}(n)$ a v celém stromu tedy $\mathcal{O}(n \log n)$. ♡

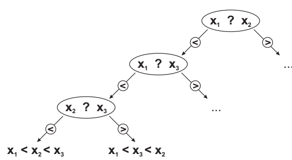
Pozorování: Naše první verze QS spotřebuje lineární množství paměti na pomocné pole a na zásobník. Na přednášce jsme ukazovali různá jeho praktická vylepšení, které stačí pomocná paměť o velikosti $\mathcal{O}(\log n)$. Detaily viz webové stránky přednášky.

Známe už několik třídících algoritmů s časovou složitostí $\mathcal{O}(n \log n)$. Následující věta ukazuje, že efektivnější algoritmus v obecném případě neseženeme.

Věta: Každý třídící algoritmus založený na porovnávání a prohazování prvků potřebuje na vstup délky n v nejhorším případě $\Omega(n \log n)$ porovnání.

Důkaz: Bez újmy na obecnosti budeme nejdříve předpokládat o algoritmu dvě věci: Jednak to, že algoritmus nejprve porovnává, a teprve potom prohazuje.⟨4⟩ Také předpokládáme, že vstup algoritmu je permutace na množině $\{1, \ldots, n\}$.

Chování tohoto algoritmu popíšeme rozhodovacím stromem. V rozhodovacím stromu vnitřní vrcholy určují jednotlivá porovnání prvků a listy odpovídají okamžikům, kdy algoritmus přestal porovnávat a začal prohazovat.



Rozhodovací Strom

Vstup je tedy permutace n prvků, a víme že počet různých permutací je $n!$. Existuje tedy právě $n!$ různých vstupů. Dále si všimneme, že nemohou existovat dvě vstupní permutace, pro které by algoritmus skončil v témže listu rozhodovacího stromu. Listů stromu je tedy alespoň tolik, kolik je různých vstupů, tedy $n!$.

Lemmátko: Binární strom hloubky k má nejvýše 2^k listů.

Důkazík: Uvažme binární strom hloubky k s maximálním počtem listů. V takovém stromu budou všechny listy určitě ležet na poslední hladině (kdyby neležely, můžeme pod některý list na vyšší hladině přidat další

⟨4⟩ Algoritmus můžeme upravit tak, aby si pamatoval aktuální permutaci prvků a podle ní prohazoval až na konci.

dva vrcholy a získat tak „listnatější“ strom stejné hloubky). Jelikož na i-té hladině je nejvýše 2^i vrcholů, všech listů je nejvýše 2^k. ♡

Z tohoto lemmátka plyne, že rozhodovací strom musí být hluboký alespoň $\log n!$.

Zbytek je už snadné cvičení z diskrétní matematiky:

Lemmátko: $n! \geq n^{n/2}$.

Důkazík: $n! = \sqrt{(n!)^2} = \sqrt{1(n-1) \cdot 2(n-2) \cdot \ldots \cdot n \cdot 1}$, což můžeme také zapsat jako $\sqrt{1(n-1)} \cdot \sqrt{2(n-2)} \cdot \ldots \cdot \sqrt{n \cdot 1}$. Přitom pro každé $1 \leq k \leq n$ je $k(n+1-k) = kn+k-k^2 = n+(k-1)n+k(1-k) = n+(k-1)(n-k) \geq n$. Proto je každá z odmocnin větší nebo rovna $n^{1/2}$ a $n! \geq (n^{1/2})^n = n^{n/2}$. ♡

Hloubka stromu tedy činí minimálně $\log n! \geq \log(n^{n/2}) = n/2 \cdot \log n = \Omega(n \log n)$, což také zdola odhaduje počet porovnání, který algoritmus provede v nejhorším případě. ♡