Kapitola 1

Státnice - Třídění

1.1 Třídění založené na porovnávání

Existuje mnoho algoritmů, známe i (za určitých podmínek) dolní odhad složitosti.

Heapsort

HEAPSORT je třídění pomocí (např.) d-regulárních hald, jen lokální podmínku haldy používáme v duální podobě a máme funkci **DELETEMAX** (která ale funguje stejně jako **DELETEMIN**). Postupně se odebírají maxima a setříděná posloupnost se staví na jejich místě od konce pole. Iterace končí, když v haldě zbývá jen jeden prvek. Celkem $O(n \log n)$ operací.

Mergesort

MERGESORT je snad nejstarší "chytrý" třídící algoritmus. Pracuje s frontou, do které na počátku nahází "předtříděné" rostoucí úseky, potom je v cyklu vybírá a jejich slití vrací zpátky, dokud nemá jen jednu posloupnost. Slití je vybrání vždy menšího na výstup a na konci dokopírování zbytků.

Jedna operace slití vyžaduje $O(n + m)$, jsou-li 2 posloupnosti dlouhé n a m prvků. První rozházení vyžaduje lineární čas, potom každé projití všech prvků slitím vyrábí posloupnosti délky $\geq 2^{i-1}$, tedy počet projití všech prvků je $\lceil \log n \rceil$ a celková složitost $O(n \log n)$. Je adaptivní na předtříděné posloupnosti a při omezeném počtu rostoucích úseků dosahuje lineární složitosti.

Quicksort

QUICKSORT je asi nejpoužívanější třídící algoritmus, v průměru má při rovnoměrném rozložení nejnižší očekávaný čas. Využívá techniky rozděl a panuj.

1. Vybere prvek a (pivot).
2. Vytvoří posloupnosti $a_1 \dots a_{k-1}$ prvků menších než a a $a_{k+1} \dots a_n$ větších než k.
3. Na ty sám sebe zavolá rekurzivně, do výsledku zapíše za sebe obě setříděné poloviny.

Procedura (bez rekurze) vyžaduje čas k nebo $n - k$ v jednom běhu, tj. pro a medián, kdyby $n - k = k$, by měl celý algoritmus složitost $O(n \log n)$. Medián lze nalézt v lineárním čase, ale pak by byly **MERGESORT** i **HEAPSORT** rychlejší, proto se jako a bere např. první prvek posloupnosti.

Potom má procedura očekávaný čas $O(n \log n)$, ale nejhorší případ $O(n^2)$, což je pro neznámé rozdělení dat nevhodné. Náhodná volba by celý běh také mohla dost zpomalit, proto se v praxi bere medián tří až pěti pevně zvolených prvků.

Očekávaný čas Quicksortu

Dva libovolné prvky a_i, a_j jsou porovnávány maximálně jednou, a to když a_i nebo a_j je pivot a předtím ani jeden z nich pivot nebyl. Vezmeme si náhodnou veličinu $X_{i,j}$, která má hodnotu 1, když **QUICKSORT** porovnal během výpočtu b_i a b_j z nějaké výsledné setříděné posloupnosti $(b_i)_{1 \leq i \leq n}$, a 0 jinak. Potom $\mathbf{E}X_{i,j} = p_{i,j}$, kde $p_{i,j}$ je pravděpodobnost porovnání. Potom celk. počet porovnání v celém běhu je

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbf{E}X_{i,j} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} p_{i,j}$$

Pro výpočet $p_{i,j}$ uvažujeme "strom rekurze", kde každý vrchol odpovídá jednomu rekurzivnímu volání procedury a s tím i nějaké podposloupnosti $a_i, \dots, a_j$ (do té už se sahá jen v jeho podstromě). V jeho levém podstromě jsou

operace na úsecích prvků menších než pivot $a_i, \dots a_{l-1}$ této posloupnosti a v pravém na větších $a_{l+1}, \dots a_j$. Přiřadíme každému vrcholu jeho pivot a očíslujeme vrcholy prohledáváním do šířky. Pak $X_{i,j} = 1$, když b_j nebo b_i je první pivot z množiny $\{b_i, \dots b_j\}$ v tomto očíslování (protože kdyby to byl jiný prvek z této množiny, rozdělím b_i a b_j, aniž bych je porovnal; naopak prvky mimo tuto množinu coby pivoty od sebe b_i a b_j neoddělí). Množina $\{b_i, \dots, b_j\}$ má $j-i+1$ prvků, takže $p_{i,j} = \frac{2}{j-i+1}$. Počet porovnání pak vyjde

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \leq 2n \sum_{k=2}^{n} \frac{1}{k} \leq 2n \int_1^n \frac{1}{x} \mathrm{d}x = 2n \ln n$$

Jednoduché třídění

- **SELECTIONSORT** třídí vybíráním nejmenšího prvku a jeho prohozením s levým krajním v nesetříděném úseku.
- **INSERTSORT** vkládá do setříděného úseku další prvek a postupným vyměňováním ho řadí na správné místo.
- **SHELLSORT** je jeho vylepšení – postupně **INSERTSORT**em třídí sekvence složené z každého k-tého prvku pro klesající $k \to 1$ (sekvence klesajících k musí být zvolena "šikovně").
- **BUBBLESORT** – iterativně prochází posloupností a prohazuje inverze
- Jeho varianta **SHAKESORT**, která posloupnosti prochází tam a zpátky.

A-sort

Tento algoritmus je aplikací (a, b) stromů v třídících algoritmech, vhodnou pro částečně předtříděné posloupnosti. Jinak proti klasickým algoritmům nemá žádné výhody. Pro algoritmus je nutné znát list s nejmenším prvekm FIRST, cestu k němu od kořene a pro každý list ukazatele na následující v uspořádání NEXT.

Postup: Odzadu (od "předtříděně největšího") vkládat prvky do stromu modifikovaným A-INSERTem a pak přečíst posloupnost listů (jít po NEXT). A-INSERT pracuje tak, že místo pro vložení prvku hledá od FIRST (jde postupně nahoru po otcích a hledá, kde nejdřív může slézt zas k listům).

Složitost: Pomalejší než běžné třídění na libovolná data (asymptoticky stejné), ale rychlejší na částečně předtříděná. Vezmeme F – počet inverzí v posloupnosti. Celk. potřebuju $O(n)$ pro načtení prvků, $O(n)$ pro všechna štěpení dohromady ze všech běhů A-INSERTu a na každé vložení $O(h)$ pro nalezení místa, kde h je výška, kam se z FIRST dostanu, přeskočím tak $f_i \geq a^{h-2}$ vrcholů (menších než vkládaný) a $h \in O(\log f_i)$. Součet f_i za $\forall i$ dává:

$$\sum_{i=1}^{n} \log f_i = n \log(\prod_{i=1}^{n} f_i)^{\frac{1}{n}} \leq n \log \frac{\sum_{i=1}^{n} f_i}{n} = n \log \frac{F}{n}$$

Protože se nepoužívá DELETE, hodí se na toto $(2,3)$ stromy. Pro míru $F \leq n \log n$ má složitost $O(n \log \log n)$, v urč. případech i rychlejší než Quicksort.

Porovnání algoritmů

Algoritmus	Čas nejhůř	Čas ∅	Porov. nejhůř	Porov. ∅	PP	Paměť	AD
QUICKSORT	$\Theta(n^2)$	$9n \log n$	$n^2/2$	$1.44n \log n$	A	$n + \log n + c$	N
HEAPSORT	$20n \log n$	$\leq 20n \log n$	$2n \log n$	$2n \log n$	A	$n + c$	N
MERGESORT	$12n \log n$	$\leq 12n \log n$	$n \log n$	$n \log n$	N	$2n + c$	A
A-SORT	$O(n \log(F/n))$	$O(n \log(F/n))$	$O(n \log(F/n))$	$O(n \log(F/n))$	A	$5n + c$	A
SELECTIONSORT	$2n^2$	$2n^2$	$n^2/2$	$n^2/2$	A	$n + c$	N
INSERTSORT	$O(n^2)$	$O(n^2)$	$n^2/2$	$n^2/4$	N	$n + c$	A

c je nějaká konstanta, F značí počet inverzí v posloupnosti, PP – přímý přístup, AD – adaptivní na předtříděné.

Pro krátké posloupnosti je do délky 22 vhodný **SELECTIONSORT**, do 15 **INSERTSORT**, jinak **QUICKSORT**, což vede k hybridnímu algoritmu. Pro **A-SORT** jsou nejvhodnější $(2,3)$-stromy. Poměr časů **QUICKSORT, MERGESORT, HEAPSORT** je v průměru $1 : 1.33 : 2.22$, platilo to ale v roce 1984 :-).

Vylepšení Mergesortu

Nedosahuji optimálních výsledků, pokud slévané posloupnosti ve frontách nejsou přibližně stejně dlouhé. Proto provedu úvahu: mějme algoritmus, který slévá rostoucí posloupnosti a uvažujme jeho "slévací" strom T (kde posloupnost $P(v)$ odp. vrcholu v (délky $l(P(v))$) vznikne slitím posloupností z jeho synů). Součet časů pro MERGESORT je pak $O(\sum\{l(P(v))|v \text{ vnitřní vrchol } T\}) = O(\sum\{d(t)l(P(t))|t \text{ list } T\}$. Dále pracujeme jen s délkami posloupností,

vytvoříme algoritmus **OPTIM**, který při slévání sumu minimalizuje – na začátku dá každé jednoprvkové posl. hodnotu c, která odpovídá hodnotě jejího prvku (?). Pro slévání vybírá posloupnosti (stromy) s nejmenším c a slitému stromu přiřadí $c_1 + c_2$. Nakonec zbyde v množině stromů jen jeden, a ten je optimální. Pro třídění fronty podle c se používá BUCKETSORT. Celkem pracuje v čase $O(\sum_{i=1}^{n} l(P_i))$ na posloupnosti rostoucích úseků $P_1, \ldots, P_n$.

1.2 Přihrádkové třídění

Bucketsort (Counting sort)

Algoritmus **BUCKETSORT** třídí jen přirozená čísla z intervalu $< 0, m >$ a to zavedením $m+1$ množin, do kterých je rozhází a nakonec tyto spojí do výsledku. Třídění je stabilní pro opakující se prvky, inicializace množin a projití při konkatenaci potřebují $O(m)$, rozházení prvků pak $O(n)$, takže celkem $O(n+m)$.

Varianta **RADIXSORT** umí třídit i ve větších intervalech, když používá BUCKETSORT na každou jednotlivou číslici. Protože BUCKETSORT je stabilní, bude to celé fungovat.

Hybrid Sort

Sofistikovanější verze **HYBRIDSORT** třídí reálná čísla z $\langle 0, 1\rangle$ (a obecně tedy jakékoliv klíče). Má dané α (počet přihrádek v poměru k n), rozhazuje do $k = \alpha n$ přihrádek a ty pak třídí haldou.

Nejhorší možný čas je $O(n \log n)$, protože nejhůře se může stát, že všechny prvky nacpu do 1 přihrádky. Očekávaný čas pro nezávislé rovnoměrně rozdělené prvky je $O(n)$ – pravděpodobnost velikostí množin se řídí binomickým rozdělením s parametrem $1/k$, střední hodnota pak je:

$$\mathbf{E}(\sum_{i=1}^{k}(1 + X_i \log X_i)) \leq \mathbf{E}(\sum_{i=1}^{k}(1 + X_i^2)) = k + k\left(\frac{n(n-1)}{k^2} + \frac{n}{k}\right) = O(n)$$

Jak je vidět z odhadu, i kdybychom použili algoritmus s kvadratickou složitostí ve třídění jednotlivých přihrádek, zůstane očekávaná složitost lineární.

Wordsort

WORDSORT je modifikace BUCKETSORTu pro třídění slov. Příprava:

1. Rozhodí slova do množin L_i podle jejich délky.
2. Vytvoří dvojice pozice-písmeno $\{(j, a_i[j])\}$, kterou setřídí podle druhé složky BUCKETSORTem, výsledek setřídí podle první složky (stejně), tj. má seznam setříděných dvojic pozice-písmeno, které se ale mohou opakovat.
3. Dvojice rozstrká do množin S_i pro každou pozici i a odstraní duplicity.

Pak v hlavním cyklu postupuje od největší možné délky a pro každé i pracuju jen s množinou slov délky $\geq i$:

1. Podle i-tého písmena rozhodím všechna aktuálně tříděná slova do množin T_x.
2. Potom podle množiny S_i vyberu všechna neprázdná T_x a sliju je za sebe.
3. Pro další krok přidávám kratší slova vždy na začátek množiny aktuálně tříděných.

Výpočet délek slov, inicializace L_i a zařazení slov do L_i vyžadují čas $O(L)$, kde L je součet délek všech řetězců. Vytvoření seznamu dvojic a jeho třídění vyžaduje také $O(L)$. Založení S_i a přeházení dvojic do nich také $O(L)$. Inicializace T_x je $O(|\Sigma|)$, kde $|\Sigma|$ je velikost abecedy. V hlavním cyklu v každém kroku potřebuji dvakrát čas $O(l_i)$ (součet délek slov dlouhých i nebo víc). Celkem tedy $O(L + |\Sigma|)$.

1.3 Pořadové statistiky (hledání mediánu)

Vstupem je (neuspořádáná) posloupnost n (navzájem různých) čísel. Výstup je $\lfloor \frac{n}{2} \rfloor$-té, nebo obecně k-té nejmenší číslo z nich. Složitost budeme měřit počtem porovnání.

Z dvou následujících **FIND** bývá rychlejší než **SELECT** pro většinu případů, ale nemá zaručenou asymptotickou složitost. Je známo, že medián lze nalézt na $3n$ porovnání a že dolní odhad počtu porovnání je $2n$.

Hledání mediánu technikou rozděl a panuj (algoritmus FIND)

Tento algoritmus používá techniku "rozděl a panuj". chová se podobně jako **QUICKSORT** a hledá obecně k-té nejmenší číslo:

1. Vybrat pivota a rozdělit posloupnost pomocí $n-1$ porovnání na 3 oddíly: čísla menší než pivot (a prvků), pivota samotného a čísla větší než pivot ($n-a-1$ prvků).
2. (a) Pokud je $a+1<k$, hledám rekurzivně $k-a+1$-tý prvek v $n-a-1$ prvcích
   (b) Pokud je $a+1=k$, vracím pivot
   (c) Pokud je $a+1>k$, hledám rekurzivně k-tý prvek v a prvcích

Rekurzivní vzorec $T(n)=T(n-1)+(n-1)$ v nejhorším případě, což dává $\Theta(n^2)$. Očekávaný čas odhadneme na $4n$ a dokážeme indukcí podle n z rekurzivního vzorce $T(n,i)=n+\frac{1}{n}\left(\sum_{k=1}^{i-1}T(n-k,i-k)+\sum_{k=i+1}^{n}T(k,i)\right)$.

Zaručeně lineární hledání mediánu (algoritmus SELECT)

Vylepšení, garantující dobrý výběr pivota a lineární složitost i v nejhorším čase:

1. Rozdělit posloupnost na pětice (poslední může být neúplná, mějme threshold např. $n=100$, pod kterým množinu přímo třídíme)
2. V každé pětici najít medián
3. Rekurzivně najít medián těchto mediánů
4. Použít ho jako pivot pro dělení celé posloupnosti, protože je větší než alespoň 3 prvky z alespoň 1/2 pětic (až na zakrouhlení), je větší než alespoň $1/2\cdot 3/5=3/10$ prvků (a také menší než alespoň 3/10 prvků)
5. Z toho mám vždy zaručeno, že zahodím alespoň 3/10 prvků pro následující krok.

Vzorec potom vychází: $T(n)=c\cdot\frac{n}{5}+T(\frac{n}{5})+(n-1)+T(\frac{7}{10}n)$ a podle Master Theoremu nebo substituční metodou se dá vyčíslit jako $T(n)=\Theta(n)$.