Medians and Order Statistics

Nechť A je množina obsahující n různých prvků:

Definice: Statistika i-tého řádu je i-tý nejmenší prvek, tj.,

- minimum = statistika 1. řádu
- maximum = statistika n-tého řádu
- median(s) = $\lfloor (n+1)/2 \rfloor$ a $\lceil (n+1)/2 \rceil$ - pro sudý počet prvků existují dva mediány nejčastěji se uvažuje první z uvedených případů

Algoritmus výběru : Pro dané i, určit statistiku i-tého řádu
vstup: Množina A n (***různých***) čísel a hodnota i, $1 \le i \le n$
výstup: Prvek $x \in A$ který je větší než $(i - 1)$ prvků množiny A

Nalezení Minima (Maxima)

Čas výpočtu:
- jediný průchod polem
- pouze n-1 porovnání

```
Minimum(A)
1. lowest ← A[1]
2. for i ← 2 to n do
3.     lowest ← min(lowest, A[i])
```

Je tohle nejlepší možný čas potřebný pro nalezení minima (maxima)? Ano!
Proč je nutné n – 1 porovnání ?

- Algoritmus vyhledávající minimum musí porovnat každý prvek s „vítězem“ (nalezení minima/maxima lze chápat jako turnaj, ve kterém musí být n-1 pořážek ke stanovení vítěze)

O(nlgn) řešení algoritmu výběru

Algoritmus výběru : Pro dané i, určit statistiku i-tého řádu
vstup: Množina A n (***různých***) čísel a hodnota i, $1 \le i \le n$
výstup: Prvek $x \in A$ který je větší než $(i - 1)$ prvků množiny A

```
NaiveSelection(A, i)
1. A' ← FavoriteSort(A)

2. return A'[i]
```

Čas výpočtu:
*O(n*lgn*)* pro algoritmy řazení, které jsou založené na porovnávání
Může být lepší???

Základní myšlenka: Použít O(nlg(n)) algoritmus pro řazení čísle, (heapsort, mergesort), pak vybrat i-tý prvek pole.

Nalezení Minima & Maxima

Co když chceme současně vyhledat minimum a maximum ?

Kolik porovnání je nutné provést ?

- Postup A: nalezneme minimum a maximum odděleně tj. n - 1 porovnání pro min a pro max, tj. celkem 2n -2 porovnání
  Lze to udělat lépe ?

- Postup B: Zpracováváme prvky po dvojicích. Nejprve se porovná vzájemně porovná dvojice prvků vstupního pole a menší hodnota se porovná s dosavadním minimem, větší hodnota pak s maximema podle výsledků porovnání se uraví aktuální hodnota minima a maxima.
  Cena = 3 porovnání pro každé 2 prvky.
  Celková cena = $3\lfloor n/2 \rfloor$.

Finding Minimum & Maximum

Postup B: Zpracováváme prvky po dvojicích. Nejprve se porovná vzájemně porovná dvojice prvků vstupního pole a menší hodnota se porovná s dosavadním minimem, větší hodnota pak s maximem a podle výsledků porovnání se upraví aktuální hodnota minima a maxima.
Cena = 3 porovnání pro každé 2 prvky.
Celková cena = $3\lfloor n/2 \rfloor$.

FindMin&Max(A)
1.if length[A] % 2 == 0
2. then if A[1] > A[2]
3. then min = A[2]
4. max = A[1]
5. else min = A[1]
6. max = A[2]

7. else // n % 2 == 1
8. then min=max=A[1]
9. Compare the rest of the elements in pairs, comparing only the maximum element of each pair with max and the minimum element of each pair with min

Analýza FindMin&Max

- Je-li n sudé, potřebujeme 1 počáteční porovnání a pak $3(n-2)/2 + 1$ porovnání $= 3n/2 - 2$
- Je-li n liché, potřebujeme $3(n-1)/2$ porovnání
- V obou případech, je maximální počet porovnání $\leq 3\lfloor n/2 \rfloor$

FindMin&Max(A)
1.if length[A] % 2 == 0
2. then if A[1] > A[2]
3. then min = A[2]
4. max = A[1]
5. else min = A[1]
6. max = A[2]

7. else // n % 2 == 1
8. then min=max=A[1]
9. Compare the rest of the elements in pairs, comparing only the maximum element of each pair with max and the minimum element of each pair with min

Výběr statistiky i-tého řádu v lineárním čase

- Randomized-Select vrací i-tý nejmenší prvek A[p..r].

Randomized-Select(A, p, r, i)
1. **if** p = r **then return** A[p]
2. q ← Randomized-Partition(A, p, r)
3. k ← q – p + 1
4. **if** i = k **then return** A[q]
5. **else if** i < k **then return** Randomized-Select(A, p, q-1, i)
6. **else return** Randomized-Select(A, q+1, r, i - k)

Randomized-Partition(A, p, r)
1. j ← Random(p, r)
2. swap A[r] ↔ A[j]
3. **return** Partition(A, p, r)

Výběr statistiky i-tého řádu v lineárním čase

- Algoritmus Randomized-Partition nejprve zamění A[r] s náhodně zvoleným prvkem A a pak zavolá proceduru Partition použitou v algoritmu QuickSort.

Randomized-Partition(A, p, r)
1. j ← Random(p, r)
2. swap A[r] ↔ A[j]
3. **return** Partition(A, p, r)

Partition(A, p, r)
1.x ← A[r]
2.i ← p – 1
3. for j ← p to r - 1 do
4.if A[j] ≤ x then
5. i ← i + 1
6. swap A[i] ↔ A[j]
7. swap A[i+1] ↔ A[r]
8. return i + 1

Doba výpočtu algoritmu Randomized-Select

- Nejhorší případ : při nešťastném výběru vznikne n - 1 částí. $T(n) = T(n - 1) + \theta(n) = \theta(n^2)$
  (stejné jako nejhorší případ u algoritmu QuickSort)
- Nejlepší případ : při vhodné volbě se rychle redukuje velikost částí $T(n) = T(n/2) + \theta(n)$
- Průměrný případ : jako Quick-Sort, asymptoticky se bude blížit nejlepšímu případu

Určení statistiky i-tého řádu v lineárním čase (v nejhorším případě)

Modifikovaná verze algoritmu Partition x je vstupní parametr , který obsahuje hodnotu prvku, okolo kterého se vytvářejí části.

```
Partition(A, p, r, x)
1. i ← p - 1
2. for j ← p to r - 1 do
3.     if A[j] ≤ x then
4.         i ← i + 1
5.         swap A[i] ↔ A[j]
6. swap A[i+1] ↔  A[r]
7. return i + 1
```

Určení statistiky i-tého řádu v lineárním čase (v nejhorším případě)

Key: Zaručíme-li a „dobré" rozdělení když separujeme pole na dílčí části pak bude algoritmus běžet v lineárním čase.

```
Select(A, p, r, i)     /* i je i-tý hledaný prvek v pořadí. */
1. Rozdělit vstupní pole A ⌊n/5⌋ skupin velikosti 5 prvků ve
   skupině (a jedna zbývající skupina, pokud n % 5 != 0)
2. Určit medián každé skupiny o velikosti 5 metodou insert-
   sort pro 5 zvolit prostřední prvek.
3. volat Select rekurzívně, abychom určili hodnotu x, což je
   medián z ⌈n/5⌉ mediánů.
4. Rozdělit celé pole okolo prvku x, do dvou polí
   A[p, q-1] and A[q+1, r] a vrátit q, index bodu rozdělení
    (použít modifikovaný Partition Algoritmus viz dále).
5.  Nechť k = q – p + 1
6. if (i = k) return x  (předpokládá, že index  x je k)
   else if (i < k) then
        call Select(A, p, q-1, i)
   else call Select(A, q+1, r, i – k)
```

Doba výpočtu procedury of Select

<u>Doba běhu (jednotlivé kroky):</u>

1. $O(n)$ (rozdělení do skupin po 5 prvcích)
2. $O(n)$ (seřazení 5 prvků a nalezení mediánu je $O(1)$ časová složitost)
3. $T(\lceil n/5 \rceil)$ (recurzívní volání k nalezení mediánu mediánů)
4. $O(n)$ (rozdělení pole)
5. $T(7n/10 + 6)$ (maximální velikost podproblému)

Celková doba zpracování je dána rekurentně

$$T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n)$$

Doba výpočtu procedury of Select

Řešení rekurentní rovnice metodou odhadu. Odhadujeme

$T(n) \leq cn$

$$
\begin{aligned}
T(n) &= T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) \\
&\leq c\lceil n/5 \rceil + c(7n/10 + 6) + O(n) \\
&\leq c((n/5) + 1) + 7cn/10 + 6c + O(n) \\
&= cn - (cn/10 - 7c) + O(n) \\
&\leq cn
\end{aligned}
$$

Když n >= 80 (cn/10 -7c) nabývá kladných hodnot

Zvolíme-li dostatečně velké c tak O(n) + (cn/10 -7c) je kladné a platí předchozí řádek. (Např. c = 200)