Třídění

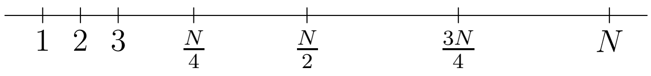
1 Hledání polohy v tabulce

Máme tabulku N hodnot x_i setříděnou podle velikosti.
Pro dané x hledáme do kterého podintervalu $\langle x_i, x_{i+1})$ patří.

Hledání polohy náhodně zadaných bodů x

Postupné prohledání nevýhodné - N operací (nejhůře), $N/2$ průměrně

Výhodná strategie - půlení intervalu indexů - $\sim \log_2 N$ operací
Postup - Vezmeme index $[N/2]$ a porovnáme x a $x_{[N/2]}$.
Je-li $x < x_{[N/2]}$, pak porovnat x a $x_{[N/4]}$, je-li $x > x_{[N/2]}$, pak porovnat x a $x_{[3N/4]}$.



Obrázek 1: Hledání polohy náhodného bodu v tabulce

Algoritmus

```
klo := 1; khi := N;
while (khi-klo) > 1 do begin
   knew = (khi+klo) div 2;
   if (x < x[knew]) then khi := knew else klo := knew
end;
```

Posloupnost zadávaných bodů

Předchozí postup je vhodný pouze pro náhodnou posloupnost bodů. Pokud jsou zadávány body x postupně podle velikosti, je výhodné zahájit hledání v intervalu, kde ležel předchozí bod a pak v sousedních intervalech.

2 Řazení (třídění)

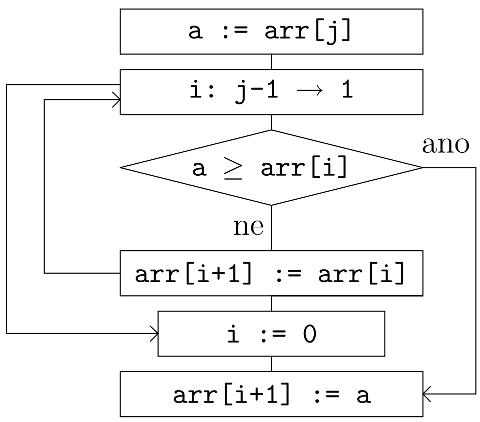
Seřazení N čísel podle velikosti - proces řádu složitosti $\sim N \log_2 N$.
Zopakovat N-krát proces vyhledání intervalu přímo nelze.

<u>**Malý počet čísel**</u> lze seřadit i algoritmem řádu $\sim N^2$ s malou multiplikativní konstantou.

Ukážeme následující algoritmy

- **Přímé vkládání** – složitost $\sim N^2$ $(\rightarrow N \lesssim 50)$
- **Shellova metoda** – složitost $\sim N^{\frac{3}{2}}$ (pro náhodná čísla $\sim N^{1.27}$) $(\rightarrow 50 \lesssim N \lesssim 1000)$
- **Heapsort** – složitost $\sim N \log_2 N$ $(\rightarrow N \lesssim 50)$
  Názorný algoritmus, nepotřebuje žádnou paměť navíc. Nejhorší případ je jen asi o 20 % pomalejší než průměrná rychlost.
- **Quicksort** – složitost $\sim N^2$ - nejhorší případ (pro náhodná čísla $\sim N \log_2 N$)
  náhodná $N \gtrsim 1000$ – nejrychlejší známý – 1.5 až 2 $times$ rychlejší než Heapsort. Ovšem v nejhorším případě (pro správně seřazená čísla) má Quicksort složitost $\sim N^2$.

2.1 Přímé vkládání



Obrázek 2: Algoritmus přímého vkládání

Postupně zprava posunujeme prvky v poli nahoru a necháváme místo volné pro vložení, dokud je vkládané $a[j]$ menší než $a[i]$.

2.2 Shellova metoda

Tato metoda spočívá v řazení prvků s proměnným krokem. Proti třídění přímým vkládáním urychlý přesun prvků na větší vzdálenost.

1. V prvním kroku řadíme prvky vzdálené o $[N/2]$.
   Seřadíme $\forall$ dvojice $(1, [N/2]+1)$, $(2, [N/2]+2)$, $\ldots$, $(N-[N/2], N)$.
   Například pro $N = 17$, tedy celá část $\left[\frac{N}{2}\right] = 8$,
   řadíme prvky s indexy $(1, 9)$, $(2, 10)$, $\ldots$, $(8, 16)$, $(9, 17)$.

2. Dále seřadíme všechny čtveřice s prvky vzdálenými o $[N/4]$.
   V našem příkladě je $\left[\frac{N}{4}\right] = 4$, a tedyo třídíme $(1, 5, 9, 13)$, $(2, 6, 10, 14)$, $(3, 7, 11, 15)$, $(4, 8, 12, 16)$ a $(5, 9, 13, 17)$.

3. Ve třetím kroku třídíme osmice s prvky vzdálenými o $[N/8]$.

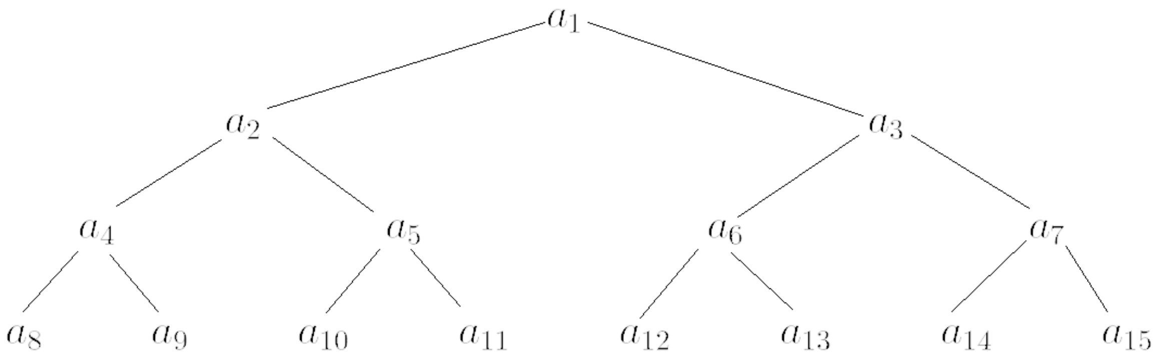
4. Poslední krok je řazení N čísel vkládáním, přehazujeme ale jen sousední čísla. Třídíme tedy jednu skupinu všech čísel.

2.3 Heapsort

Heapsort neboli třídění pomocí **kupy (haldy, hromady)** využívá hierarchické třídění. Každý nadřízený prvek má dva podřízené. Řazení má dva kroky.

1. Vystavíme strukturu tak, aby prvky byly seřazeny ve všech větvích. Ke 2 prvkům přiřadím 3 tak, aby byl větší byl nahoře.

2. Největší prvek dáme na konec pole. Ve zbylé části prvky postupně povyšujeme a nejvyšší dáme na konec neseřazené části pole.



Mezi jednotlivými prvky této haldy platí vztahy $a_1 \geq a_2$ a $a_1 \geq a_3$, dále $a_2 \geq a_4$ a $a_2 \geq a_5$, a tak dále, tedy $a_n \geq a_{2n}$ a $a_n \geq a_{2n+1}$.
V prvním kroku k prvkům a_8 a a_9 přidáme a_4 a seřadíme je podle daných vztahů. Dále k prvkům a_4 a a_5 přidáme a_2 a seřadíme je, pokud dojde k přehození, řadíme oddíl, ve kterém došlo ke změně.

Když takto sestavíme kupu, máme už prvek a_1 zařazený (je to nejvyšší prvek). Proto jej odsuneme na konec pole. Další nejvyšší je $\max(a_2, a_3)$, ten přejde na vrchol kupy a musíme opět upravit oddíl, ve kterém se změnil nejvyšší prvek. Pole máme seřazené, když se dostane poslední člen do čela kupy.

2.4 Quicksort

Vybereme v poli jeden prvek, tak zvaný **pivot**, pole rozdělíme na dvě podpole a přeházíme prvky tak, aby v jednom podpoli byly prvky menší než pivot a ve druhém větší než pivot. Pivot už tedy máme zařazený na správném místě. Nyní stejný postup aplikujeme na obě podpole až do té doby než dostaneme úseky se 7 nebo méně prvky. Tyto úseky je již efektivnější třídit přímým vkládáním.

Pozn. Pokud dělící prvky vybíráme odleva a pole je již seřazené, má algoritmus $\sim N^2$ operací. Pro pole náhodných prvků je ovšem nejrychlejší.

Pozn. Pokud je nebezpečí, že pole může být seřazené, je lépe dělící prvky vybírat náhodně.

3 Pole indexů, pořadí

Pole indexů `Index[i]` nám určuje místo, na kterém je i-tý prvek podle velikosti. Pole pořadí `Pořadí[i]` určuje kolikátý je podle velikosti i-tý prvek. Pole indexů hledáme tímto postupem:

1. Přiřadíme `Index[j] := j`.
2. Porovnáváme prvky, ale místo `arr[j]` používáme `arr[Index[j]]`.
3. Přehazujeme prvky v poli `Index[j]`.

Prvky pole pořadí hledáme pomocí pole indexů podle vztahu
`Pořadí[Index[j]] := j`.

Pořadí v poli	Původní pole	Pole indexů	Pole pořadí	Uspořádané pole
1	14	5	4	3
2	8	4	3	7
3	32	2	6	8
4	7	1	2	14
5	3	6	1	15
6	15	3	5	32

V této tabulce tedy první prvek pole indexů číslo 5 znamená, že pátý prvek původního pole je nejmenší. Dále čtvrtý prvek je druhý v pořadí a třetí prvek je největší. První prvek pole pořadí říká, že na prvním místě původního pole je čtvrtý prvek podle velikosti, druhý prvek je třetí podle velikosti a poslední prvek je podle velikosti pátý.

4 Třídy ekvivalence

Chceme vytvořit pole `CisTridEq[n]` takové, že
$(j \Leftrightarrow k) \Longleftrightarrow (\texttt{CisTriEq}[j] = \texttt{CisTridEq}[k]$. Tedy j je ekvivalentní k, právě když jsou hodnoty pole `CisTridEq` v obou bodech stejné.
2 možnosti zadání

1. Seznam podmínek, tedy vektory `lista` a `listb`. Tyto podmínky pro všechna $i = 1, \ldots, M$ znamenají, že pokud $\texttt{lista}[i] = j$ a $\texttt{listb}[i] = k$, potom j a k jsou ekvivalentní.
2. Boolovská funkce `Equiv`, pro kterou platí, že `Equiv(j, k) = True`, právě když j je ekvivalentní k.