Algoritmi in podatkovne strukture – 2

Urejanje (*sorting*)

končne množice

Urejanje

Imamo vhodni niz predmetov: $X = [x_1, x_2, \ldots x_n]$.

Za poljubna predmeta velja, da sta urejena, kar pomeni, da je eden večji ali enak kot drugi: $x_i \leq x_j$ ali $x_i \leq x_j$ in da je ta relacija tranzitivna.

Rezultat urejevalnega postopka je niz predmetov

$$X' = [x'_i \mid (x'_i = y_j) \text{ in } x'_i \leq x'_{i+1}]$$

Urejanje

Imamo vhodni niz predmetov polj: $X = [x_1, x_2, \ldots x_n]$, kjer $x_i = x_{i,l-1}x_{i,l-2}\ldots x_{i,0}$ in $x_{i,j} \in [0 \ldots M-1]$.

Za poljubna predmeta velja, da sta urejena, kar pomeni, da je eden večji ali enak kot drugi: $x_i \leq x_j$ ali $x_i \leq x_j$ in da je ta relacija tranzitivna.

Rezultat urejevalnega postopka je niz predmetov

$$X' = [x'_i \mid (x'_i = y_j) \text{ in } x'_i \leq x'_{i+1}]$$

Stabilnost

Imamo vhodne ključe, ki pa sedaj *niso* različni: npr. $x_{i,l} = x_{j,l}$.[1]

Če $i < j$ in, če v urejenem nizu nastopi $x_{i,l-i}$ pred $x_{j,l}$, pravimo, da je urejanje *stabilno*, sicer ni – z drugimi besedami, njun relativni vrstni red *se ni spremenil*.

[1] Isto se je dogajalo že doslej, a se nismo posebej ukvarjali s tem pojavom.

Urejanje s štetjem

Imamo vhodni niz predmetov polj: $X = [x_1, x_2, \ldots x_n]$, kjer $x_i = x_{i,l-1}x_{i,l-2}\ldots x_{i,0}$, $l = 0$ in $x_{i,j} \in [0 \ldots M-1]$.

OSNOVNA IDEJA: Za vsak element x_i določimo *število elementov, ki so manjši od* x_i. To informacijo uporabimo za to, da postavimo x_i točno na njegovo mesto v tabeli!

Primer: če imamo 17 elementov, ki so manjši od x_i, potem je x_i na 18. mestu (na indeksu 17).

Urejanje s štetjem – *Counting Sort*

```
public int[] CountingSort (int[] X, int M) {
  int[] R= new int[X.length];
  int[] C= new int[M];
  for (i= M-1, i >= 0, i--)          C[i]= 0;
  for (i= A.length-1, i >= 0, i--)   C[ A[i] ]++;
  for (i= 1, i < M, i++)             C[i]+= C[i-1];
  for (i= A.length()-1, i >= 0, i--) {
    R[ C[ A[i] ] ]= A[i];
    C[ A[i] ]--;
  }
  return R;
}
```

(ZAKAJ NEKATERI ŠTEVCI TEČEJO NAVZGOR IN NEKATERI NAVZDOL?)

(ALI JE UREJANJE STABILNO ALI NE? OD ČESA JE ODVISNO?)

(KAKO BI IZGLEDALA VZPOREDNA VARIANTA UREJANJA?)

Časovna zahtevnost

Imamo štiri zanke:

- šteje do M
- šteje do N
- šteje do M
- šteje do N

kar pomeni, da je čas $O(M + N)$. Kadar $N = O(M)$, takrat dobimo čas $O(N)$.

(ALI JE LAHKO $N > M$?)

(KAKO JE MOŽEN ČAS $O(N)$, ČE SMO ZADNJIČ VIDELI, DA JE $\Omega(N \log N)$?)

Korensko urejanje *radix sort*

Imamo vhodni niz predmetov polj: $X = [x_1, x_2, \ldots x_n]$, kjer $x_i = x_{i,l-1}x_{i,l-2}\ldots x_{i,0}$ in $x_{i,j} \in [0 \ldots M-1]$, kjer je števka $x_{i,0}$ najmanj pomembna.

```
public int[][] RadixSort(int[][] X) {
  int[][] R= new int[X.length][X[0].length];
  for (i= 0, i < X[0].length, i++)
    R[i]= StabileSort(X[i])
  return R
}
```

Časovna zahtevnost

- imamo zanko dolžine l
- pri vsakem ponavljanju se izvede eno stabilno ureejanje ($O(N \log N)$, $O(M + N)$
- skupaj: $O(lN \log N)$, $O(lM + lN)$.

Zahtevnost

urejanje	**čas**
z vstavljanjem	$O(n^2)$
z zlivanjem	$O(n \log n)$
s kopico	$O(n \log n)$
hitro urejanje	$O(n^2)$
korensko	$O(lN \log N)$, $O(lM + lN)$

Vprašanja:

- Kaj je pravzaprav l?
- So ta urejanja primerljiva med seboj?