

SDA – Seminar 3

1. Sortări

A. BucketSort

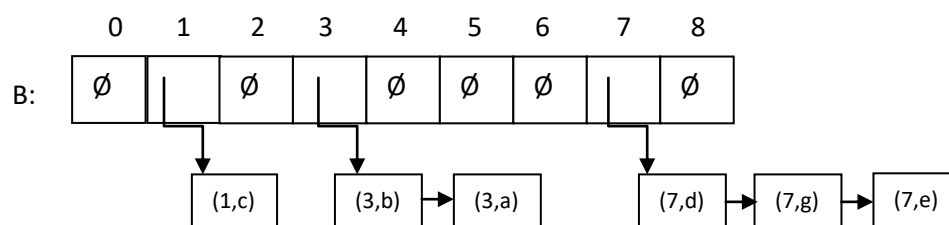
- Se dă un şir S de n perechi (cheie, valoare), cheile $\in [0, N-1]$
- Se cere să sortăm S după chei

Ex. S: (7, d) (1, c) (3, b) (7, g) (3, a) (7, e) \Rightarrow (1, c) (3, b) (3, a) (7, d) (7, g) (7, e)

N = 9

Ideea:

- Se foloseşte un şir de şiruri auxiliar B, de dimensiune N
- Fiecare pereche se pune pe poziţia corespunzătoare în B (B[c])
- Se parcurge B (de la 0 la N-1) şi se mută perechile din fiecare subşir al lui B la finalul secvenţei S.



Operaţii pe şir:

- vid (şir)
- prim (şir): poziţie
- şterge (şir, poziţie)
- adaugăFinal(şir, element)

Algoritm BucketSort(S, N) este:

//presupunem că şirul B există

Cât timp \neg vid (S) execută:

$p \leftarrow$ prim (S)

$(c, v) \leftarrow$ şterge (S, p)

 adaugaFinal (B[c], (c,v))

sf_cât timp

Pentru $i \leftarrow 0, N-1$, execută:

 Cât timp \neg vid (B[i]) execută:

$p \leftarrow$ prim (B[i])

$(c, v) \leftarrow$ şterge (B[i], p)

 adaugaFinal (S, (c,v))

 sf_cât timp

sf_pentru

sf_algoritm

Complexitate: $\Theta(N + n)$

Observaţii:

- Cheile trebuie să fie nr. naturale (deoarece sunt folosite ca și indici).
- Se păstrează ordinea perechilor care au aceeași cheie (sortare stabilă - stable sort)

B. Sortare Lexicografică

d-tuplu (x_1, x_2, \dots, x_d)

$$(x_1, x_2, \dots, x_d) < (y_1, y_2, \dots, y_d) \Leftrightarrow x_1 < y_1 \vee (x_1 = y_1 \wedge ((x_2, \dots, x_d) < (y_2, \dots, y_d)))$$

- Se face compararea după prima dimensiune, după aceea după a 2-a, etc.

Se dă o secvență S de tuple. Se cere să sortăm S în ordine lexicografică.

Vom folosi:

- C_i – obiect comparator (relație) care compară 2 tuple folosind dimensiunea i
- *stableSort*(S, c) – algoritm de sortare care folosește un comparator c

Sortarea Lexicografică execută alg. *StableSort* de d ori (odată pentru fiecare dimensiune)

Algoritm *LexicographicSort*(S):

Pentru $i \leftarrow d, 1$ exec:

stableSort(S, c_i)

sf_pentru

sf_subalgoritm

Complexitate: $\Theta(d * T(n))$

unde $T(n)$ – complexitatea algoritmului *StableSort*

Ex. (7, 4, 6) (5, 1, 5) (2, 4, 6) (2, 1, 4) (3, 2, 4)

Sortare după $d = 3$: (2, 1, 4) (3, 2, 4) (5, 1, 5) (7, 4, 6) (2, 4, 6)

Sortare după $d = 2$: (2, 1, 4) (5, 1, 5) (3, 2, 4) (7, 4, 6) (2, 4, 6)

Sortare după $d = 1$: (2, 1, 4) (2, 4, 6) (3, 2, 4) (5, 1, 5) (7, 4, 6)

C. Radix Sort

- O specializare a sortării lexicografice, în care algoritmul de sortare stabilă folosit este *BucketSort* → toate elementele din tuple trebuie să fie numere naturale cuprinse între $[0, N-1]$.
- Complexitate: $\Theta(d * (n + N))$

Sf_pentru

I. Reprezentare secvențială:

a. $\Theta(n)$

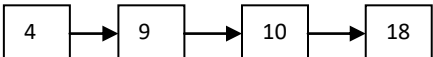
b. $\Theta(n)$

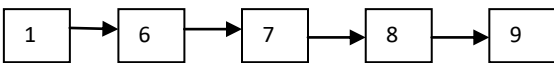
II. Reprezentare înlănțuită

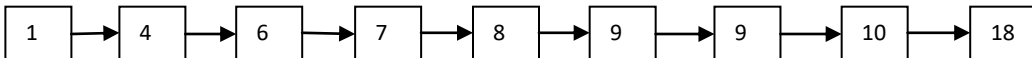
a. $\Theta(n)$

b. $\Theta(n^2)$

2. Scrieți o procedură care interclasează 2 liste simplu înlănțuite sortate. Analizați complexitatea operației.

L1. 

L2. 

Rezultat: 

Reprezentare:

NodT:

e: TComparabil

urm: \uparrow NodT

Lista:

prim: \uparrow NodT

R: relație

Subalgoritm interclasare (L1, L2, LR):

// nu modificăm listele L1 și L2

curentL1 \leftarrow L1.prim

curentL2 \leftarrow L2.prim

primLR \leftarrow NIL

ultimLR \leftarrow NIL

cât timp curentL1 \neq NIL și curentL2 \neq NIL execută

aloca(nou)

[nou].urm \leftarrow NIL

dacă [curentL1].e < [curentL2].e atunci

[nou].e \leftarrow [curentL1].e

curentL1 \leftarrow [curentL1].urm

altfel

[nou].e \leftarrow [curentL2].e

curentL2 \leftarrow [curentL2].urm

sf_dacă

dacă primLR = NIL atunci

primLR \leftarrow nou

ultimLR \leftarrow nou

altfel

[ultimLR].urm \leftarrow nou

ultimLR \leftarrow nou

sf_dacă

sf_cât timp

curent \leftarrow curentL1

dacă curentL1 = NIL atunci

curent \leftarrow curentL2

sf_dacă

cât timp curent \neq NIL execută

alocă (nou)

[nou].urm \leftarrow NIL

[nou].e \leftarrow [curent].e

curent \leftarrow [curent].urm

dacă primLR = NIL atunci

```
        primLR ← nou
        ultimLR ← nou
    altfel
        [ultimLR].urm ← nou
        ultimLR ← nou
    sf_dacă
    sf_cât timp
        LR.prim ← primLR
Sf_subalgoritm
Complexitate:  $\Theta(n + m)$ 
n – lungimea listei L1
m – lungimea listei L2
```