Seminar 3 SDA

Alg. de Sortare

1. Alg de sortare
   1. Bucket sort
   2. z
   3. z
2. Interclasare liste simplu inlantuite ordonate

**I1. Bucket sort**

Enunt: Se da o secventa S formata din n perechi <cheie, valoarea> a. i. c apartine lui {0,1...N-1}

Se cere sa se sorteze S dupa chei.

Ex. (7,d),(1,c),(3,b),(7,g),(3,a),(7,e);

n = 6;

N = 9;

B[0...8] [0.....N-1]

|  |
| --- |
| Ø |
|  |
| Ø |
|  |
| Ø |
| Ø |
| Ø |
|  |
| Ø |

B 0

1 (1,c)

2

3 (3,b) – (3,a)

4

5

6

7 (7,d) – (7,g) – (7,e)

8

S’ = (1,c),(3,b),(3,a),(7,d),(7,g),(7,e);

Pseudocod:

TAD Secventa

adaugaFinal(Secventa (i/o), <TCheie, TValoare>(i))

stergeInceput(Secventa(i/o))

prim(Secventa(i)) => <TCheie, TValoare>

vida(Secventa(i)) => A/F

**Alg de sortare**

Subalgoritm BucketSort(S,N)

I

cat timp !vida(S) executa

<c,v> <- prim(S)

θ(n) adaugaFinal (B[c], <c,v>)

stergeInceput(S)

Sfcat timp

SfI

Pentru i<-0, N-1 executa:

cat timp !vid(B[i]) executa:

adaugaFinal(S, prim(B[i]))

O(N\*n) stergeInceput(B[i])

Sfcat timp

SfPentru

SfSubalgoritm

complexitate:

θ(max{n, N}) = θ(n+N)

N ∈ O(n) =>T(n) ∈ θ(n)

* c ∈ {-a, -a+1, ..., 0, ... a-1, a}

a ∈ N;

2a+1 bucket-uri

B[0.....2a]

<c,v> ->B[c+a]

* c ∈ {a, a+1,...b}, a <- N

<c,v> -> B[c-a]

* c ∈ {‘A’, ...’Z’}

<c,v> -> B[ASCII(c)-ASCII(‘A’)]

* [0,1)

B[0...9] N = 10 v->B[[v\*10]]

N = n

v -> B[[v\*N]]

**I2. Sortarea lexicografica**

Enunt: Se da o secevnta S de n d-tupluri de forma (x1, x2...xd).

Se cere sa se sorteze S lexicografic.

Ex.

S: (7,4,6), (5,1,5), (2,4,6), (2,1,4), (3,2,4)

S’: (2,1,4), (2,4,6), (3,2,4), (5,1,5), (7,4,6)

1 (dupa prima componenta): (2,4,6), (2,1,4), (3,2,4), (5,1,5), (7,4,6)

2 (dupa a doua componenta): (2,1,4), (5,1,5), (3,2,4), (2,4,6), (7,4,6)

3 (dupa a treia componenta): (2,1,4), (3,2,4), (5,1,5), (2,4,6), (7,4,6) X(nu este ce doream)

Ci, i = 1,d

stableSort(S, Ci)

Subalgoritm lx(S, Ci = 1,d)

Pentru j <-d,1,-1 executa

stableSort(S, Cj)

SfPentru

SfSubalgoritm

**I3. Radix Sort**

* Sortarea lexicografica cu BucketSort

T(n,N,d) = θ(d\*(n+N)

=> θ(d,n) => θ(n)

d - const

N ∈ O(n)

n – nr tupluri

N – nr maxim

d – nr de elemente din tuplu

II. Enunt: Se dau 2 LSIO (Liste simple inlantuite ordonate) L1 si L2 .

Sa se interclaseze L1 si L2 intr-o noua LSIO LR

Ex.

L1: 5 -> 8 -> 11

L2: 7 -> 12

LR: 5 -> 7 -> 8 -> 11 -> 12

Nod LSIO

e: Tcomp prim: ^Nod (^ - pointer)

urm: ^Nod r: Relatia: TComp x TComp - > {A,F}

Subalgoritm interclasare(L1, L2, LR)

p1 <- L1.prim

p2 <- L2.prim

LR.prim <- NIL

LR.r <- L1.r

ultim <- LR.prim

Cat timp p1 != NIL si p2 != NIL executa:

nod <- aloca()

[nod].urm <- NIL [] – dereferentiere

Daca L1.r([p1­].e, [p2].e) atunci:

[nod].e <- [p1].e

p1 <- [p1].urm

altfel

[nod].e <- [p2].e

p2 <- [p2].urm

SfDaca

Daca LR.prim = NIL atunci:

LR.prim <- nod

altfel:

[ultim].urm <- nod

SfDaca

ultim <- nod

SfCat timp

p <- p1

Daca p = NIL atunci:

p<-p2

SfDaca

Cat timp p != NIL executa

nou <- aloca()

[nou].urm <-NIL

[nou].e <- [p].e

p <- [p].urm

Daca LR.prim = NIL atunci:

LR.prim <- nou

altfel:

[ultim].urm <- nou

SfDaca

ultim <- nou

SfCat timp

SfSubalgoritm