Seminar 3

1. Algoritmi de sortare
   A. Bucket Sort
   B. Sortare Lexicografica
   C. Radix Sort
2. Interclasarea a 2 liste simplu inlantuite

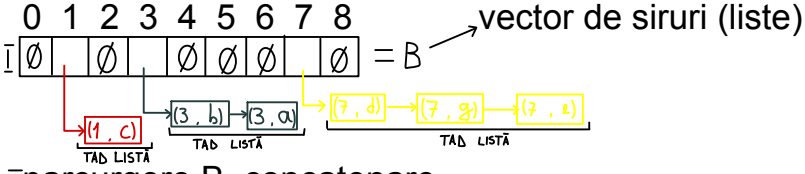
A. **Bucket Sort**

Se da un sir S cu n perechi (cheie, valoare)
cheie[i] apartine {0, 1, ..., N-1}
i = 1,n
Se cere sa se sorteze dupa chei

Ex : S : (7,d), (1,c), (3,b), (7,g), (3,a), (7,e)
n = 6
N = 8 (sau >= 8)
Ssortat : (1,c), (3,b), (3,a), (7,d), (7,g), (7,e)

ALG STABIL

Algoritm stabil : pentru chei egale, pastreaza in sirul sortat ordinea cheilor din sirul initial



IIparcurgere B, concatenare

S : (1,c), (3,b), (3,a), (7,d), (7,g), (7,e)
TAD LISTĂ

-> vida(lista) : verifica daca lista e vida
-> prim(lista) : TPozitie
-> adaugaSfarsit(lista,(c,v))
-> sterge(lista,p)
TPozitie

```
algoritm BucketSort
    //presupunem ca avem B[0...N-1]
    cat timp !vida(S) executa
        p <- prim(S)
        (c,v) <- sterge(S,p)
        adaugaSfarsit(B[c],(c,v))
    sf cat timp
    pentru i <- 0,N-1 executa
        cat timp !vida(B[i]) executa
            p <- prim(B[i])
            (c,v) <- sterge(B[i],p)
            adaugaSfarsit(S,(c,v))
        sf cat timp
    sf pentru
sf algoritm
```

Θ(n) pt. LDI

Θ(max(N,n))

Complexitate totala : Θ(max(N,n))

1) cheie[i] apartine {a,a+1,...,b}
   c -> B[c-a]
   B de lungime b-a+1
2) cheie[i] apartine {-a,-a+1,...,a-1,a}
   c -> B[c+a]
   B de lungime 2a+1

3) cheie[i] apartine {'A',...'Z'}
c -> B[ASCII(c)-ASCII('A')]
B de lungime 26 (lungimea alfabetului)
4) cheie[i] apartine [0,1)
c -> B[parte_intreaga(c*10)]

B. Sortare Lexicografica
Se da o secventa S de n d-tupluri (tupluri cu d componente) de forma (x1,...,xd)
Se cere sa se sorteze S lexicografic, adica (x1,x2,x3,...,xd) $<_{lexicografic}$ (y1,...,yd) <=> (x1<y1) sau (x1=y1 si (x2,...,xd) < (y2,...,yd))
S : (7,4,6), (5,1,5), (2,4,6), (2,1,4), (3,2,4)
n = 5
d = 3
Ssortat : (2,1,4), (2,4,6), (3,2,4), (5,1,5), (7,4,6)

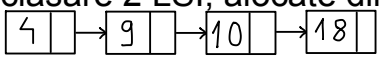
Folosim radix sort ca sa sortam
Radix Sort :
-> sortare lexicografica
+Bucket Sort ca algoritm de sortare

Initial : (7,4,6), (5,1,5), (2,4,6), (2,1,4), (3,2,4)
dupa d = 1 : (2,4,6), (2,1,4), (3,2,4), (5,1,5), (7,4,6)
dupa d = 2 : (2,1,4), (5,1,5), (3,2,4), (2,4,6), (7,4,6)
dupa d = 3 : (2,1,4), (3,2,4), (5,1,5), (2,4,6), (7,4,6)

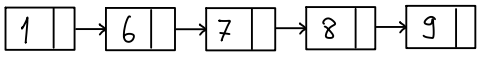
Initial : (7,4,6), (5,1,5), (2,4,6), (2,1,4), (3,2,4)
dupa d = 3 : (2,1,4), (3,2,4), (5,1,5), (7,4,6), (2,4,6)
dupa d = 2 : (2,1,4), (5,1,5), (3,2,4), (7,4,6), (2,4,6)
dupa d = 1 : (2,1,4), (2,4,6), (3,2,4), (5,1,5), (7,4,6)

-> algoritm de sortare pentru fiecare dimensiune stabil pentru a pastra ordinea obtinuta anterior
algoritmul sortare_lexicografica $\Theta(d \cdot T(n))$
pentru i <- d,1
stableSort(S,ci) //ci este comparatorul pe care il folosesc pentru fiecare dimensiune
sf algoritmul sortare_lexicografica

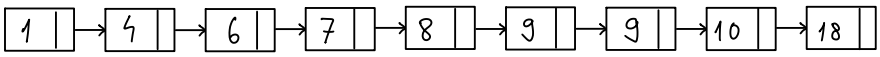
2. Interclasare 2 LSI, alocate dinamic
L1 :



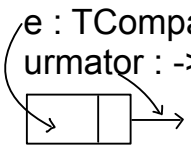
L2 :



L3 :



Nod
e : TComparabil
urmator : ->Nod



LSI
primul : ->Nod
R : Relatie. TComparabil x TComparabil -> {A,F}

algoritm interclasare (L1,L2,LR)
curentL1 <- L1 primul
curent L2 <- L2 primul

```
primLR <- NIL
ultimLR <- NIL
cat timp curentL1 != NIL si curentL2 != NIL executa
    (*) aloca(nou)
    [nou].urmator <- NIL
    daca L1.R([curentL1].e,[curentL2].e) atunci
        [nou]e <- [curentL1].e
        curentL1 <- [curentL1].urmator
    altfel
        [nou]e <- [curentL2].e
        curentL2 <- [curentL2].urmator
    sf daca
    daca primLR = NIL
        primLR <- nou
    altfel
        [ultimLR].urmator <- nou
    sfdaca
    ultimLR <- nou
+ prelucare elem ramase
```