SD – Seminar 10
22.12.2020

SORTARE (continuare)

Sortare prin numarare:

Input: n, a[0..n-1], si a[i] ∈{1,2,…,k} → **O(n+k)**

Exemplu:

k = 5, n = 9

	0	1	2	3	4	5	6	7	8
a=	4	3	3	4	1	2	1	3	3

Tabloul de frecvente: c[i] = numarul de valori din a[] **egale** cu i (1<=i<=k)

	0	1	2	3	4	5
c=		2	1	4	2	0

O(k + n) – initializare c[] + parcurgerea lui a cu incrementarea lui c[a[i]]

Tabloul de frecvente cumulate: c[i] = numarul de valori din a[] **mai mici sau egale** decat i.

	0	1	2	3	4	5
c=		2	3	7	9	9

O(k)

sorare : O(n)

Simulare exemplu:

	0	1	2	3	4	5
c=		0	2	4	8	9

	0	1	2	3	4	5	6	7	8
b=	1	1	2	3	3	3	3	4	4

Pr. 1 Consideram n valori intregi din intervalul [1, k]. Propuneti un algoritm care pre-proceseaza intrarea in O(n+k) si determina cate valori sunt ın intervalul [x,y] in O(1) (x,y ∈{1,2,…,k} si x<=y).

```
function preprocesare( n, k, a[], c[] )  \\  O(n+k)
begin
        for j <- 0 to k do c[j] <- 0
        for j <- 0 to n-1 do c[a[j]] <- c[a[j]] + 1
        for j <- 2 to k do c[j] <- c[j] + c[j-1]
end
```

```
function numara( c[], x, y )  \\ O(1)
begin
        return c[y] – c[x-1]
end
```

Sortare prin distribuire:

Input: n, a[0..n-1], si valorile a[i] sunt distribuite **uniform** peste intervalul [0,1) → complexitatea medie O(n)

Se formeaza **n** liste cu lungime medie **1**.

struct nod {
    double inf
    nod * succ
}

nod * B[0..n-1]

a[i] se isereaza in lista cu primul nod aflat la adresa B[floor(n * a[i])]
exemplu: (n = 10)   0.37 va fi inserat in lista B[3]

Exemplu:

n = 10

	0	1	2	3	4	5	6	7	8	9
a=	0.41	0.57	0.92	0.35	0.11	0.28	0.76	0.85	0.65	0.97

O(n) – parcurgerealui a[] cu inserare lui a[i] in B[floor(n * a[i])] in timp constant.

B[0] = ()
B[1] = (0.11)
B[2] = (0.28)
B[3] = (0.35)
B[4] = (0.41)
B[5] = (0.57)
B[6] = (0.65)
B[7] = (0.76)
B[8] = (0.85)
B[9] = (0.97, 0.92) -> (0.92, 0.97)

lungimea unei liste B[i] este, in medie 1.
Listele B[i] se sorteaza folosind o alta metoda - in medie timp constant. – O(n)
tabloul sortat se obtine prin parcurgerea acestor liste sortate. O(n)

ARBORI BINARI DE CAUTARE

Problema cautarii:

U – multime univers
$S \subset U$, $|S| = n$
$a \in U$

$a \in S$?

Exemple:

i) U – lista studentilor din anul I si S – grupa E1
ii) U – Z multimea numerelor intregi si S – o multime de numere intregi

Structura de date (**pentru S**)

- lista oarecare → cautarea are complexitatea O(n)
- lista ordonata, in implementarea cu tablouri → alg. de cautare binare → cautarea are complexitatea O(log n) \\ T(n) = T(n/2) + 1
    - inserarea, stergerea O(n) !

Exemplu n = 10^6

10^6 comparatii vs. log 10^6 ~ 20 comparatii

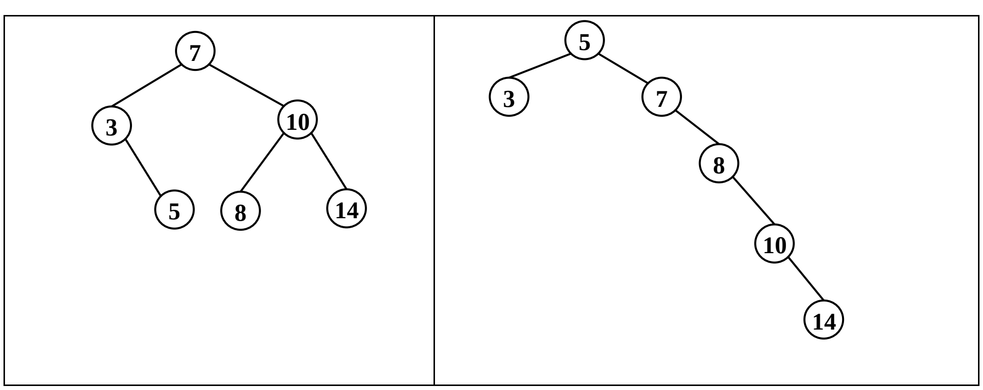
Aspectul dinamic al operatiei de cautare: asupra lui S se efectueaza operatii de inserare / stergere

Dorim gasirea unei structuri de date care sa ne asigure complexitatea in cazul cel mai nefavorabil, O(log n) pentru operatiile de cautare, inserare, stergere.

Arbori binari de cautare (ABC):

Un arbore binar este ABC daca pentru orive varf v, valorile memorate in subraborele sau stang sunt mai mici decat v, iar cele memorate in subarborele sau drept sunt mai mare decat v.

Exemple:
Inserarea intr-un arbore binar de cautare, intial vid, a valorilor: 5, 3, 7, 8, 10, 14.



parcurgerea inordine: 3, 5, 7, 8, 10, 14 - obtinem un sir sortat crescator!

```
struct nod {
        int inf
        nod * stg
        nod * drp
}
```
ArbBinCautare alias pentru nod *.

```
function cauta ( nod * t, int a )
begin
        if (t==NUL) then return false
        else if (t->inf  == a) then return true
                else if (a < t->inf) then return cauta(t->stg, a)
                                else return cauta(t->drp, a)
end
```

ABC: Cautarea are complexitatea O(h), unde h este inaltimea arborelui.

Pr. 2 Se da un arbore binar implementat cu structuri inlantuite si care contine ca informatie utila in nodurile sale numere intregi. Scrieti o functie care verifica daca acest arbore este arbore binar de cautare.

Input: nod *t - arbore binar oarecare
Output: true – daca t este ABC
        false – c.c.

\\ TEMA
Idee: parcurgem t inordine si verificam faptul ca obtinem un sir sortat!
! fara a folosi un spatiu de memorie suplimentar proportional cu n

Pr. 3 Sa se determite cea mai mare valoare dintr-un ABC.

```
function maxim(nod * t) \\ O(n) = O(h), Ω(1) – fiul dreapta al radacinii lispeste
begin
        if t == NULL then throw “eroare – arbore vid”
        while (t->drp != NULL) do t <-  t->drp
        return t->inf
end
```

Pr. 4 Sa se scrie un subprogramcare determina elementele k dintr-un arbore binar de cautare cu proprietatea : k1 <= k <= k2, unde k1 si k2 sunt parametri de intrare ai subprogramului. Care este complexitatea subprogramului
\\ TEMA