ASD - Heapsort (II)

1 Sortarea prin selectie folosind structuri arborescente (Sortarea cu ansamble/HeapSort)

1.1 Arbori partial ordonati si ansamble

Def Se numeste **arbore partial max-ordonat** un arbore binar cu chei de un tip total ordonat si cu proprietatea ca in orice nod **u** al sau avem relatiile:

$$\begin{cases} info[u] > info[root(left[u])], \text{ daca } left[u] \text{ este nevid} \\ info[u] > info[root(right[u])], \text{ daca } right[u] \text{ este nevid} \end{cases}$$

Pentru $\forall$ nod u: $\begin{cases} info[u] > info[v], \forall v \in left[u] \\ info[u] > info[w], \forall w \in right[u] \end{cases}$

$\Rightarrow$ **cheia maxima se va afla in radacina**

Conceptul de *arbore partial min-ordonat* **se defineste analog.**

Def: Arbore binar **complet pe niveluri** este un a.b. cu toate nivelurile pline, eventual cu exceptia ultimului nivel, unde toate nodurile vor fi aliniate cel mai la stanga.

Acest tip de arbore binar se poate reprezenta ca vector (deci alocare statica si acces in timp 1 la fii si la tata).

Def: Se numeste ansamblu (max-ansamblu) un arbore binar max-ordonat si complet pe niveluri, reprezentat ca vector.

Conceptul de *min-ansamblu* **se defineste analog.**

1.2 Inserarea intr-un ansamblu

Se urmeaza pasii:

1. Se pune nodul de inserat **(nod)** pe ultimul nivel al arborelui, aliniat cel mai la stanga. (*arborele ramane complet*).
2. Se repeta (eventual pana la radacina) comparatia intre **info[nod]** si **info[tata[nod]]**
   (a) Daca $info[nod] < info[tata[nod]]$ atunci am gasit locul lui nod in ansamblu (noua cheie nu violeaza conditia de arbore max-ordonat)
   (b) Daca nu, interschimb nod cu tata[nod] si reluam de la (a).

procedure InsHeap(Ans, n, Val) *in ansamblul Ans[1...n] se insereaza Val*
    $n \leftarrow n + 1$ *creste dimensiunea ansamblului cu 1*
    $p \leftarrow n$ *p = indice pentru nodul curent*
    While $p > 1$ **do** *cat timp nu am ajuns la radacina*
        $Tata \leftarrow p$ div 2
        If $Val \leq Ans[Tata]$ **then**
            $Ans[p] \leftarrow Val$ *se insereaza - cazul (a)*
            exit *am terminat*
        Else
            $Ans[p] \leftarrow Ans[Tata]$ *se coboara tatal in locul fiului*
            $p \leftarrow Tata$ *nodul curent se reactualizeaza*
        endIf
    endWhile
    $Ans[1] \leftarrow Val$ *inserarea in radacina - cazul (b)*
endproc

1.3 Construirea unui ansamblu. Asamblarea

Se realizeaza prin inserari repetate: daca avem cel putin o cheie, atunci ea se va insera in radacina (un arbore cu un nod este ansamblu).
- pt fiecare valoare noua se foloseste algoritmul de inserare **InsHeap**.

Asamblarea, operatie specifica, este construirea unui ansamblu din inserari repetate de chei care se afla deja in locatiile unui vector:
- A[1] este ansamblu
- la fiecare pas iterativ j se apeleaza procedura de inserare in ansamblul A[1..j] a valorii A[j+1], pentru j=$1, \cdots, n-1$.

procedure Asamblare (A,n)
    For $j \leftarrow 1, n-1$ **do**
        InsHeap(A,j,A[j+1])
    endFor
endproc

1.4 Extragerea maximului sau decapitarea unui ansamblu

1. Se extrage valoarea din radacina in vederea procesarii;
2. Se inlocuieste radacina cu ultimul nod (arborele binar ramane complet, dar eventual nu mai e max-ordonat)
3. Coboram noua radacina la locul ei prin comparatii cu cel mai mare dintre fii.

procedure DelHeap (Ans, n, Val)
$Val \leftarrow Ans[1]$ *extragerea radacinii*
$Last \leftarrow Ans[n]$ *Last e ultimul elem din ans ce tb inserat in Radacina, apoi tb sa cautam locul lui*
$n \leftarrow n - 1$ *scade dimensiunea ansamblului*

$p \leftarrow 1; l \leftarrow 2; r \leftarrow 3$ *p indice nod curent, l - fiu stang,r - fiu drept*
While $(r \leq n)$ **do** *test de nedepasire a structurii*
If$(Last \geq Ans[l])$ **AND** $(Last \geq Ans[r])$ **then**
$Ans[p] \leftarrow Last$ *inserarea*
exit *am terminat*
endIf

If $Ans[l] \geq Ans[r]$ **then** *continuam pe ramura stanga*
$Ans[p] \leftarrow Ans[l]$
$p \leftarrow l$ *reactualizarea lui p*
Else *continuam pe ramura dreapta*
$Ans[p] \leftarrow Ans[r]$
$p \leftarrow r$ *reactualizarea lui p*
endIf

$l \leftarrow 2 * p; r \leftarrow l + 1;$ *actualizarea fiilor lui p*
endWhile

If (l=n) **AND** $(Last < Ans[l])$ **then**
$Ans[p] \leftarrow Ans[l]$
$p \leftarrow l$
endIf
$Ans[p] \leftarrow Last$ *inserarea propriu-zisa a lui Last la locul lui*
endproc

1.5 Complexitatea operatiilor intr-un ansamblu

Sortarea cu ansamble

1. (Pas 0) Se asambleaza vectorul A[1..n]. Maximul va fi pe A[1];
2. (Pas 1) Se decapiteaza ansamblul A[1..n], cu reasamblarea lui A[1..n-1] si se pune maximul pe A[n];
3. (Pas j) Se decapiteaza ansamblul A[1..n-j+1], cu reasamblarea lui A[1..n-j] si se pune maximul pe A[n-j+1].

Dupa n-1 pasi iterativi vectorul A este sortat.

procedure HeapSort (A,n) *sortam vectorul A[1..n]*
    Asamblare (A,n)
    While $n > 1$ **do**
        DelHeap (A,n,Val)
            $A[n+1] \leftarrow Val$
    endWhile
endproc

Complexitate: $O(n \lg n)$.