**Tema seminar III**

**4. Numarul minim de parcurgeri pentru un ABC astfel incat sa se poata reconstituirea structura sa.**

**-> R: este suficient de o singura parcurgere : postorder … de ce?**

Observam ca postorder va pastra pe ultima pozitie din vectorul corespunzator parcugerii **radacina arborelui** **binar de cautare** .

**🡪** Faptul ca arborele e ABC ne ajuta sa determinam mai intai subarborele drept ( partitionam de fapt sirul in elemente mai mari (in dreapta unui index) si mai mici (in stanga indexului) decat radacina : **de la dreapta la stanga toate elementele >= decat radacina** (sa ii spunem pozitiei pe care se afla radacina in array **right**). Cum procedam in continuare ? Putem considera si subarborele un alt arbore cu radacina sa cea mai apropiata de radacina initiala ( postArray[right-1] ) . Iarasi partitionam. Pana cand?

* Cand nu mai avem elemente mai mari decat radacina, mergem pe partitia din stanga pivotului . Stim sigur ca toate elementele din acea partitie formeaza subarborele stang . Dar care e ordinea? Ca si in cazul precedent, consideram subsirul stang un sir ce are radacina postArray[pivot-1]

Pseudocod

structuraABCcuPostOrder(postArray, left, right, parent)

|

|if left <= right then

||

|| output(postArray(right) este fiul lui parent)

||

|| j <- right

|| while postArray(j-1) >= postArray(right)

|| | j <- j - 1

|| |\_\_

||

|| rightChild <- structuraABCcuPostOrder(postArray, j, right - 1,postArray(right))

|| leftChild <- structuraABCcuPostOrder(postArray, left, j - 1, postArray(right))

||

|| output( postArray(right) -> parinte , leftChild , rightChild)

||

|| return postArray(right)

||

||\_\_\_ else return -1

|

|\_\_\_\_\_\_\_\_\_

Cod in C

--- de spoate liniar cu left inainte de right!

int structuraABCcuPostOrdine(int\* post, int left, int right, int parent)

{

if (left <= right)

{

printf("%d este copilul lui %d\n", post[right], parent);

int j = right;

while (post[j - 1] >= post[right])

{

j--;

}

// de la left la j-1 subarbore stang pt nodul post(right) si de la j la right-1 subarbore drept pt nodul post(right)

int rightChild, leftChild;

//merg pe subarborele drept deci de la j la right - 1

rightChild = structuraABCcuPostOrdine(post, j, right - 1, post[right]);

// am terminat cu subarborele drept.. ma duc pe cel din stanga

leftChild = structuraABCcuPostOrdine(post, left, j - 1, post[right]);

printf("%d(parent) -> %d (left child) , %d (right child)\n", post[right], leftChild, rightChild);

return post[right];

}

else

{

return -1;

}

}

**Problema 2. B .Diametrul unui arbore binar implementat eficient**

HD diametru(tree\* t)

{

if (t == NULL )

{

return makeHD(-1, -1);

}

else {

printf("%d ", t->key);

HD L, R;

L = diametru(t->left);

R = diametru(t->right);

t->hd.h = max(L.h, R.h) + 1;

t->hd.d = max(L.h + R.h + 2, max(R.d, L.d));

return t->hd;

}

return makeHD(0, 0);

}

* Nu retin in t

**Unde:**

HD makeHD(int h, int d)

{

HD x;

x.h = h;

x.d = d;

return x;

}

tree\* createNode(int key)

{

tree\* node = (tree\*)malloc(sizeof(tree));

node->key = key;

node->left = node->right = NULL;

node->hd = makeHD(0, 0);

return node;

}

**Complexitate**? La fel ca si la inaltimea arborelui:

Running time c=0,a=2,b=2

Arbore exchilibrat : T(n) = 2T(n/2) + O(1) = (MT) = O( n ^(logb a) ) = O(n)

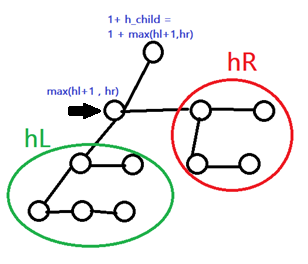
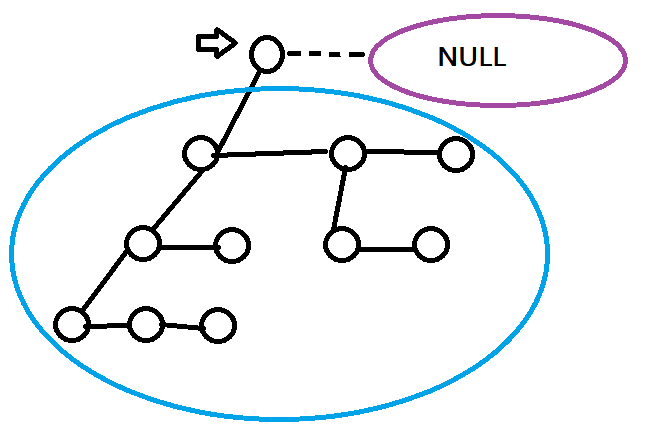
Worst = Arbore ce are doar fiu drept : T(n) = O(1)(stg) + T(n-1) + O(1) = .. = nO(1) =O(n)

Complexitatea algoritmului de la seminar **este O(nlog b ) average** si pentru **WORST este O(n^2)** (pentru ca inaltimea arborelui este n , si operatiile pentru determinarea inaltimii au complexitate O(n) )

**Probelma 2. C) H multi-way trees =>**

h = { max(hL + 1, hR) + 1, tree != NULL

-1, tree = NULL }



**OBSERVATIE**: subarborele stang e de fapt subarborele in care se afla copii nodului, iar cel drept reprezinta subarborele in care se afla fratii acestuia. => radacina nu are subarbore drept si inaltimea reprezinta defapt max(1+hLeft , - 1) = max(1+hLeft) la care adaugam 1 ( difera cu un nivel)

Algoritmul “aduce”( la nivel recursiv ) pe pozitia fiului din stanga inaltimea maxima obtinuta pe acel nivel.

**Pseudocod**

inaltimeMC(Mwtree\* t)

inaltimeMC( T )

|

| if T = NIL then

| |

| |\_\_\_\_ return -1;

|

| else

| |\_\_\_\_ return maxim( inaltimeMC(T->left) + 1, InaltimeMC(T->right))

|\_\_\_\_

**Problema 2. D) diameter MultiWay trees**

* Diametrul reprezinta fie suma celor mai mari 2 inaltimi (+2 🡪 cele 2 muchii ce leaga nodul de cei 2 subarbori) de pe **urmatorul nivel** de la radacina ( deci din subarborele stang), fie cel mai mare diametru obtinut pentru noduri interne ( retinut fie in subarborele drept, fie in cel stang)

🡪 la parcurgerea arborelui cel mai din stanga fiu va retine diametrul si inaltimea maxime obtinute din subarborele stang iar in subarborele drept ( fratele nodului curent ) valorile maxime obtinute atat de el, cat si de restul fratilor (deci nu e nevoie sa cautam printre toti fratii, sunt suficienti primii 2)

**Relatie de recurenta:**

**diamMC(t) = { max(height(t->left) + height(t->left->right) + 2, diamLeft, diamRight), T !=NULL**

**-1, T = NULL }**

