Ministerul Educaţiei, Culturii și Cercetării

al Republicii Moldova   
  
Universitatea Tehnică a Moldovei   
  
  
  
  
  
  
  
RAPORT

Lucrarea de laborator nr. 4   
la Structuri de date si algoritmi

A efectuat: st. gr. TI-211 Popa Cătălin   
  
A verificat: Stratulat Ștefan

UTM, Chișinău 2021

**Tema:**

Implementarea tipului de date abstract “Arbore binar de căutare” în limbajul C.

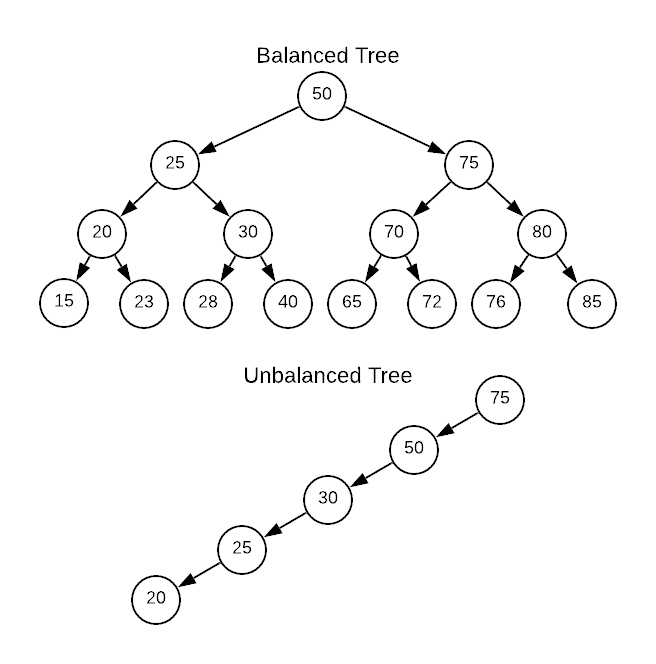
**Scopul lucrării**

Scopul lucrării este de a familiariza studentul cu mecanismul de creare a arborelui binar de căutare și operații elementare asupra acestuia, utilizînd pentru aceasta limbajul C.

**Problema**

1. **Să se elaboreze un program ce va aloca dinamic un arbore de căutare binar de structuri (unde structurile vor fi cu mai multe cîmpuri, minim 4 cîmpuri, dintre care un cîmp pentru cheie numit key) și va realiza următoarele funcții, unde funcțiile date reprezintă opțiuni organizate într-un meniu în cadrul programului:**

* citirea de la tastatură a elementelor arborelui;
* afișarea la consolă a elementelor arborelui;
* căutarea unui nod în baza cîmpului cheie și afișarea cîmpurilor nodului găsit;
* parcurgerea arborelui în inordine;
* parcurgerea arborelui în preordine;
* parcurgerea arborelui în postordine;
* parcurgerea arborelui în adîncime( DFS);
* parcurgerea arborelui în lărgime ( BFS);
* balansarea arborelui (în imaginea de mai jos este redat un arbore balansat și unul nebalansat );



* oglindirea arborelui (orice nod copil drept, devine un nod copil stîng și analog orice nod copil stîng devine un nod copil drept), ține cont că după oglindirea arborelui binar de căutare proprietatea între nod și copii se va schimba, prin urmare și căutarea deja se va face în altă ordine;
* curățirea elementelor arborelui;
* eliberarea memoriei arborelui.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <time.h>

#define COUNT 10

//////////////////////////////////////////////////////////////////

typedef struct BinaryTreeNode

{

char \*marca;

char \*culoare;

int anul;

char \*tip\_tractiune;

int putere;

int key;

struct BinaryTreeNode \* left;

struct BinaryTreeNode \* right;

}BinaryTreeNode;

//////////////////////////////////////////////////////////////////

typedef struct BinaryTree

{

struct BinaryTreeNode \* root;

}BinaryTree;

//////////////////////////////////////////////////////////////////

BinaryTreeNode \* newTreeNode(char \*marca\_m, char \*culoare\_m,char \*type, int a,int p, int key)

{

BinaryTreeNode \* new\_node = calloc(sizeof(BinaryTreeNode),1);

new\_node->marca = calloc(sizeof(char),10);

strcpy(new\_node->marca, marca\_m);

new\_node->culoare = calloc(sizeof(char),10);

strcpy(new\_node->culoare, culoare\_m);

new\_node->tip\_tractiune = calloc(sizeof(char),10);

strcpy(new\_node->tip\_tractiune,type);

new\_node->anul = a;

new\_node->putere = p;

new\_node->key = key;

return new\_node;

}

//////////////////////////////////////////////////////////////////

struct BinaryTree \* create\_binary\_tree()

{

BinaryTree \* my\_binary = malloc(sizeof(struct BinaryTree));

my\_binary->root = NULL;

return my\_binary;

}

//////////////////////////////////////////////////////////////////

void insert\_new\_node(BinaryTreeNode \*current\_node, BinaryTreeNode \* new\_node )

{

if(current\_node)

{

if(current\_node->key > new\_node->key)

{

if(current\_node->left)

{

insert\_new\_node(current\_node->left,new\_node);

}

else

{

current\_node->left = new\_node;

}

}

else

{

if(current\_node->right)

{

insert\_new\_node(current\_node->right,new\_node);

}

else

{

current\_node->right = new\_node;

}

}

}

}

//////////////////////////////////////////////////////////////////

void insert\_data\_in\_binary\_tree(BinaryTree \*insert, BinaryTreeNode \*\*arr,int i)

{

char \*marca\_m, \*culoare\_m, \*type;

int a,puterea,key;

printf("Marca: \n");

marca\_m = calloc(sizeof(char),10);

scanf("%s", marca\_m);

printf("Culoarea masinii: \n");

culoare\_m = calloc(sizeof(char),10);

scanf("%s", culoare\_m);

printf("Tipul tractiunii: \n");

type = calloc(sizeof(char),10);

scanf("%s",type);

printf("Anul masinii\n");

scanf("%d", &a);

printf("Puterea masinii(cati caluti): \n");

scanf("%d", &puterea);

printf("Key: \n");

scanf("%d",&key);

BinaryTreeNode \*new\_node = newTreeNode(marca\_m, culoare\_m,type, a, puterea,key);

arr[i] = new\_node;

if(insert->root)

{

insert\_new\_node(insert->root, new\_node);

}

else

{

insert->root = new\_node;

}

}

//////////////////////////////////////////////////////////////////

//parcurgere in inordine (SVD)

void inorder(BinaryTreeNode \* nodes)

{

if (nodes != NULL)

{

inorder(nodes->left);

printf("\n");

printf("Key : |%d|\n", nodes->key);

printf("Marca : |%s|\n", nodes->marca);

printf("Culoare : |%s|\n", nodes->culoare);

printf("Tipul tractiunii : |%s|\n", nodes->tip\_tractiune);

printf("Anul : |%d|\n", nodes->anul);

printf("Puterea : |%d| (cai putere)\n", nodes->putere);

printf(".................................................\n");

inorder(nodes->right);

}

else

{

return;

}

}

//////////////////////////////////////////////////////////////////

//parcurgere in preordine (VSD)

void preorder(BinaryTreeNode \* nodes)

{

if(nodes != NULL)

{

printf("|%d|",nodes->key);

printf("Key : |%d|\n", nodes->key);

printf("Marca : |%s|\n", nodes->marca);

printf("Culoare : |%s|\n", nodes->culoare);

printf("Tipul tractiunii : |%s|\n", nodes->tip\_tractiune);

printf("Anul : |%d|\n", nodes->anul);

printf("Puterea : |%d| (cai putere)\n", nodes->putere);

printf(".................................................\n");

preorder(nodes->left);

preorder(nodes->right);

}

else

{

return;

}

}

//////////////////////////////////////////////////////////////////

//parcurgere in postordine (SDV)

void postorder(BinaryTreeNode \* nodes)

{

if(nodes != NULL)

{

postorder(nodes->left);

postorder(nodes->right);

printf("Key : |%d|\n", nodes->key);

printf("Marca : |%s|\n", nodes->marca);

printf("Culoare : |%s|\n", nodes->culoare);

printf("Tipul tractiunii : |%s|\n", nodes->tip\_tractiune);

printf("Anul : |%d|\n", nodes->anul);

printf("Puterea : |%d| (cai putere)\n", nodes->putere);

printf(".................................................\n");

}

else

{

return;

}

}

//////////////////////////////////////////////////////////////////

void Tree2D(BinaryTreeNode \* node, int space)

{

if (node == NULL)

return;

space += COUNT;

Tree2D(node->right, space);

printf("\n");

for (int i = COUNT; i < space; i++)

printf(" ");

printf("%d[%s|%d]\n", node->key, node->marca, node->anul);

Tree2D(node->left, space);

}

void print2(BinaryTreeNode \*node)

{

Tree2D(node, 0);

}

//////////////////////////////////////////////////////////////////

void DeleteNode(BinaryTreeNode \*node)

{

if(node)

{

DeleteNode(node->left);

DeleteNode(node->right);

free(node);

}

}

//////////////////////////////////////////////////////////////////

void clear\_tree(BinaryTree \*tree)

{

if(tree)

{

if(tree->root)

{

DeleteNode(tree->root);

tree->root = NULL;

}

}

}

//////////////////////////////////////////////////////////////////

void DeleteTree(BinaryTree \*tree)

{

if(tree)

{

clear\_tree(tree);

free(tree);

}

}

//////////////////////////////////////////////////////////////////

BinaryTreeNode \*search(BinaryTreeNode \*nodes, int cauta, int kek)

{

if (nodes == NULL)

{

printf("!Nu exista element cu cheia respectiva!\n");

return NULL;

}

if(nodes->key == cauta)

{

printf(".................................................\n");

printf("Key : |%d|\n", nodes->key);

printf("Marca : |%s|\n", nodes->marca);

printf("Culoare : |%s|\n", nodes->culoare);

printf("Tipul tractiunii : |%s|\n", nodes->tip\_tractiune);

printf("Anul : |%d|\n", nodes->anul);

printf("Puterea : |%d| (cai putere)\n", nodes->putere);

printf(".................................................\n");

return nodes;

}

if(kek%2 == 0)

{

if(nodes && cauta < nodes->key)

{

search(nodes->left,cauta,kek);

}

if(nodes && cauta > nodes->key)

{

search(nodes->right,cauta,kek);

}

}

else if(kek%2 !=0)

{

if(nodes && cauta < nodes->key)

{

search(nodes->right,cauta,kek);

}

if(nodes && cauta > nodes->key)

{

search(nodes->left,cauta,kek);

}

}

}

//////////////////////////////////////////////////////////////////

void mirror(BinaryTreeNode\* node)

{

if (node == NULL)

return;

else

{

BinaryTreeNode\* temp;

mirror(node->left);

mirror(node->right);

temp = node->left;

node->left = node->right;

node->right = temp;

}

}

//////////////////////////////////////////////////////////////////

void DFS(BinaryTreeNode \*node, int key\_search)

{

if(node == NULL)

return;

if(node->key == key\_search)

{

printf(".................................................\n");

printf("Key : |%d|\n", node->key);

printf("Marca : |%s|\n", node->marca);

printf("Culoare : |%s|\n", node->culoare);

printf("Tipul tractiunii : |%s|\n", node->tip\_tractiune);

printf("Anul : |%d|\n", node->anul);

printf("Puterea : |%d| (cai putere)\n", node->putere);

printf(".................................................\n");

}

else

{

DFS(node->left, key\_search);

DFS(node->right, key\_search);

}

}

//////////////////////////////////////////////////////////////////

void current\_Level(BinaryTreeNode \*node, int level);

int height(BinaryTreeNode \*node);

//////////////////////////////////////////////////////////////////

void level\_Order(BinaryTreeNode \*node)

{

int h = height(node);

int i;

for (i = 1; i <= h; i++)

current\_Level(node, i);

}

//////////////////////////////////////////////////////////////////

void current\_Level(BinaryTreeNode \*node, int level)

{

if (node == NULL)

return;

if (level == 1)

{

printf(".................................................\n");

printf("Key : |%d|\n", node->key);

printf("Marca : |%s|\n", node->marca);

printf("Culoare : |%s|\n", node->culoare);

printf("Tipul tractiunii : |%s|\n", node->tip\_tractiune);

printf("Anul : |%d|\n", node->anul);

printf("Puterea : |%d| (cai putere)\n", node->putere);

printf(".................................................\n");

}

else if (level > 1) {

current\_Level(node->left, level - 1);

current\_Level(node->right, level - 1);

}

}

//////////////////////////////////////////////////////////////////

int height(BinaryTreeNode \*node)

{

if (node == NULL)

return 0;

else {

/\* compute the height of each subtree \*/

int lheight = height(node->left);

int rheight = height(node->right);

/\* use the larger one \*/

if (lheight > rheight)

return (lheight + 1);

else

return (rheight + 1);

}

}

//////////////////////////////////////////////////////////////////

void sortArray(BinaryTreeNode \*\* arr, int size)

{

BinaryTreeNode \* temp;

for(int i=0;i<size;i++)

{

arr[i]->left = NULL;

arr[i]->right = NULL;

}

for(int i=0;i<size;i++)

{

for(int j=i+1;j<size;j++)

{

if(arr[i]->key>arr[j]->key)

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

}

BinaryTreeNode \* balanceTreeHelper(BinaryTreeNode \*\* arr, int start, int end)

{

int check = 0;

if(start<end){

int mid = (start + end)/2;

BinaryTreeNode \* node = arr[mid];

if(check==0)

{

node->left = balanceTreeHelper(arr, start, mid);

node->right = balanceTreeHelper(arr, mid+1, end);

}

return node;

}

else

{

return NULL;

}

}

void balanceTree(BinaryTree \* tree, BinaryTreeNode \*\* arr, int size)

{

int start = 0, end = size;

tree->root = balanceTreeHelper(arr, start, end);

}

int menu()

{

int error\_input\_found = 0;

menu\_start:

system("cls");

printf("|--------------------------------------|\n");

printf("| A efectuat: Popa Catalin |\n");

printf("| Grupa: TI-211 |\n");

printf("| A verificat: Stratulat Stefan |\n");

printf("|--------------------------------------|\n");

printf("|----------------------//Menu\\--------------------------|\n");

printf("|1.Creaza arborele |\n");

printf("|2.Afisarea elementelor |\n");

printf("|3.Cautarea unui nod in baza campului cheie + afisarea |\n");

printf("|4.Parcurgerea in inordine |\n");

printf("|5.Parcurgerea in preordine |\n");

printf("|6.Parcurgerea in postordine |\n");

printf("|7.Parcurgerea in adancime (DFS) |\n");

printf("|8.Parcurgerea in largime (BFS) |\n");

printf("|9.Balansarea arborelui |\n");

printf("|10.Oglindirea arborelui |\n");

printf("|11.Curatarea elementel arborelui |\n");

printf("|12.Eliberarea memoriei arborelui |\n");

printf("|-------------------------------------------------------|\n");

int command = 100;

if(error\_input\_found)

{

printf("\n\nCommanda trebuie sa fie o optiune de mai sus!!![0-12]\n");

}

printf("\n\nSelectati commanda: ");

scanf("%d",&command);

if((command>12)||(command<0))

{

error\_input\_found = 1;

goto menu\_start;

}

return command;

}

//////////////////////////////////////////////////////////////////

int main()

{

int num,key,kek=0;

BinaryTree \*tree = NULL;

BinaryTreeNode \*\*array = NULL;

int command=100;

while(command)

{

command = menu();

if(command == 1)

{

if(tree != NULL)

{

DeleteTree(tree);

}

tree = create\_binary\_tree();

//tree\_2 = create\_binary\_tree();

printf("Dati numarul de masini: \n");

scanf("%d",&num);

array = calloc(num, sizeof(BinaryTreeNode\*));

for(int i=0; i<num; i++)

{

printf("Masina: [%d]\n", i);

insert\_data\_in\_binary\_tree(tree,array,i);

printf("\n");

}

}

switch (command)

{

case 2:

{

if(tree == NULL)

{

printf("Arborele este null\n");

break;

}

else

{

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

print2(tree->root);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

}

}

case 3:

{

if(tree == NULL)

{

printf("Arborele este null\n");

}

else

{

printf("Introduceti cheia cautata: \n");

scanf("%d", &key);

search(tree->root,key,kek);

}

break;

}

case 4:

{

inorder(tree->root);

break;

}

case 5:

{

preorder(tree->root);

break;

}

case 6:

{

postorder(tree->root);

break;

}

case 7:

{

printf("Introduceti cheia:\n");

scanf("%d", &key);

clock\_t start = clock();

DFS(tree->root,key);

clock\_t end = clock();

float time\_t = (float)(end-start)/CLOCKS\_PER\_SEC;

printf("%f",time\_t);

break;

}

case 8:

{

clock\_t start = clock();

level\_Order(tree->root);

clock\_t end = clock();

float time\_t = (float)(end-start)/CLOCKS\_PER\_SEC;

printf("%f",time\_t);

break;

}

case 9:

{

sortArray(array,num);

balanceTree(tree,array,num);

break;

}

case 10:

{

if(tree == NULL)

{

printf("Arborele este null!\n");

}

else

{

mirror(tree->root);

kek++;

printf("Oglindirea a avut loc cu succes!\n");

}

break;

}

case 11:

{

clear\_tree(tree);

printf("Elementele arborelui au fost curatite\n");

break;

}

case 12:

{

DeleteTree(tree);

tree = NULL;

printf("\nMemoria a fost eliberata cu succes!");

printf("\nArborele a fost sters\n");

break;

}

}

printf("\n\nTastati orice buton pentru continuare!\n");

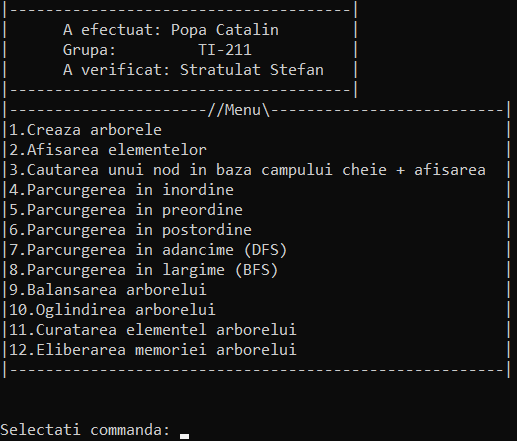
\_getch();

}

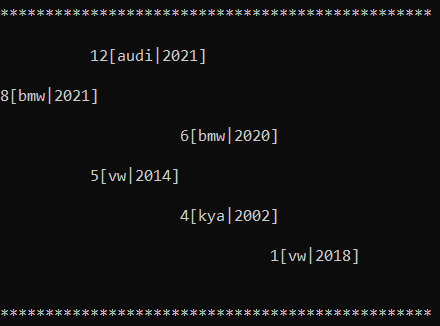
return 0;

}

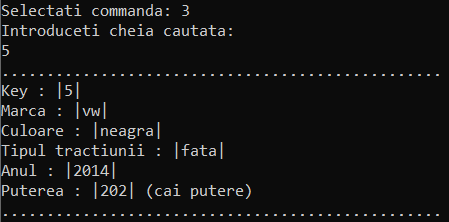
**Menu**:



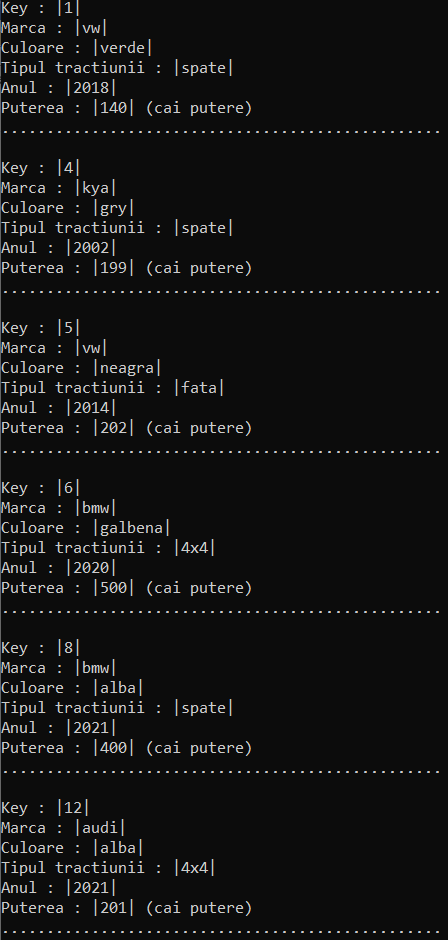
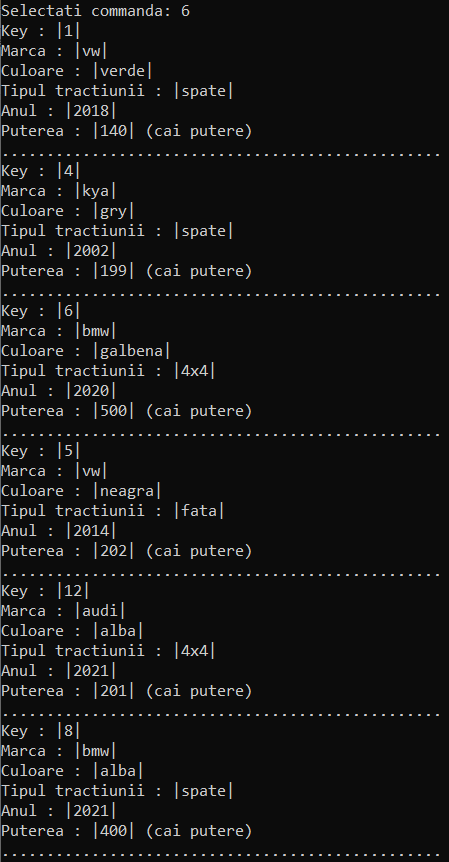
**Afisarea**:



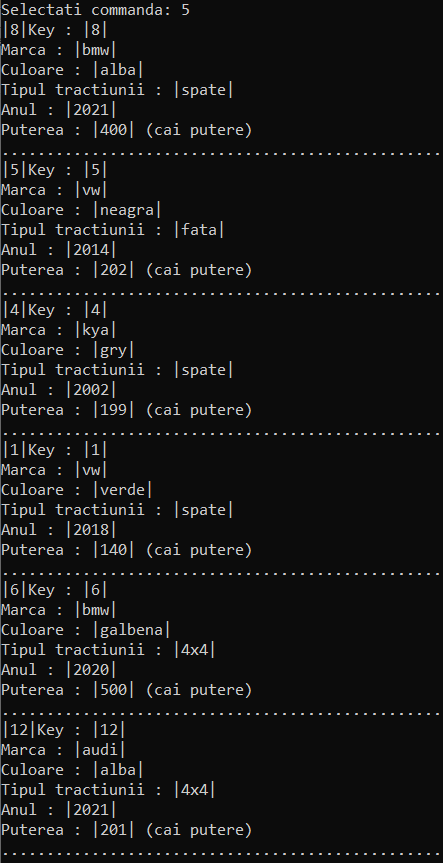
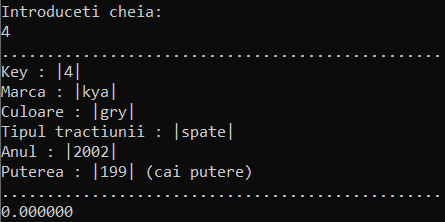
**Cauta uni nod**:



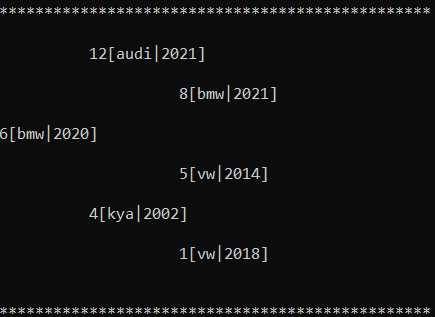
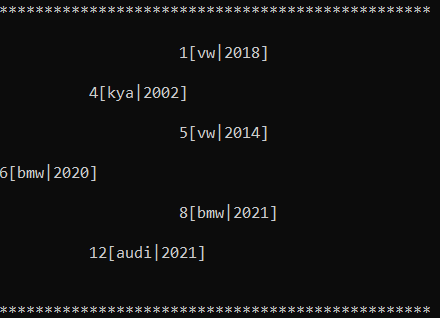
**Parcurgerea in inordine**: **Parcurgerea in postordine**:

**Parcurgere in preordine**: **DFS**:



**Balansarea**: **Oglindirea**:

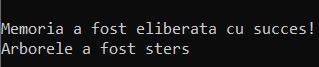
 

**Curatarea elementelor**:



**Afisarea dupa curatirea elementelor**:

**Eliberarea memoriei**:



**Afisarea dupa eliberarea memoriei**:



**BFS:**

