Ministerul Educaţiei, Culturii și Cercetării

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Departamentul Ingineria Software și Automatică

RAPORT

La structuri de date și algoritmi

Lucrarea de laborator nr. 6

Varianta 18

A efectuat:

st. gr. TI-206 Cătălin Pleșu

A verificat:

Lector universitar Vitalie Mititelu

Chișinău 2021

Tema:
Algoritmi de prelucrare a tipului abstract de date „Arbori binari”

Scopul:
Obținerea deprinderilor practice de implementare și de utilizare a tipului abstract de date (TAD) „Arbore binar” cu asigurarea operațiilor de prelucrare de bază ale arborelui binar oarecare prin parcurgerea recursivă a nodurilor arborelui folosind algoritmi recursivi sau structurile respective de date „coadă” și „stivă”.

Sarcina:
Să se scrie 3 fișiere-text în limbajul C pentru implementarea și utilizarea TAD „Arbore binar” cu asigurarea operațiilor de prelucrare de bază ale arborilor binari oarecareprin parcurgerea nodurilor arborelui cu ajutorul algoritmilor recursivi sau iterativi:

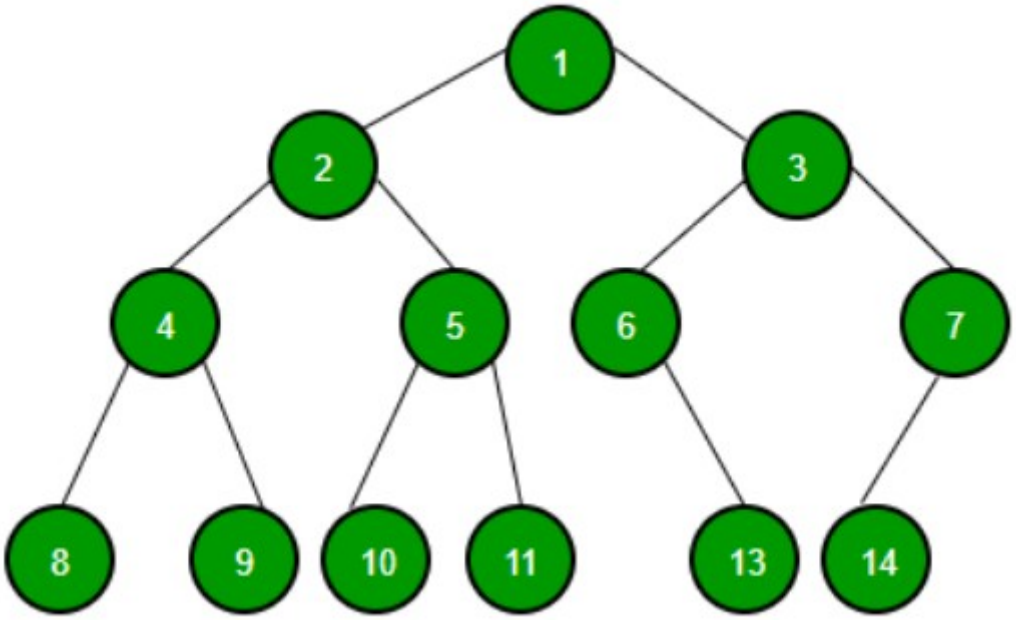
1. Fișier antet cu extensia .h, care conține specificarea structurii de date a nodului arborelui binar (conform variantelor) și prototipurile funcțiilor de prelucrare de bază ale arborilor binari.
2. Fișier cu extensia .c sau .cpp, care conține implementările (codurile) funcțiilor declarate în fișierul antet.
3. Fișier al utilizatorului, funcția main() pentru prelucrarea arborelui binar oarecare cu afișarea la ecran a următorului meniu de opțiuni de bază:

    1. Crearea nodurilor arborelui binar oarecare în memoria dinamică și introducerea informației despre nodurile arborelui de la tastatură în mod interactiv.
    2. Afișarea informației despre nodurile arborelui la ecran.
    3. Căutarea nodului în arbore.
    4. Modificarea informației unui nod din arbore.
    5. Determinarea numărului de noduri.
    6. Determinarea înălțimii arborelui.
    7. Eliberarea memoriei alocate pentru listă.

Structura Imobil cu câmpurile: **proprietarul**, **tipul**, **adresa**, **suprafaţa**, **costul**.

Rezumat la temă:

Un arbore al cărui elemente au cel mult 2 copii se numește arbore binar. Deoarece fiecare element dintr-un arbore binar poate avea doar 2 copii, de obicei îi denumim copilul stâng și drept.



Un nod al arborelui binar conține:

1. informația
2. pointer la stana
3. pointer la dreapta

Exista 3 metode principale de parcurgere a unui arbore și anume:

1) preordine ;
2) inordine ;
3) postordine.

Traversarea în preordine e acea, în care mai întâi se prelucrează informația din nod și apoi se parcurge prin arbore.
Traversarea în inordine e acea, în care secvență acțiunilor e următoarea: parcurgerea subarborelui stâng, vizitarea nodului rădăcină și apoi parcurgerea subarborelui drept.
Traversarea în postordine cu parcurgerea subarborelui stâng, subarborelui drept și apoi vizitând rădăcină.

Structura de arbore binar poate fi utilizată pentru a reprezenta în mod convenabil o mulțime de elemente, în care elementele se regăsesc după o cheie unică. Deci nu este foarte un TAD foarte potrivit pentru structura dată.

Cod sursă:

const.h

```
#ifndef HERMINA
#define HERMINA

const char NAME[][30] = {"Catalin", "Marius", "Daniel", "Mirela", "Alex", "Colea", "Sandu", "Ion B", "Maximo", "Melissa", "Petru", "Stas", "Vlad", "Crstian", "Ion T", "Mihail", "Victor", "Vladislav", "Maria", "Vitalie", "Nicoleta", "Sam", "Nicu", "Viorel"};
const char TYPE[][30] = {"Apartament", "Birou", "Fabrica", "Magazin", "Mol", "Hotel", "Cladire Istorica", "Teren gol", "Restaurant"};
const char ADDRESS[][30] = {"Strada Albisoara", "Strada Alexandru Bernardazzi", "Strada Alexandru cel Bun", "Strada Alexei Mateevici", "Strada Armeneasca", "Strada Bucuresti", "Strada Calea Iesilor", "Strada Mihail Kogalniceanu"};
const int NAME_COUNT = 24, TYPE_COUNT = 9, ADDRESS_COUNT = 8;

#endif
```

bin_tree.h

```
#ifndef MIRELA
#define MIRELA

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <math.h>

void bin_test();
typedef struct
{
    char *owner, *type, *address;
    int surface,price;
}realty;
typedef struct node
{
    realty immovable;
    struct node *left, *right;
}tree;

typedef struct list
{
    tree* node;
    int level;
```

```
    struct list* next;
}list;


tree* forge_node( realty immovable);
void print_node(tree* node);
void grow_tree(tree** root,realty immovable);
void inorder(tree* root);
void preorder(tree* root);
void postorder(tree* root);
tree* search_node(tree* node, int key);
void modify_node(tree* node);
void append_to_queue(tree* node, int level, list** tail);
void levels(list* head, list** tail);
int print_levels(tree* root);
int menu();
int count_list_nodes(list*head);
void free_list(list** head);
int isnt_tree_root(tree* root);
void get_string(char** var,const char *message);
realty get_realty(int order);
void put_realty(realty immovable);
realty generate_realty(int order);
void postorder_free(tree** node);
void preorder_grow(tree** root, tree* node);
void postorder_free_lite(tree** node);
#endif
```

bin_tree.cpp

```
#include "bin_tree.h"
#include "const.h"

void bin_test()
{
    printf("__bin_tree__\n");
}

tree *forge_node(realty immovable)
{
    tree *node = (tree *)malloc(sizeof(tree));
    node->immovable = immovable;

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

```
    return node;
}

void print_node(tree *node)
{
    //printf("\n");
    //printf("parent: %p\n",node->parent);
    //printf("adress: %p\n",node);
    put_realty(node->immovable);
    //printf("left: %p\n",node->left);
    //printf("right: %p\n",node->right);
}

void grow_tree(tree **root, realty immovable)
{
    if ((*root) == NULL)
    {
        (*root) = forge_node(immovable);
        return;
    }

    tree *temp = (*root);
    while (1)
    {
        if (immovable.price <= temp->immovable.price)
        {
            if (temp->left == NULL)
            {
                temp->left = forge_node( immovable);
                break;
            }
            else
                temp = temp->left;
        }
        if (immovable.price > temp->immovable.price)
        {
            if (temp->right == NULL)
            {
                temp->right = forge_node(immovable);
                break;
            }
            else
                temp = temp->right;
        }
```

```
    }
}

void inorder(tree *node)
{
    if (!node)
        return;
    inorder(node->left);
    print_node(node);
    inorder(node->right);
}

void preorder(tree *node)
{
    if (!node)
        return;
    print_node(node);
    preorder(node->left);
    preorder(node->right);
}

void postorder(tree *node)
{
    if (!node)
        return;
    postorder(node->left);
    postorder(node->right);
    print_node(node);
}

tree *search_node(tree *node, int key)
{
    if (!node)
        return NULL;
    if (node->immovable.price == key)
        return node;
    if (node->left)
        if(node->left->immovable.price == key)
            return node;
    if (node->right)
        if (node->right->immovable.price == key)
            return node;

    if (node->immovable.price >= key)
        return search_node(node->left, key);
```

```
    else
        return search_node(node->right, key);
}

void modify_node(tree *node)
{
    char c;
    printf("modifica proprietarul? (%s) [y/n]\n",node->immovable.owner);
    scanf(" %c", &c);
    if (c == 'y')
        get_string(&node->immovable.owner, "proprietar nou");

    printf("modifica tipul? (%s) [y/n]\n",node->immovable.type);
    scanf(" %c", &c);
    if (c == 'y')
        get_string(&node->immovable.type, "tipul");
    printf("modifica adresa? (%s) [y/n]\n",node->immovable.address);
    scanf(" %c", &c);
    if (c == 'y')
        get_string(&node->immovable.address, "adresa");
    printf("modifica suprafata? (%d) [y/n]\n",node->immovable.surface);
    scanf(" %c", &c);
    if (c == 'y'){
        printf("suprafata : ");
        scanf(" %d", &node->immovable.surface);}
    printf("modifica pretul? (%d) [y/n]\n",node->immovable.price);
    scanf(" %c", &c);
    if (c == 'y'){
    printf("costul : ");
    scanf(" %d", &node->immovable.price);}
}

void append_to_queue(tree *node, int level, list **tail)
{
    list *element = (list *)malloc(sizeof(list));
    element->node = node;
    element->level = level; element->next = NULL;
    if (!(*tail))
        (*tail) = element;
    else
    {
        (*tail)->next = element;
        (*tail) = element;
    }
}
```

```c
void levels(list *node, list **tail)
{
    if (!node)
        return;
    if (node->node->left)
        append_to_queue(node->node->left, node->level + 1, tail);
    if (node->node->right)
        append_to_queue(node->node->right, node->level + 1, tail);
    levels(node->next, tail);
}

int print_levels(tree *root)
{
    list *tail = NULL;
    append_to_queue(root, 0, &tail);
    list *head = tail;
    levels(head, &tail);

    int current_level = -1;
    while (head)
    {
        if (head->level != current_level)
        {
            printf("\nnivelul curent: %d\n", head->level);
            printf("numarul maxim de noduri posibile: %d\n",(int)
pow(2,head->level));
            current_level = head->level;
        }
        print_node(head->node);
        head = head->next;
    }
    printf("\n");
    return current_level;
}

int menu()
{   printf("\n");
    printf("%-20s %-15s\n","1. adauga nod","3. cautarea");
    printf("%-20s %-15s\n","11. genereaza nod","4. modificarea");
    printf("%-20s %-15s\n","Afisarea:","5. numarul de elemente");
    printf("%-20s %-15s\n","21. inordine","6. inaltimea arborelui");
    printf("%-20s %-15s\n","22. preordine","7. eliberarea");
    printf("%-20s %-15s\n","23. postordine","0. iesirea");
    printf("%-20s\n","2. pe nivele");
```

```
    int option;
    printf("Optiunea - ");
    scanf(" %d", &option);

    return option;
}

int count_list_nodes(list *head)
{
    int count = 0;
    while (head)
    {
        count++;
        head = head->next;
    }
    return count;
}

void free_list(list **head)
{
    list *next;
    while ((*head))
    {
        next = (*head)->next;
        free((*head));
        (*head)=NULL;
        (*head) = next;
    }
}

int isnt_tree_root(tree *root)
{
    if (!root)
    {
        printf("radacina arborelui este %p\n", root);
        return 1;
    }
    return 0;
}

void get_string(char **var, const char *message)
{
    char str[250];
    printf("%s:  ", message);
    scanf(" %[^\n]", str);
```

```
    (*var) = strdup(str);
}

realty get_realty(int order)
{
    printf("citirea imobilului %d\n", order);
    realty immovable;
    get_string(&immovable.owner, "proprietar");
    get_string(&immovable.type, "tipul");
    get_string(&immovable.address, "adresa");

    printf("suprafata : ");
    scanf(" %d", &immovable.surface);
    printf("costul : ");
    scanf(" %d", &immovable.price);

    return immovable;
}

void put_realty(realty immovable)
{
    printf("%-15s ", immovable.owner);
    printf("%-20s ", immovable.type);
    printf("%-30s ", immovable.address);
    printf("%9d m^2 ", immovable.surface);
    printf("%9d $\n", immovable.price);
}

realty generate_realty(int order)
{
    printf("imobilului %d, este generat:\n", order);
    realty immovable;

    immovable.owner = strdup(NAME[rand() % NAME_COUNT]);
    immovable.type = strdup(TYPE[rand() % TYPE_COUNT]);
    immovable.address = strdup(ADDRESS[rand() % ADDRESS_COUNT]);
    immovable.surface = (rand() % 100) + 16;
    immovable.price = immovable.surface * ((rand() % 1000) + 100);
    printf("proprietarul: %s\n", immovable.owner);
    printf("tipul: %s\n", immovable.type);
    printf("adresa: %s\n", immovable.address);
    printf("suprafata: %d m^2\n", immovable.surface);
    printf("costul: %d $\n", immovable.price);
    return immovable;
}
```

```c
void postorder_free(tree** node)
{
    if (!(*node))
        return;
    postorder_free(&(*node)->left);
    postorder_free(&(*node)->right);
    free((*node)->immovable.owner);
    free((*node)->immovable.type);
    free((*node)->immovable.address);
    free((*node));
    (*node)=NULL;
}

void preorder_grow(tree** root, tree *node)
{
    if (!node)
        return;
    grow_tree(root,node->immovable);
    preorder_grow(root,node->left);
    preorder_grow(root,node->right);
}

void postorder_free_lite(tree** node)
{
    if (!(*node))
        return;
    postorder_free_lite(&(*node)->left);
    postorder_free_lite(&(*node)->right);
    free((*node));
    (*node)=NULL;
}
```

main.c

```c
#include "bin_tree.h"

int main()
{
    srand(time(NULL));
    printf("__main__\n");
    bin_test();

    tree*root=NULL;
```

```
    list* tail = NULL;
    list* head = NULL;

    int order = 0;
    int option = 666;
    while(option)
    {
    option = menu();

    switch (option)
        {
        case 1:
        {
            realty temp = get_realty(++order);
            grow_tree(&root, temp);
            break;
        }
        case 11:
        {
            realty temp = generate_realty(++order);
            grow_tree(&root, temp);
            break;
        }
        case 21:
                if (isnt_tree_root(root))
                    break;
                printf("inordine\n");
                inorder(root);
                break;
        case 22:
                if (isnt_tree_root(root))
                    break;
                printf("preordine\n");
                preorder(root);
                break;
        case 23:
                if (isnt_tree_root(root))
                    break;
                printf("postordine\n");
                postorder(root);
                break;
        case 2:
                if (isnt_tree_root(root))
                    break;
                append_to_queue(root, 0, &tail);
```

```
            head = tail;
            levels(head, &tail);
            print_levels(root);
            printf("numarul de noduri: %d\n",count_list_nodes(head));
            break;
        case 3:
        {
           if (isnt_tree_root(root))
                 break;
            printf("elementul cautat (dupa pret)\n");
            int key;
            scanf("%d",&key);
            tree*found = search_node(root,key);
            if (!found){
                printf("nu exista asa nod\n");
                break;
            }

            if (found->left)
                if(found->left->immovable.price == key)
                    found = found->left;
            if (found->right)
                if (found->right->immovable.price == key)
                    found = found->right;
            while(found->immovable.price == key){
            put_realty(found->immovable);
            if (found->left==NULL)
                break;
            found=found->left;
            }
            break;
        }
        case 4:
        {
           if (isnt_tree_root(root))
                 break;
            printf("elementul cautat pentru modificare (dupa pret)\n");
            int key;
            scanf("%d",&key);
            tree*found = search_node(root,key);
            if (!found){
                printf("nu exista asa nod\n");
                break;
            }
```

```
        tree * sub_tree = NULL;
        if (found->immovable.price != key)
        {
        if (found->left)
            if(found->left->immovable.price == key)
            {
                sub_tree = found->left;
                found->left=NULL;
                found = sub_tree;
            }
        if (found->right)
            if (found->right->immovable.price == key)
            {
                sub_tree = found->right;
                found->right=NULL;
                found = sub_tree;
            }
        }
        else
        {
            sub_tree = found;
            root = NULL;
        }

        while(sub_tree->immovable.price == key)
        {
            modify_node(sub_tree);
            if (sub_tree->left==NULL)
                break;
            sub_tree=sub_tree->left;
        }
        preorder_grow(&root,found);
        postorder_free_lite(&found);
        break;
    }
    case 5:
        append_to_queue(root, 0, &tail);
        head = tail;
        levels(head, &tail);
        printf("numarul de noduri: %d\n",count_list_nodes(head));
        break;
    case 6:
       if (isnt_tree_root(root))
           break;
        printf("inaltimii arborelui este: %d\n",print_levels(root));
```

```
            break;
        case 7:
            free_list(&head);
            postorder_free(&root);
            order = 0;
            break;
        }
    }
    return 0;
}
```

makefile

```
compile:
    g++ main.c bin_tree.cpp -o prog
    ./prog
```

Testarea programului:

- Am introdus de mână un element apoi am introdus câteva elemente aleatorii.

```
g++ main.c bin_tree.cpp -o prog
./prog
__main__
__bin_tree__

1. adauga nod        3. cautarea
11. genereaza nod    4. modificarea
Afisarea:            5. numarul de elemente
21. inordine         6. inaltimea arborelui
22. preordine        7. eliberarea
23. postordine       0. iesirea
2. pe nivele
Optiunea - 1
citirea imobilului 1
proprietar:  Catalin
tipul:  casa
adresa:  livezilor 12
suprafata : 100
costul : 10000

1. adauga nod        3. cautarea
11. genereaza nod    4. modificarea
Afisarea:            5. numarul de elemente
21. inordine         6. inaltimea arborelui
22. preordine        7. eliberarea
23. postordine       0. iesirea
2. pe nivele
Optiunea - 11
imobilului 2, este generat:
proprietarul: Melissa
tipul: Magazin
adresa: Strada Alexei Mateevici
suprafata: 66 m^2
costul: 63030 $

1. adauga nod        3. cautarea
11. genereaza nod    4. modificarea
Afisarea:            5. numarul de elemente
21. inordine         6. inaltimea arborelui
22. preordine        7. eliberarea
23. postordine       0. iesirea
2. pe nivele
Optiunea - 11
imobilului 6, este generat:
proprietarul: Sam
tipul: Birou
adresa: Strada Alexei Mateevici
suprafata: 31 m^2
costul: 15655 $

1. adauga nod        3. cautarea
11. genereaza nod    4. modificarea
Afisarea:            5. numarul de elemente
21. inordine         6. inaltimea arborelui
22. preordine        7. eliberarea
23. postordine       0. iesirea
2. pe nivele
Optiunea - 11
imobilului 7, este generat:
proprietarul: Alex
tipul: Birou
adresa: Strada Alexei Mateevici
suprafata: 109 m^2
costul: 106820 $

1. adauga nod        3. cautarea
11. genereaza nod    4. modificarea
Afisarea:            5. numarul de elemente
21. inordine         6. inaltimea arborelui
```

```
Optiunea - 11
imobilului 3, este generat:
proprietarul: Mirela
tipul: Fabrica
adresa: Strada Albisoara
suprafata: 19 m^2
costul: 2926 $

1. adauga nod        3. cautarea
11. genereaza nod    4. modificarea
Afisarea:            5. numarul de elemente
21. inordine         6. inaltimea arborelui
22. preordine        7. eliberarea
23. postordine       0. iesirea
2. pe nivele
Optiunea - 11
imobilului 4, este generat:
proprietarul: Vitalie
tipul: Fabrica
adresa: Strada Calea Iesilor
suprafata: 100 m^2
costul: 31600 $

1. adauga nod        3. cautarea
11. genereaza nod    4. modificarea
Afisarea:            5. numarul de elemente
21. inordine         6. inaltimea arborelui
22. preordine        7. eliberarea
23. postordine       0. iesirea
2. pe nivele
Optiunea - 11
imobilului 5, este generat:
proprietarul: Petru
tipul: Fabrica
adresa: Strada Calea Iesilor
suprafata: 31 m^2
costul: 16213 $
```

îm total am introdus 8 elemente.

- Apoi am utilizat toate metodele de afişare a arborelui.

```
Optiunea - 21
inordine
Mirela          Fabrica             Strada Albisoara                19 m^2      2926 $
Catalin         casa                livezilor 12                   100 m^2     10000 $
Sam             Birou               Strada Alexei Mateevici         31 m^2     15655 $
Petru           Fabrica             Strada Calea Iesilor            31 m^2     16213 $
Vitalie         Fabrica             Strada Calea Iesilor           100 m^2     31600 $
Melissa         Magazin             Strada Alexei Mateevici         66 m^2     63030 $
Mihail          Apartament          Strada Albisoara                99 m^2     75240 $
Alex            Birou               Strada Alexei Mateevici        109 m^2    106820 $
```

```
Optiunea - 22
preordine
Catalin         casa                livezilor 12                   100 m^2     10000 $
Mirela          Fabrica             Strada Albisoara                19 m^2      2926 $
Melissa         Magazin             Strada Alexei Mateevici         66 m^2     63030 $
Vitalie         Fabrica             Strada Calea Iesilor           100 m^2     31600 $
Petru           Fabrica             Strada Calea Iesilor            31 m^2     16213 $
Sam             Birou               Strada Alexei Mateevici         31 m^2     15655 $
Alex            Birou               Strada Alexei Mateevici        109 m^2    106820 $
Mihail          Apartament          Strada Albisoara                99 m^2     75240 $
```

```
Optiunea - 23
postordine
Mirela          Fabrica             Strada Albisoara                19 m^2      2926 $
Sam             Birou               Strada Alexei Mateevici         31 m^2     15655 $
Petru           Fabrica             Strada Calea Iesilor            31 m^2     16213 $
Vitalie         Fabrica             Strada Calea Iesilor           100 m^2     31600 $
Mihail          Apartament          Strada Albisoara                99 m^2     75240 $
Alex            Birou               Strada Alexei Mateevici        109 m^2    106820 $
Melissa         Magazin             Strada Alexei Mateevici         66 m^2     63030 $
Catalin         casa                livezilor 12                   100 m^2     10000 $
```

```
Optiunea - 2

nivelul curent: 0
numarul maxim de noduri posibile: 1
Catalin         casa                livezilor 12                   100 m^2     10000 $

nivelul curent: 1
numarul maxim de noduri posibile: 2
Mirela          Fabrica             Strada Albisoara                19 m^2      2926 $
Melissa         Magazin             Strada Alexei Mateevici         66 m^2     63030 $

nivelul curent: 2
numarul maxim de noduri posibile: 4
Vitalie         Fabrica             Strada Calea Iesilor           100 m^2     31600 $
Alex            Birou               Strada Alexei Mateevici        109 m^2    106820 $

nivelul curent: 3
numarul maxim de noduri posibile: 8
Petru           Fabrica             Strada Calea Iesilor            31 m^2     16213 $
Mihail          Apartament          Strada Albisoara                99 m^2     75240 $

nivelul curent: 4
numarul maxim de noduri posibile: 16
Sam             Birou               Strada Alexei Mateevici         31 m^2     15655 $

numarul de noduri: 8
```

- Căutarea elementului după preț:

```
Optiunea - 3
elementul cautat (dupa pret)
16213
Petru           Fabrica             Strada Calea Iesilor                31 m^2     16213 $
```

- Modificarea nodului:

```
Optiunea - 4
elementul cautat pentru modificare (dupa pret)
31600
modifica proprietarul? (Vitalie) [y/n]
y
proprietar nou:  Vitalie Tibirna
modifica tipul? (Fabrica) [y/n]
n
modifica adresa? (Strada Calea Iesilor) [y/n]
n
modifica suprafata? (100) [y/n]
y
suprafata : 500
modifica pretul? (31600) [y/n]
y
costul : 90000
```

- După modificarea elementului subarborele începând cu elementul respectiv au fost redistribuit:

```
nivelul curent: 0
numarul maxim de noduri posibile: 1
Catalin         casa                livezilor 12                  100 m^2     10000 $

nivelul curent: 1
numarul maxim de noduri posibile: 2
Mirela          Fabrica             Strada Albisoara               19 m^2      2926 $
Melissa         Magazin             Strada Alexei Mateevici        66 m^2     63030 $

nivelul curent: 2
numarul maxim de noduri posibile: 4
Petru           Fabrica             Strada Calea Iesilor           31 m^2     16213 $
Alex            Birou               Strada Alexei Mateevici       109 m^2    106820 $

nivelul curent: 3
numarul maxim de noduri posibile: 8
Sam             Birou               Strada Alexei Mateevici        31 m^2     15655 $
Mihail          Apartament          Strada Albisoara               99 m^2     75240 $

nivelul curent: 4
numarul maxim de noduri posibile: 16
Vitalie Tibirna Fabrica             Strada Calea Iesilor          500 m^2     90000 $
```

Înainte de modificare Vitalie era în nivelul 2 iar dupa a ajuns în nivelul 4, Petru şi Sam au urcat un nivel.

```
nivelul curent: 2
numarul maxim de noduri posibile: 4
Vitalie         Fabrica             Strada Calea Iesilor          100 m^2     31600 $
Alex            Birou               Strada Alexei Mateevici       109 m^2    106820 $

nivelul curent: 3
numarul maxim de noduri posibile: 8
Petru           Fabrica             Strada Calea Iesilor           31 m^2     16213 $
Mihail          Apartament          Strada Albisoara               99 m^2     75240 $

nivelul curent: 4
numarul maxim de noduri posibile: 16
Sam             Birou               Strada Alexei Mateevici        31 m^2     15655 $
```

- Identificarea numărului de noduri:

```
Optiunea - 5
numarul de noduri: 8
```

- Determinarea Înălțimii arborelui:

```
nivelul curent: 0
numarul maxim de noduri posibile: 1
Catalin         casa                livezilor 12                    100 m^2     10000 $

nivelul curent: 1
numarul maxim de noduri posibile: 2
Mirela          Fabrica             Strada Albisoara                 19 m^2      2926 $
Melissa         Magazin             Strada Alexei Mateevici          66 m^2     63030 $

nivelul curent: 2
numarul maxim de noduri posibile: 4
Petru           Fabrica             Strada Calea Iesilor             31 m^2     16213 $
Alex            Birou               Strada Alexei Mateevici         109 m^2    106820 $

nivelul curent: 3
numarul maxim de noduri posibile: 8
Sam             Birou               Strada Alexei Mateevici          31 m^2     15655 $
Mihail          Apartament          Strada Albisoara                 99 m^2     75240 $

nivelul curent: 4
numarul maxim de noduri posibile: 16
Vitalie Tibirna Fabrica             Strada Calea Iesilor            500 m^2     90000 $

inaltimii arborelui este: 4
```

Arborele are 5 nivele iar înălțimea lui maximă este 4.

- Eliberarea memoriei:

```
1. adauga nod          3. cautarea
11. genereaza nod      4. modificarea
Afisarea:              5. numarul de elemente
21. inordine           6. inaltimea arborelui
22. preordine          7. eliberarea
23. postordine         0. iesirea
2. pe nivele
Optiunea - 7

1. adauga nod          3. cautarea
11. genereaza nod      4. modificarea
Afisarea:              5. numarul de elemente
21. inordine           6. inaltimea arborelui
22. preordine          7. eliberarea
23. postordine         0. iesirea
2. pe nivele
Optiunea - 2
radacina arborelui este (nil)
```

După eliberarea memoriei dacă încercăm să facem altceva decât să adăugăm un element v-om primi acest mesaj: rădăcina arborelui este null-ă.

Concluzii:

1. Am obținut deprinderile practice de a implementa si de a utiliza TAD „Arbore binar”.
2. Am parcurs arborele utilizând atât metoda recursivă pentru parcurgerea în preordine, ordine ş postordine dar și parcurgerea cu ajutorul cozii care permite afișarea pe nivele.
3. Am scris niste funcţiile necesare pentru a efectua operațiile de baza cu arborele cum ar fi: citirea, căutarea, modificarea.
4. Deși ne este valabliă căutarea binară noi o putem utiliza doar pentru cheia aleasă, căutarea după alte date din nod ar fi liniară deci nu atât de eficientă. Presupun că în python ar fi mai simplu de a alege cheia dinamic fiind că variabilele nu au un tip predefinit ceea ce ar face mai simplu introducerea unui arbore ordonat.
5. Arborii în general pot fi utilizaţi în domeniul matematicii discrete însă la moment nu cunosc o aplicare a arborilor binari în matematica discretă ei fiind un caz particular de arbore.
6. Cred că ar fi eficient de utilizat arborii binari întrun sistem spre exemplu unde utilizatorii au un id unic cum ar fi idnp-ul care ar servi drept chei, însă trebuie de luat în considerare cum am alege rădăcina pentru a obține un arbore echilibrat.
7. Modificarea unui element dacă acesta are copii poate afecta destul de tare structura arborelui.
8. La realizarea acestui laborator am învățat să utilizez un nou editor: VIM datorită faptului că nu are autocorecție și nu sugerează cuvintele cheie m-am deprins mai bine să scriu singur cuvintele în întregime și am utilizat șoricelul mai puțin.