МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсивная обработка иерархических списков

Студент(ка) гр. 9382 _____ Голубева В.П.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Научиться рекурсивно работать с иерархическими списками.

Задание.

1д. Задано бинарное дерево *b* типа *BT* с типом элементов *Elem*. Для введенной пользователем величины *E* (**var** *E*: *Elem*):

- определить, входит ли элемент *E* в дерево *b*;
- определить число вхождений элемента *E* в дерево *b*;
- найти в дереве *b* длину пути (число ветвей) от корня до ближайшего узла с элементом *E* (если *E* не входит в *b*, за ответ принять -1).

Основные теоретические положения.

Определим скобочное представление бинарного дерева (БД):

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= ///,

< непустое БД > ::= < корень > < БД > < БД > .

Например, бинарное дерево, изображенное на рис. 3.4, имеет скобочное представление

abd/h//e//cfi//j//g/kl///

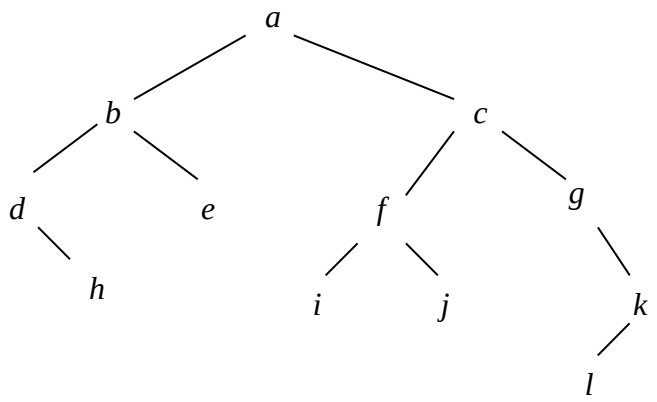

Рис. 3.4. Бинарное дерево

Реализованные алгоритмы.

Был реализован класс бинарного дерева BinTree, который включает поля T info, BinTree *lt - левое поддерево, BinTree *rt - правое поддерево и методы для работы с бинарным деревом.

В программе были использован шаблон template <class T> для задания типа элемента поля T info бинарного дерева.

Приватные методы(вызывются публичными методами внутри класса для работы с объектом класса дерева):

BinTree* left (BinTree* b) — принимает бинарное дерево b, возвращаем левое поддерево, если b не нулевое

BinTree* consBT(const T &x, BinTree *lst, BinTree *rst) - создаём новое дерево, путём склеивания его из двух(BinTree *lst и BinTree *rst) и добавления корня T x.

void destroy (BinTree* b) — получает на вход бинарное дерево b, проверяет, что дерево не нулевое и рекурсивно очищает правое и левое поддеревья

Публичные методы:

BinTree* enterBT () - считывает из потока ifstream infile запись бинарного дерева, создаёт дерево и возвращает указатель на него

BinTree() - конструктор по умолчанию

~BinTree() - декструктор вызывает метод void destroy (BinTree*) для очистки памяти дерева

void outBT(BinTree* b) — принимает указатель на бинарное дерево, выводит на экран запись введённого дерева

void displayBT (BinTree* b, int n) - принимает указатель на бинарное дерево, выводит на экран запись введённого дерева с указание уровня

void find_elem (BinTree* b, int n, T elem, int &count, int &track) — принимает указатель на бинарное дерево, уровень n, элемент для поиска в дереве, количество вхождений элемента в дерево int &count и минимальный

путь да элемента int &track, после отработки результат отражается в count и track

BinTree* create() - создаёт новое пустое дерево

bool isNull(BinTree* b) — проверяет, является ли дерево пустым

T rootBT (BinTree* b) — возвращает поле info дерева b

Описание рекурсивной функции.

void find_elem (BinTree* b, int n, T elem, int &count, int &track) - принимает указатель на бинарное дерево, уровень n, элемент для поиска в дереве, количество вхождений элемента в дерево int &count и минимальный путь да элемента int &track

Функция проверяет, что дерево не нулевое, затем проверяет совпадение текущего значение поля info дерева. Если совпадает, то count увеличивается, и идёт проверка на то, что текущий путь до элемента меньше минимального или нет. Если да, то минимальный путь изменяется. Выводится сообщение о соответствии. С отступами, соответствующими глубине рекурсии. Если поле info дерева не совпало с введённым элементом, то выводится сообщение о несоответствии. Дальше функция рекурсивно вызывается для правого и левого поддерева с увеличением уровня n дерева

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	l ga	Элемент для поиска: l Неправильно введён список	Проверка на некорректных данных
2.	l s/df/f/f/	Элемент для поиска: l	Проверка на некорректных

		Неправильно введён список	данных
3.	l	Элемент для поиска: l Неправильно введён список	Проверка на некорректных данных
4.	l l//	Элемент для поиска: l Вы ввели: l// Совпадение с искомым элементом на уровне 1 Искомый элемент обнаружен Количество вхождений в дерево 1 Минимальный путь до элемента 1	
5.	l ga//f///	Элемент для поиска: l Вы ввели: ga//f// Искомый элемент не совпадает с g на уровне 1 Искомый элемент не совпадает с f на уровне 2 Искомый элемент не совпадает с a на уровне 2 Искомый элемент не обнаружен Количество вхождений в дерево 0 Путь до элемента -1	
6.	x gkpb/a//s//xc//z//	Элемент для поиска: x	

	d//	Вы ввели: gkpb/a//s//xc//z//d// Искомый элемент не совпадает с g на уровне 1 Искомый элемент не совпадает с d на уровне 2 Искомый элемент не совпадает с k на уровне 2 Совпадение с искомым элементом на уровне 3 Искомый элемент не совпадает с z на уровне 4 Искомый элемент не совпадает с c на уровне 4 Искомый элемент не совпадает с p на уровне 3 Искомый элемент не совпадает с s на уровне 4 Искомый элемент не совпадает с b на уровне 4 Искомый элемент не совпадает с a на уровне 5 Искомый элемент обнаружен	

		Количество вхождений в дерево 1 Минимальный путь до элемента 3	
7.	x bm//eg//hu//i//l//	Элемент для поиска: x Вы ввели: bm//eg//hu//i// Искомый элемент не совпадает с b на уровне 1 Искомый элемент не совпадает с e на уровне 2 Искомый элемент не совпадает с h на уровне 3 Искомый элемент не совпадает с i на уровне 4 Искомый элемент не совпадает с u на уровне 4 Искомый элемент не совпадает с g на уровне 3 Искомый элемент не совпадает с m на уровне 2 Искомый элемент не обнаружен Количество вхождений в дерево 0 Путь до элемента -1	
8.	q asf////	Элемент для поиска: q	

		Вы ввели: asf//// Искомый элемент не совпадает с a на уровне 1 Искомый элемент не совпадает с s на уровне 2 Искомый элемент не совпадает с f на уровне 3 Искомый элемент не обнаружен Количество вхождений в дерево 0 Путь до элемента -1	
9.	q w//qqqq////////	Элемент для поиска: q Вы ввели: w/qqqq///// Искомый элемент не совпадает с w на уровне 1 Совпадение с искомым элементом на уровне 2 Совпадение с искомым элементом на уровне 3 Совпадение с искомым элементом на уровне 4 Совпадение с искомым элементом на уровне 5	

		Искомый элемент обнаружен Количество вхождений в дерево 4 Минимальный путь до элемента 2	
10.	f we///	Элемент для поиска: f Вы ввели: we/// Искомый элемент не совпадает с w на уровне 1 Искомый элемент не совпадает с e на уровне 2 Искомый элемент не обнаружен Количество вхождений в дерево 0 Путь до элемента -1	

Выводы.

Я научилась работать с бинарными деревьями. Создала программу, которая проверяет бинарное дерео на наличие в нём ввёдённого элемента, считат количество вхождений в дерево и выводит минимальный путь до элемента, если он найден.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: BinTree.h

```
#pragma once
#include <fstream>
#include <cstdlib>

#include <iostream>
#include <cstdlib>
#include "limits.h"

using namespace std;

typedef unsigned int unInt;
//typedef char base;
ifstream infile ("KLP.txt");

template <class T>

class BinTree{
    T info;

     BinTree *lt; //левое поддерево
     BinTree *rt; //правое поддерево

    BinTree* left (BinTree* b)
        {   if (b == nullptr) {
                cerr << "Error: Left(null) \n";
                exit(1);
            }
            else
                 return b ->lt;//возвращаем левое поддерево, если
b не нулевое
        }
//-------------------------------------
    BinTree* right (BinTree* b)
    {    if (b == nullptr) {
            cerr << "Error: Right(null) \n";
            exit(1);
            }
        else
            return b->rt;//возвращаем правое поддерево, если b не
нулевое
    }
//-------------------------------------
    BinTree* consBT(const T &x, BinTree *lst,  BinTree *rst)
    {    BinTree *p=new BinTree;

        if ( p != NULL)    {
            p ->info = x;
            p ->lt = lst;
            p ->rt = rst;
```

```
                return p;//создаём новое дерево, путём склеивания его
из двух и добавления корня
            }
            else {
                cerr << "Memory not enough\n";
                exit(1);
                }
        }
    //-------------------------------------
        void destroy (BinTree* b)
         {     if (b != NULL){//проверяем, что дерево не нулевое и
рекурсивно очищаем правое и левое поддеревья
                destroy (b->lt);
                destroy (b->rt);
                delete b;
                b = NULL;
            }
        }

    public:

    //-----------------------------------------
        BinTree* enterBT ()
        {
            T ch;
            BinTree* p;
            BinTree* q;
            if (!infile){
                cout<<"Неправильно введён список\n";
                exit(1);
            }

            infile >> ch;//считываем очередной символ из файла
            if (ch=='/')
                return nullptr;
            else {
                p = enterBT();
                q = enterBT();
                return consBT(ch, p, q);
            }
        }

     //-----------------------------------------
        BinTree(){
            //конструктор по умолчанию
            lt=nullptr;
            rt=nullptr;
        }

    //-----------------------------------------
        ~BinTree(){
            //декструктор. Вызывает метод для очистки памяти дерева
            this->destroy(this);
        }

    //-----------------------------------------
```

```
        void outBT(BinTree* b)
        {
            //выводит на экран запись введённого дерева
            if (b!=nullptr) {
                cout << rootBT(b);
                outBT(left(b));
                outBT(right(b));
            }
            else {
                cout<<"/";
            };
        }

    //---------------------------------------
        void displayBT (BinTree* b, int n)
        {
              //выводит на экран запись введённого дерева с указание
уровня
            if (b!=NULL) {

                cout << ' ' << rootBT(b)<< n;

                if(!isNull(right(b))){
                        cout << endl;
                        for (int i=1;i<=n;i++)
                            cout << " ";
                        displayBT (right(b),n+1);
                    }
                else
                    cout << endl;
                if(!isNull(left(b))){
                    for (int i=1;i<=n;i++)
                        cout << " ";
                    displayBT (left(b),n+1);}
            }
            else {

            };
        }

    //---------------------------------------
    void  find_elem  (BinTree*  b,  int  n,  T  elem,  int  &count,  int
&track)
        {
            if (b!=NULL) {
                if (elem==rootBT(b)){
                    count+=1;
                    if (n<track)
                        track=n;
                      cout<<"Совпадение с искомым элементом на уровне
"<<n<<"\n";
                }
                else
                          cout <<"Искомый элемент не совпадает с "<<
rootBT(b)<<" на уровне "<< n<<"\n";
```

```
                    if(!isNull(right(b))) {
                            cout << endl;
                            for (int i=1;i<=n;i++)
                                cout << " ";
                                find_elem (right(b),n+1, elem, count,
track);
                        }
                    else
                        cout << endl;
                    if(!isNull(left(b))) {
                        for (int i=1;i<=n;i++)
                            cout << " ";
                        find_elem (left(b),n+1, elem, count, track);}

            }
            else {};
        }
    //-------------------------------------
         BinTree* create()
        {
            return nullptr;
        }
    //-------------------------------------
        bool isNull(BinTree* b)
        {
            return (b == nullptr);
        }
    //-------------------------------------
        T rootBT (BinTree* b)
        {    if (b == nullptr) {
                cerr << "Error: RootBT(null) \n";
                exit(1);
            }
            else
                return b->info;//возвращает значение поля информации
        }
    };
```

Название файла: main.cpp

```
    #include "BinTree.h"

    int main ()
    {
        int track= INT_MAX;
        int count=0;

        char elem;

        infile>>elem;//вводим искомый элемен  из файла

        cout<<"Элемент для поиска: "<<elem<<"\n\n";

        BinTree<char>* b;
```

```
    b=b->enterBT();//вводим дерево из файла

    cout<<"Вы ввели: ";

    b->outBT(b);

    cout<<"\n\n";

    b->find_elem(b, 1, elem, count, track);

    if (count==0){
        cout<<"Искомый элемент не обнаружен\n";
        cout<<"Количество вхождений в дерево 0\n";
        cout<<"Путь до элемента -1\n";
    }
    else
    {
       cout<<"Искомый элемент обнаружен\n";
       cout<<"Количество вхождений в дерево "<<count<<"\n";
       cout<<"Минимальный путь до элемента "<< track<<"\n";
    }

    cout << endl;

    return  0;
}
```

Название файла: KLP.txt

q

we///