

**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.

Изучить структуру данных — бинарные деревья и научиться с ними работать.

### 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):

$\langle \text{БД} \rangle ::= \langle \text{пусто} \rangle \mid \langle \text{непустое БД} \rangle$ ,

$\langle \text{пусто} \rangle ::= ///$ ,

$\langle \text{непустое БД} \rangle ::= \langle \text{корень} \rangle \langle \text{БД} \rangle \langle \text{БД} \rangle$ .

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

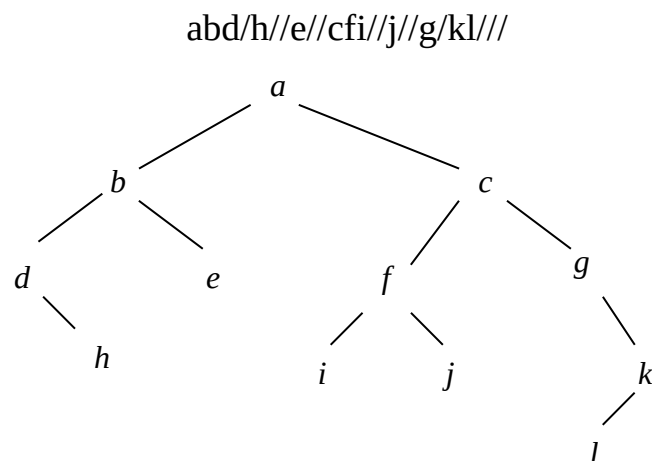


Рис. 3.4. Бинарное дерево

## Функции и структуры данных.

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(вызываются 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 поддерева

Публичные методы:

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

`BinTree()` - конструктор по умолчанию

`~BinTree()` - деконструктор 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 указанием 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 алгоритма.

Алгоритм поиска элемента реализован функцией `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. Далее функция рекурсивно вызывается для правого и левого поддеревя 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	Элемент для поиска: l	Проверка на

	s/df/f/f/	Неправильно введен список	некорректных данных
3.	l	Элемент для поиска: l  Неправильно введен 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	Элемент для поиска: x	

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

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

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



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

### **Выводы.**

Была изучена такая структура данных как бинарные деревья. Была создана программа, которая проверяет бинарное дерево на наличие в нём введённого элемента, считает 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 введен 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(){
    //деструктор. 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 указанием
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 элемент 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///