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

# ОТЧЕТ

**по лабораторной работе №5**
**по дисциплине «Алгоритмы и структуры данных»**
**Тема: Бинарные деревья поиска**

Студент гр. 7383 _____ Левкович Д.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2018

## Содержание

**Цель работы**

Познакомиться с рандомизированными деревьями поиска и научиться реализовывать их на языке программирования C++.

По заданному файлу F (типа file of Elem), все элементы которого различны, построить рандомизированное дерево поиска. Для построенного БДП проверить, входит ли в него элемент e типа Elem, и если входит, то удалить элемент e из дерева поиска.

**Реализация задачи**

Двоичное дерево поиска (англ. binary search tree, BST) — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.
- У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

В данной работе было написано несколько функций и класс для работы с деревом поиска.

`class BST`– структура, представляющая узел БДП, содержит в себе поля `int key` для хранения ключа, `int size` для хранения данного дерева с корнем в данном узле, `BST* left, right` для хранения указателей на правое и левое поддерево.

`BST* rotateright (node* p)` – функция, делающая правый поворот вокруг узла *p*.

`BST* rotateleft(node* p)` – функция, делающая левый поворот вокруг узла *p*.

`BST* insertroot(BST* p, int k)` – вставка узла с ключом `k` в корень дерева.

`BST* insertrandom(BST* p, int k)` – функция, добавляющая узел с ключом *k*. Если ключ *k* больше (меньше) ключа рассматриваемого узла, то он вставляется вправо (влево) от этого узла, при надобности делается правый или левый поворот.

`node* join(node *p, node *q)` – функция, получающая на вход два дерева, которые она объединяет.

`node* remove(node* p, int k)` – функция получает на вход дерево и ключ узла. Удаляет узел с заданным ключом, объединяя его правое и левое поддерево с помощью функции `merge`.

`void printtree(node* treenode, int l)` – функция, печатающая дерево.

`int main()` – головная функция, которая в зависимости от выбора пользователя считывает ключи из файла или с коносоли, затем создает бинарное дерево, выводит само дерево и удаляет узел, с заданным пользователем ключом.

**Тестирование**

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

**Вывод**

В ходе выполнения лабораторной работы были изучены основные понятия о деревьях поиска, была реализовано бинарное дерево поиска с рандомизацией на языке программирования C++. Также была написана программа для удаления узла с заданным ключом.

# ПРИЛОЖЕНИЕ А.

# ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <cstring>
#include <fstream>
using namespace std;
//template <class T>
class BST{
private:
    int key;
    int size;
    BST* right;
    BST* left;

public:
    BST(int k){key = k; left = right = nullptr; size = 1;}
    int Root(BST* b)
    {
        if (b == NULL)
            exit(1);
        else
            return b->key;
    }


    BST* Left(BST* b)
    {
        if (b == NULL) { exit(1); }
        else return b->left;
    }

    BST* Right(BST* b)
    {
        if (b == NULL) { exit(1); }
        else return b->right;
    }
    BST* find( BST* p, int k){
        if(!p)
            return nullptr;
        if(k==Root(p))
            return p;
        if(k<Root(p))
            return find(Left(p), k);
        else
            return find(Right(p), k);
    }

    int getsize(BST* p) // обертка для поля size, работает с пустыми деревьями (t=NULL)
    {
        if(!p)
            return 0;
        return p->size;
    }
```

```
    void fixsize(BST* p) // установление корректного размера дерева
    {
        p->size = getsize(Left(p))+getsize(Right(p))+1;
    }

    BST* rotateright(BST* p) // правый поворот вокруг узла p
    {
        BST* q = p->left;
        if( !q ) return p;
        p->left = q->right;
        q->right = p;
        q->size = p->size;
        fixsize(p);
        return q;
    }

    BST* rotateleft(BST* q) // левый поворот вокруг узла q
    {
        BST* p = q->right;
        if( !p ) return q;
        q->right = p->left;
        p->left = q;
        p->size = q->size;
        fixsize(q);
        return p;
    }

    BST* insertroot(BST* p, int k) // вставка нового узла с ключом k в корень дерева p
    {
        if( !p )
            return new BST(k);
        if( k<p->key )
        {
            p->left = insertroot(p->left,k);
            return rotateright(p);
        }
        else
        {
            p->right = insertroot(p->right,k);
            return rotateleft(p);
        }
    }

    BST* insertrandom(BST* p, int k) // рандомизированная вставка нового узла с ключом k в
дерево p
    {
        if( !p ) return new BST(k);
            if( rand()%(getsize(p)+1)==0 ){
              //  cout<<"COMBS: "<<rand()%(getsize(p)+1) <<endl;
                return insertroot(p,k);
            }
            if( p->key>k )
                p->left = insertrandom(Left(p),k);
            else
                p->right = insertrandom(Right(p),k);
            fixsize(p);
            return p;

    }

    BST* join(BST* p, BST* q) // объединение двух деревьев
    {
```

```
        if( !p ) return q;
        if( !q ) return p;
        if( rand()%(p->size+q->size) < p->size )
        {
            p->right = join(p->right,q);
            fixsize(p);
            return p;
        }
        else
        {
            q->left = join(p,q->left);
            fixsize(q);
            return q;
        }
    }

    BST* remove(BST* p, int k) // удаление из дерева p первого найденного узла с ключом k
    {
        if( !p ) return p;
        if( p->key==k )
        {
            BST* q = join(p->left,p->right);
            p=q;
            return p;
        }
        else if( k<p->key )
            p->left = remove(p->left,k);
        else
            p->right = remove(p->right,k);
        return p;
    }

    BST* Delete(BST* p){
        if (left)
            delete p->left;
        if (right)
            delete p->right;
        delete p;
        return p = NULL;
    }
};

void printtree(BST* treenode, int l){
   if(treenode==NULL){
       for(int i = 0;i<l;++i)
           cout<<"\t";
       cout<<'#'<<endl;
       return;
   }
   printtree(treenode->Right(treenode), l+1);
   for(int i = 0; i < l; i++)
       cout << "\t";
   cout << treenode->Root(treenode)<< endl;
   printtree(treenode->Left(treenode),l+1);
}


int main(){
    BST* b = NULL;
    int key = 0;
    string str;
    char forSwitch;
```

```
    while(1){
       cout<<"Press 1 to read from console, press 2 to read from 1.txt file, press 0 to
exit."<<endl;
       cin >> forSwitch;
       getchar();
       switch (forSwitch) {
           case '2':{
               ifstream infile("1.txt");
               if(!infile){
                     cout<<"There is no file"<<endl;
                     continue;
                     }
               getline(infile, str);
               break;
           }
           case '1':{
                   cout<<"Enter the keys"<<endl;
                   getline(cin, str);
                   break;
           }
           case '0':{
                   return 0;
           }
           default:{
                   cout<<"Incorrect input"<<endl;
                   break;
           }
       }
       char* arr = new char[str.size()+1];
       strcpy(arr, str.c_str());
       char* tok;
       tok = strtok(arr, " ");
       while(tok != NULL){
           b = b->insertrandom(b, atoi(tok));
           tok = strtok(NULL, " ");

       }
       cout<<endl;
       printtree(b,0);
   cout<<"Enter the key"<<endl<<"============================"<<endl;
       cin >> key;
       b = b->remove(b, key);
       printtree(b,0);
       b = b->Delete(b);
   str.clear();
   delete tok;
   delete[] arr;
   }

}
```

# ПРИЛОЖЕНИЕ Б.

# ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены на рис 1-3.

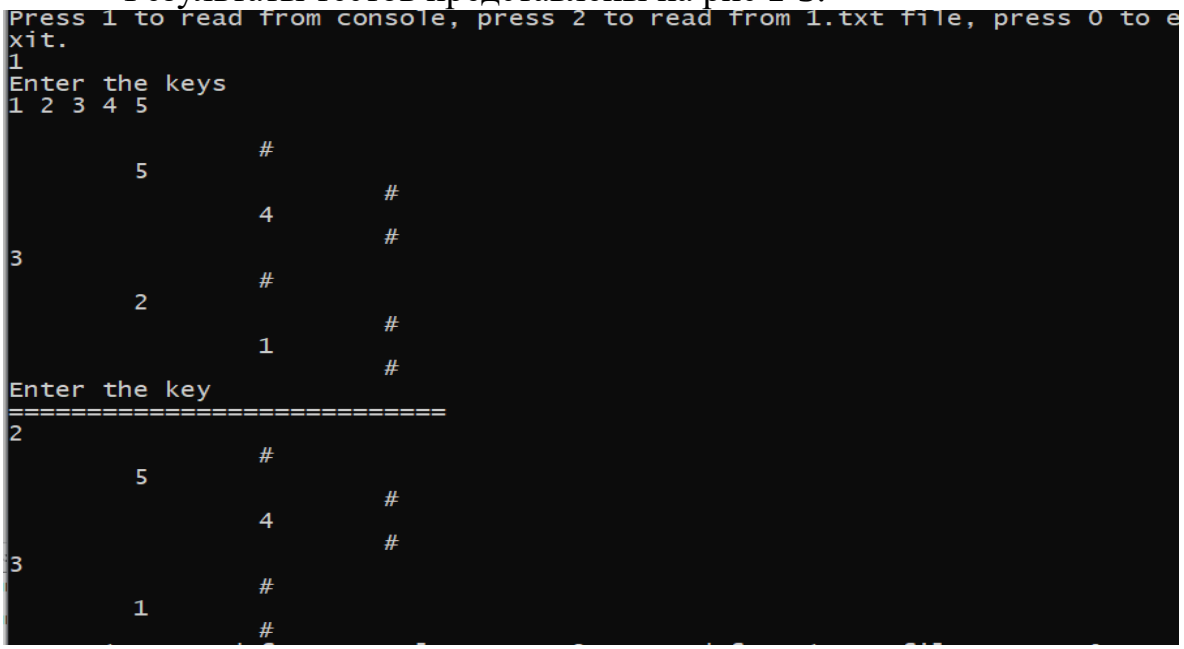


Рисунок 1 – Тест №1

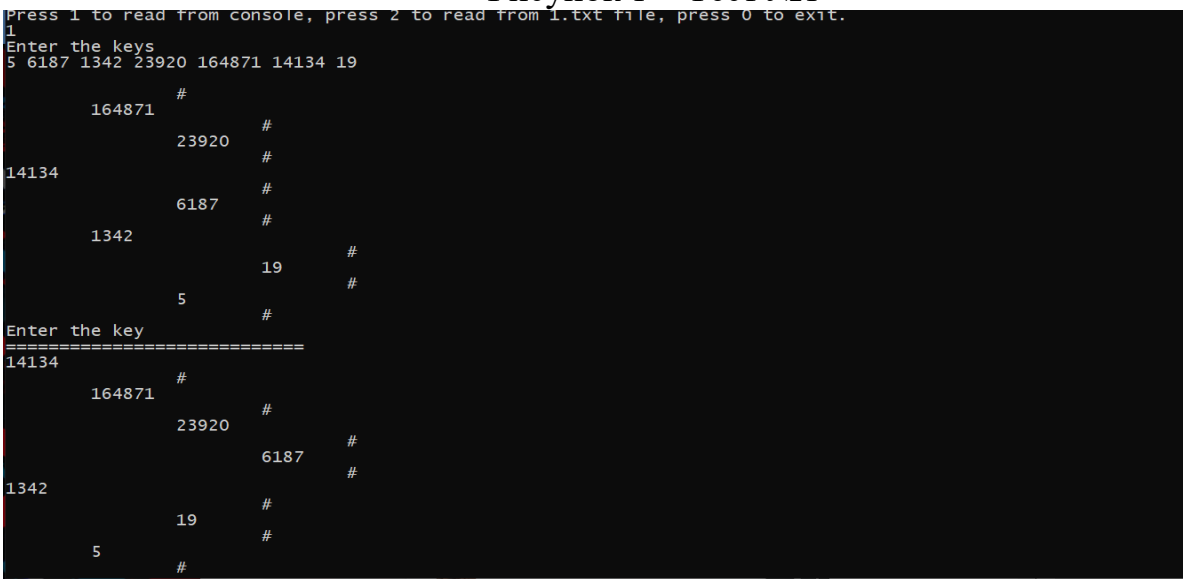


Рисунок 2 – Тест №2

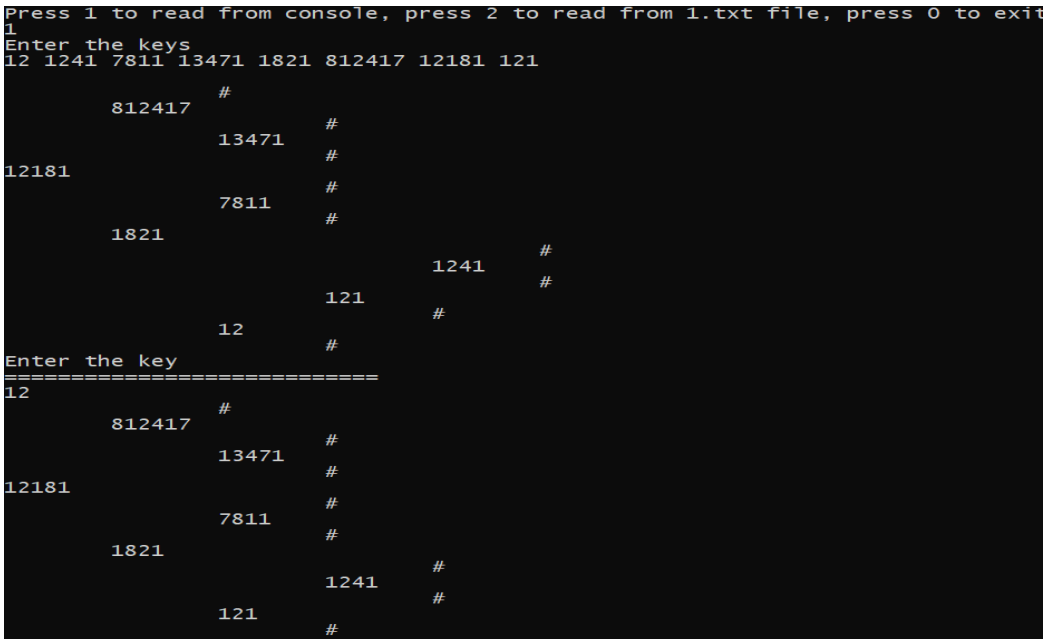


Рисунок 3 – Тест №3

Результаты показали, что данная программа работает корректно.