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 _____ Медведев И. С.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург

2018

Содержание

Цель работы

Познакомиться с рандомизированные пирамиды поиска и научиться реализовывать их на языке программирования C++.

По заданному файлу F (типа file of Elem), все элементы которого различны, построить рандомизированную пирамиду 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

Пирамида поиска (treap) – структура данных, объединяющая в себе бинарное дерево и кучу. Каждый узел содержит пару (*x*; *y*), где *x* – ключ бинарного дерева поиска, а *y* – приоритет бинарной кучи. Обладает свойствами: ключи *x* узлов правого (левого) поддерева больше (меньше) ключа *x* узла *n*, приоритеты *y* узлов правого и левого детей больше приоритета *y* узла *n*.

В данной работе было написано несколько функций и структура для работы с пирамидой поиска.

`struct node` – структура, представляющая узел БДП, содержит в себе поля `int key` для хранения ключа, `int prior` для хранения приоритета, `node* left, right` для хранения указателей на правое и левое поддерево.

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

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

`node* insert(int key, node* root)` – функция, добавляющая узел с ключом *k*, учитывая его приоритет и ключ. Если ключ *k* больше (меньше) ключа рассматриваемого узла, то он вставляется вправо (влево) от этого узла, при надобности делается правый или левый поворот.

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

`void printPriority(node* root)` – функция, печатающая приоритеты узлов (задаются случайным образом).

`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, печатает приоритеты ключей, 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 пирамидах поиска, была реализована рандомизированная пирамида поиска на языке программирования C++. Также была написана программа для удаления узла с заданным ключом.

ПРИЛОЖЕНИЕ А.

КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <fstream>
using namespace std;


struct node{     // структура для представления узлов дерева
    int key;
    int prior;
    node* left;
    node* right;
    node(int k) { key = k; left = right = NULL; prior = rand()%100;}
};



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

node* rotateleft(node* q){ // левый поворот вокруг узла q
```

```
    node* p = q->right;
    if( !p )
        return q;
    q->right = p->left;
    p->left = q;
    return p;
}


node* insert(int key, node* root){
    if(!root){
        node* p = new node(key);
        return (p);
    }
    if(key <= root->key)
    {
        root->left = insert(key, root->left);
        if(root->left->prior < root->prior)
            root = rotateright(root);
    }
    else{
        root->right = insert(key, root->right);
        if(root->right->prior < root->prior)
            root = rotateleft(root);
    }
    return root;
}
```

```
node* merge(node *p, node *q) {
    if (p == NULL) return q;
    if (q == NULL) return p;

    if (p->prior > q->prior) {
        p->right = merge(p->right, q);
        return p;
    }
    else {
        q->left = merge(p, q->left);
        return q;
    }
}



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

void printPriority(node* root){
    if (!root)
        return;
    cout<<"Priority of key ["<< root->key <<"] is "<<root-
>prior<<endl;
    printPriority(root->right);
```

```
    printPriority(root->left);
}



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



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

```
        cout << "\t";
    cout << treenode->key<< endl;
    printtree(treenode->left,l+1);
}
 int main(){
     node* treap = NULL;
     int key = 0;
     string str;
     char forSwitch;
     while(1){
        cout<<"Press 1 to read from console, press 2 to read from
Treap.txt file, press 0 to exit."<<endl;
        cin >> forSwitch;
        getchar();
        switch (forSwitch) {
            case '2':{
                ifstream infile("Treap.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){
            treap = insert(atoi(tok), treap);
            tok = strtok(NULL, " ");

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

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены на рис 1-3.

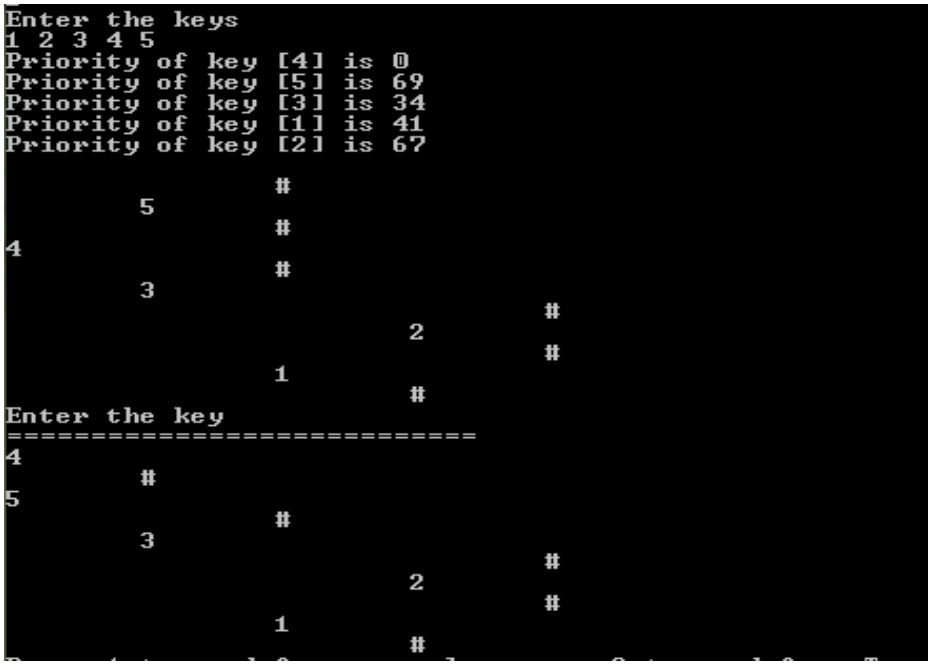

Рисунок 1 – Тест №1

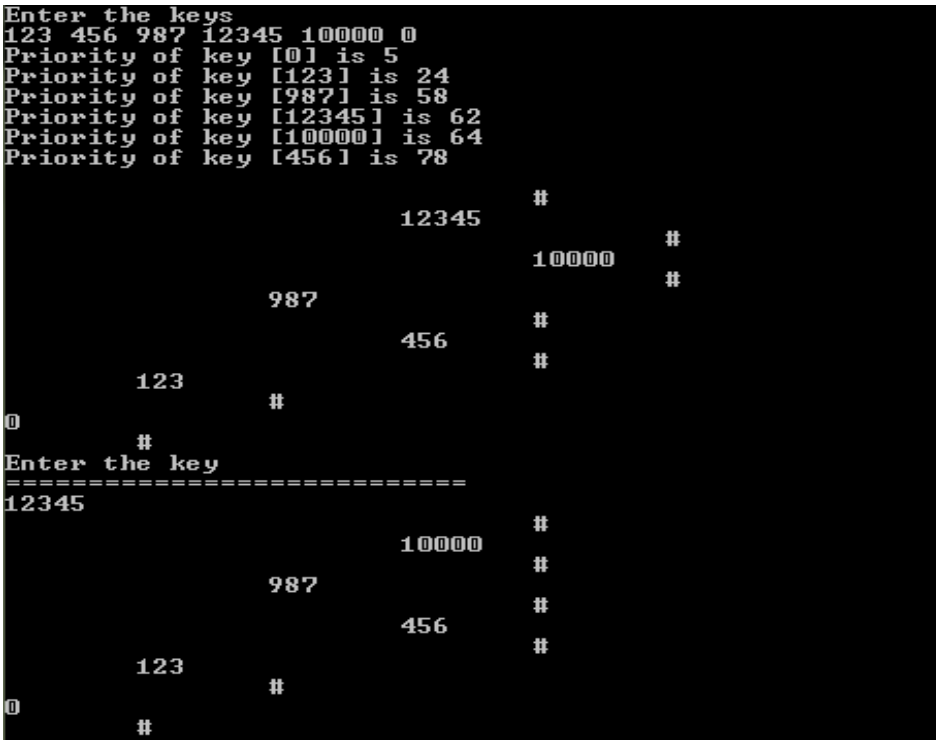

Рисунок 2 – Тест №2

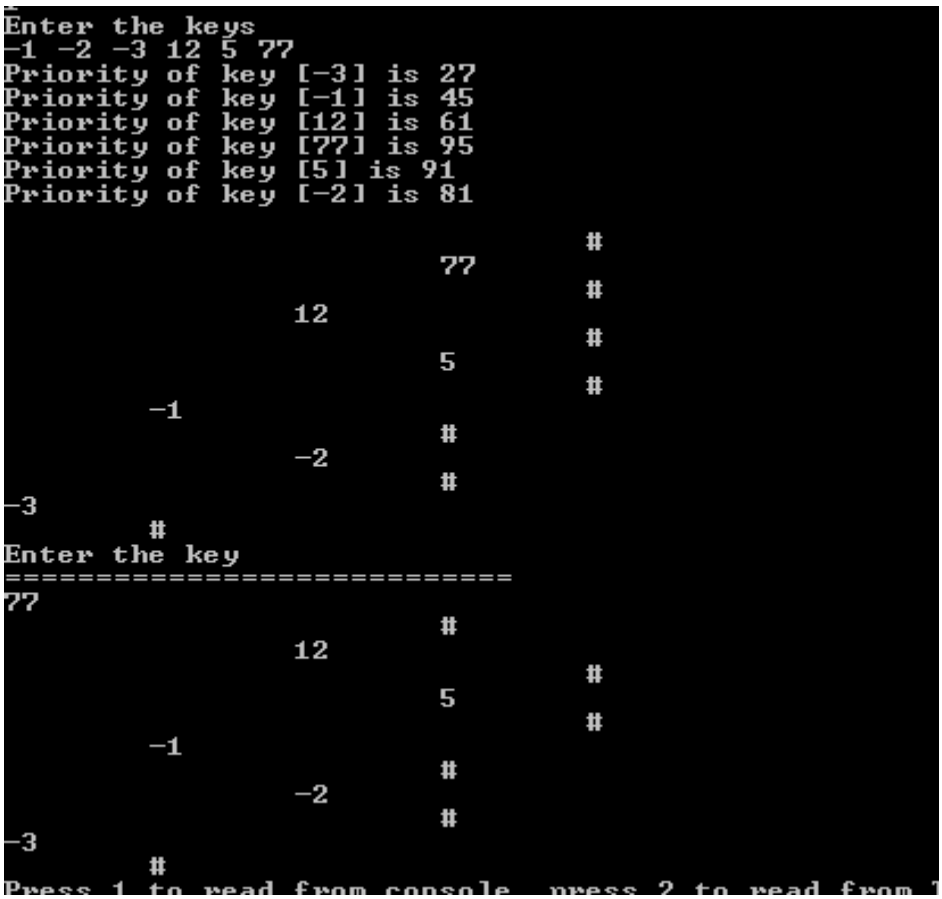

Рисунок 3 – Тест №3

Результаты показали, что данная программа работает корректно.