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

ОТЧЕТ

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

Студентка гр. 7383 _____ Маркова А.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2018

СОДЕРЖАНИЕ

Цель работы

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

Формулировка варианта 13:

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 не входит, то добавить элемент e в дерево 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.

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

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

`node* add(node* p, int el)` – функция, добавляющая новый элемент в пирамиду.

`node* find( node* tree, int key)` – функция для поиска элемента по ключу.

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

Тестирование

Программа была собрана в компиляторе g++ в OS Linux Ubuntu. В других системах тестирование не проводилось. Результаты тестирования, приведенные в приложении Б, показали, что поставленная задача была выполнена.

Выводы

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 добавления 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;     // ключ-значение
    int prior;   // приоритет
    node* left;  // указатель на левое поддерево
    node* right; // указатель на правое поддерево
    node(int k) {
        key = k; // инициализация структуры
        left = right = NULL;
        prior = rand()%100; // рандомные числа от 0 до 99
    }
};

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) { // вставка
    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* 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<<"Приоритет ключа ["<< root->key <<"] - "<<root->prior<<endl;
    printPriority(root->right);
    printPriority(root->left);
}

node* find( node* tree, int key) {
    if(!tree)
        return NULL;
    if(key == tree->key)
        return tree;
    if(key < tree->key)
        return find(tree->left, key);
    else
        return find(tree->right, key);
}
```

```
node* add(node* p, int el) { // добавление нового элемент
    if (find(p,el)) {
        cout << "Ключ [" << el << "] повторяется"<<endl;
     return p;
    }
    else {
        p=insert(el, p);
        cout<<"Приоритет нового ключа ["<< (find(p,el))->key <<"] -
"<<(find(p,el))->prior<<endl;
        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; // пирамида поиска
    int el = 0, c;
    string str;
    char forSwitch;
    cout << "\033[34m\tЗдравствуйте!Выберите что вы хотите:\033[0m\n
1) Нажмите 1, чтобы считать с консоли.\n 2) Нажмите 2, чтобы считать с
файла.\n 3) Нажмите 3, чтобы выйти из программы.\n" << endl;
    while(1) {
        cin >> forSwitch;
        getchar();
        switch (forSwitch) {
        case '2': {
            ifstream infile("Test.txt");
            if(!infile) {
                cout<<"Файл не может быть открыт!"<<endl;
```

```
                cout<<"Введите следующую команду:\n";
                continue;
            }
            getline(infile, str);
            break;
        }
        case '1': {
            cout<<"Введите ключи в строку:"<<endl;
            getline(cin, str);
            break;
        }
        case '3': {
            cout<<"До свидания!"<<endl;
            return 0;
        }
        default: {
            cout<<"Некорректные данные!"<<endl;
            return 0;
        }
        }
        char* arr = new char[str.size()+1];
        strcpy(arr, str.c_str()); // запись строки в массив, который
содержит последовательность символов с нулевым завершением
        char* tok;
        tok = strtok(arr, " "); // разделяем строку на цифры - ключи
        while(tok != NULL) {
            c = atoi(tok);      // конвертируем строку в величину типа
int
         if(c==0) {
             cout<<"Некорректные данные!"<<endl;
                return 0;
         }
            if (find(treap,c)) {
                cout << "Ключ [" << c << "] повторяется"<<endl; //
повторение ключа не допустимо, тк должен быть уникальным
                tok = strtok(NULL, " ");
                continue;
            }
            treap = insert(c, treap);
            tok = strtok(NULL, " ");
        }
        printPriority(treap); // печать приоритетов
        cout<<endl;
```

```
        printtree(treap,0);
        cout<<"--------------------------"<<endl<<"Введите ключ,
который хотите добавить"<<endl;
        cin >> el;
        treap = add(treap, el);
        printtree(treap,0);
        treap = Delete(treap);
        str.clear();
        delete tok;
        delete[] arr;
        cout<<"Введите следующую команду:\n";
    }
}
```

ПРИЛОЖЕНИЕ Б.
ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены на рис. 1-3.

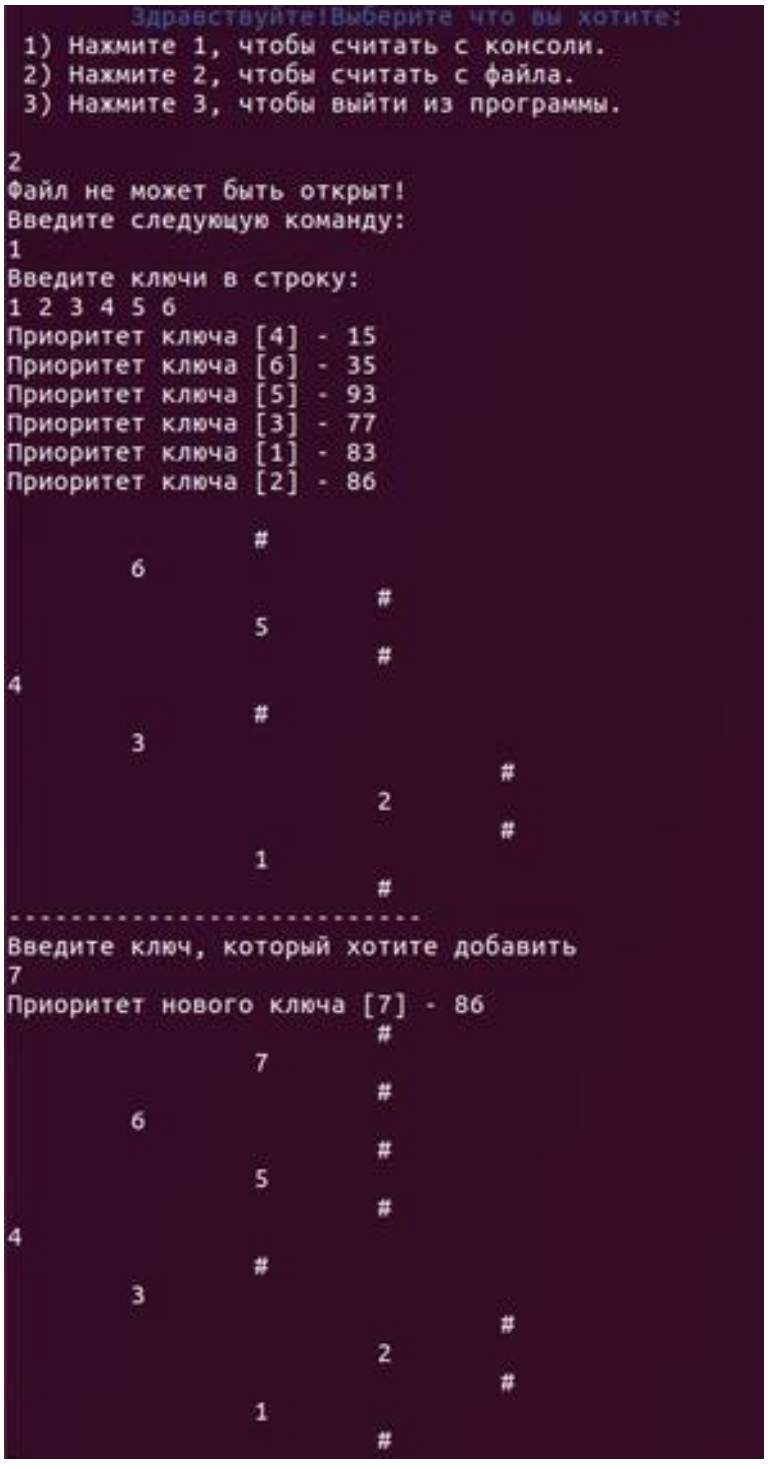

Рисунок 1 – Тест №1

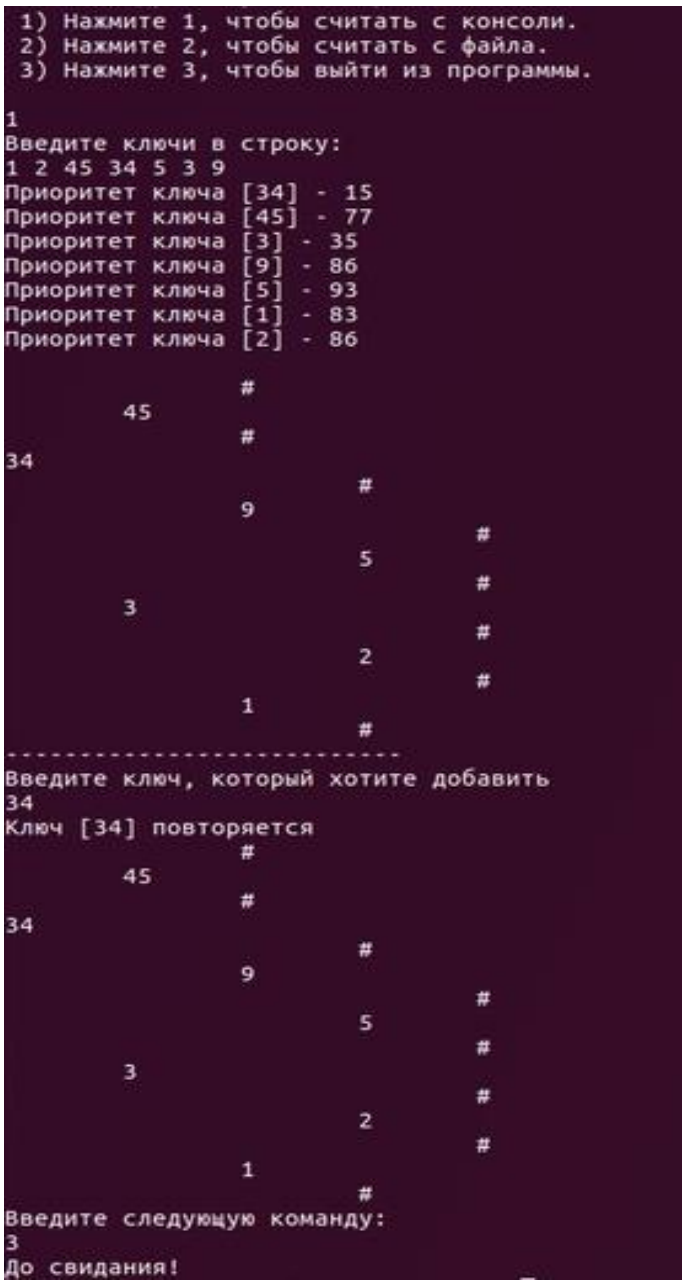

Рисунок 2 – Тест №2

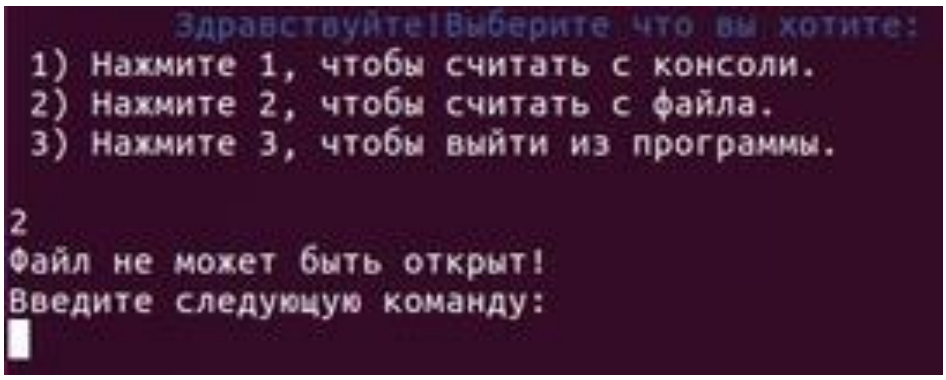

Рисунок 3 – Тест №3