

**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. 9304

Алексеевко Б.

Преподаватель

Филатов Ар.Ю.

Санкт-Петербург

2020

### **Цель работы.**

Написание бинарного дерева поиска в соответствии с вариантом задания, а также его функционала.

### **Задание.**

Вариант 16.

БДП: AVL-дерево; действие: 1+2б.

1. По заданной последовательности элементов *Elem* построить структуру данных определённого типа – БДП или хеш-таблицу;
2. Выполнить одно из следующих действий:
  - а) Для построенной структуры данных проверить, входит ли в неё элемент *e* типа *Elem*, и если входит, то в скольких экземплярах. Добавить элемент *e* в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.
  - б) Для построенной структуры данных проверить, входит ли в неё элемент *e* типа *Elem*, и если входит, то удалить элемент *e* из структуры данных (первое обнаруженное вхождение). Предусмотреть возможность повторного выполнения с другим элементом.
  - в) Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран в наглядном виде.
  - г) Другое действие.

### **Основные теоретические положения.**

AVL-дерево — это прежде всего двоичное дерево поиска, ключи которого удовлетворяют стандартному свойству: ключ любого узла дерева не меньше любого ключа в левом поддереве данного узла и не больше любого ключа в правом поддереве этого узла. Это значит, что для поиска нужного ключа в AVL-дереве можно использовать стандартный алгоритм. Для простоты дальнейшего изложения будем считать, что все ключи в дереве целочисленны и не повторяются.

Особенностью АВЛ-дерева является то, что оно является сбалансированным в следующем смысле: *для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу.*

### **Выполнение работы.**

Работает алгоритм в коде следующим образом: мы поддерживаем два указателя — `begin` и `end`, которые показывают, какая часть массива еще не была отсортирована. Как и в сортировке пузырьком мы идем по массиву и попутно сравниваем элементы, но при достижении конца массива, мы начинаем идти в обратную сторону, пока не будут отсортированы все элементы.

### **Выводы.**

Была реализована структура `Node` – узел АВЛ-дерева, в котором определены следующие поля:

- `int key` – значение ключа.
- `Int height` – значение высоты ключа.
- `Node* left` – указатель на левый элемент.
- `Node* right` – указатель на правый элемент.

Создан класс `Tree`, в котором определено поле АВЛ-дерева `root`, а также созданы некоторые методы для работы с АВЛ-деревом:

- `void differenceHight(node* tempNode)` – расчет разницы высоты левого и правого поддерева.
- `Int countingHeight(node* tempNode)` – расчет высоты АВЛ-дерева
- `node* balance(node* tempNode)` – метод, выполняющий балансировку АВЛ-дерева
- `node* pushNode(int key, node* tempNode)` – метод, который вставляет новый элемент в дерево.
- `Void printTree(node* tempNode)` – метод, который выводит в консоль АВЛ-дерево.

- Node\* findMin – метод, который возвращает наименьший элемент в узле.
- Node\* delMin – метод, который удаляет минимальный элемент в узле.
- Node\* del(Node\* tempNode, int key) – метод, который удаляет заданный элемент в узле.

### **Тестирование.**

Тестирование программы представлено в приложении Б.

### **Выводы.**

Во время выполнения лабораторной было изучено понятие AVL-дерева. Была разработана программа, которая строит AVL-дерево из потока данных, а также в соответствии с вариантом, удаляющая заданный элемент из дерева.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
using namespace std;

class Tree {
private:

    struct node
    {
        int key;
        int height;
        node* left;
        node* right;
        node(int k) { key = k; left = right = 0; height = 1; }
    };

    node* root;

    int differenceHeight(node* tempNode) {
        return calculateHeight(tempNode->right) - calculateHeight(tempNode->left);
    }

    node* rotateright(node* tempNode)
    {
        node* tempNodeQ = tempNode->left;
        tempNode->left = tempNode->right;
        tempNodeQ->right = tempNode;
        countingHeight(tempNode);
        countingHeight(tempNodeQ);
        return tempNodeQ;
    }

    node* turnLeft(node* tempNode)
    {
        node* tempNodeQ = tempNode->right;
        tempNode->right = tempNodeQ->left;
        tempNodeQ->left = tempNode;
        countingHeight(tempNode);
        countingHeight(tempNodeQ);
        return tempNodeQ;
    }

    int calculateHeight(node* tempNode) {
        if (tempNode) {
            return tempNode->height;
        }
        return 0;
    }

    void countingHeight(node* tempNode) {
        if (tempNode == nullptr) {
            return;
        }
        countingHeight(tempNode->right);
        countingHeight(tempNode->left);
        int rightHeight = calculateHeight(tempNode->right), leftHeight =
calculateHeight(tempNode->left);

        if (rightHeight > leftHeight) {
            tempNode->height = rightHeight + 1;
        }
    }
};
```

```

        else {
            tempNode->height = leftHeight + 1;
        }
    }

node* balance(node* tempNode)
{
    countingHeight(tempNode);

    if (differenceHeight(tempNode) == 2)
    {
        if (differenceHeight(tempNode->right) < 0)
            tempNode->right = rotateright(tempNode->right);
        return turnLeft(tempNode);
    }
    if (differenceHeight(tempNode) == -2)
    {
        if (differenceHeight(tempNode->left) > 0)
            tempNode->left = turnLeft(tempNode->left);
        return rotateright(tempNode);
    }
    return tempNode;
}

node* pushNode(int key, node* tempNode) {
    if (!tempNode) {
        node* temp = new node(key);
        return temp;
    }
    if (key < tempNode->key)
        tempNode->left = pushNode(key, tempNode->left);
    else
        tempNode->right = pushNode(key, tempNode->right);
    return balance(tempNode);
}

void showVAL(node* tempNode) {
    cout << tempNode->key << " ";
    if (tempNode->left) {
        showVAL(tempNode->left);
    }
    if (tempNode->right) {
        showVAL(tempNode->right);
    }
}

node* findMin(node* tempNode)
{
    if (tempNode->left) {
        return findMin(tempNode->left);
    }
    return tempNode;
}

node* delMin(node* tempNode)
{
    if (tempNode->left == 0)
        return tempNode->right;
    tempNode->left = delMin(tempNode->left);
    return balance(tempNode);
}

```

```

node* del(node* tempNode, int key)
{
    if (!tempNode) return 0;
    if (key < tempNode->key)
        tempNode->left = del(tempNode->left, key);
    else if (key > tempNode->key)
        tempNode->right = del(tempNode->right, key);
    else
    {
        node* tempNodeQ = tempNode->left;
        node* tempNodeR = tempNode->right;
        delete tempNode;
        if (!tempNodeR) return tempNodeQ;
        node* min = findMin(tempNodeR);
        min->right = delMin(tempNodeR);
        min->left = tempNodeQ;
        return balance(min);
    }
    return balance(tempNode);
}

```

```

bool checkElem(node* tempNode, int key, bool& it_) {
    cout << "root: " << tempNode->key << endl;

    if (tempNode->key == key) {
        it_ = true;
    }
    if (tempNode->left) {
        checkElem(tempNode->left, key, it_);
    }
    if (tempNode->right) {
        checkElem(tempNode->right, key, it_);
    }

    return it_;
}

```

```

void printTree(node* tempNode, int level)
{
    if (tempNode)
    {
        printTree(tempNode->right, level + 1);

        for (int i = 0; i < level; i++) {
            cout << "  ";
        }
        cout << tempNode->key << endl;

        printTree(tempNode->left, level + 1);
    }
}

```

public:

```

Tree() {
    root = 0;
}

void push(int key) {
    root = pushNode(key, root);
}

node* getRoot() {

```

```

        return this->root;
    }

    void show() {
        showVAL(root);
    }

    void pop(int key) {
        if (key == root->key) {
            root = del(root, key);
        }
        else {
            del(root, key);
        }
    }

    bool check(int key) {
        bool it_ = false;
        if (checkElem(root, key, it_) == true) {
            return true;
        }
        else {
            return false;
        }
    }

    void showAVL() {
        printTree(root, 0);
    }

};

int main() {
    char c = 'y';
    int input = 0;
    Tree tree;

    int size;
    cout << "Input count of elements: ";
    cin >> size;

    for (int i = 0; i < size; i++)
    {
        int in;
        cin >> in;
        tree.push(in);
    }

    std::cout << "\nTree from input\n";
    tree.showAVL();
    cout << "q - for exit." << endl;
    while (c == 'y')
    {
        cout << "Input key wich you want to delete: ";
        std::cin >> input;
        if (tree.check(input)) {
            tree.pop(input);
            std::cout << "\nTree after edit\n";
            tree.showAVL();
        }
        else {
            cout << "There isn't that element which one you inputed." << endl;
            cout << "Tree: " << endl;
            tree.showAVL();
        }
    }
}

```

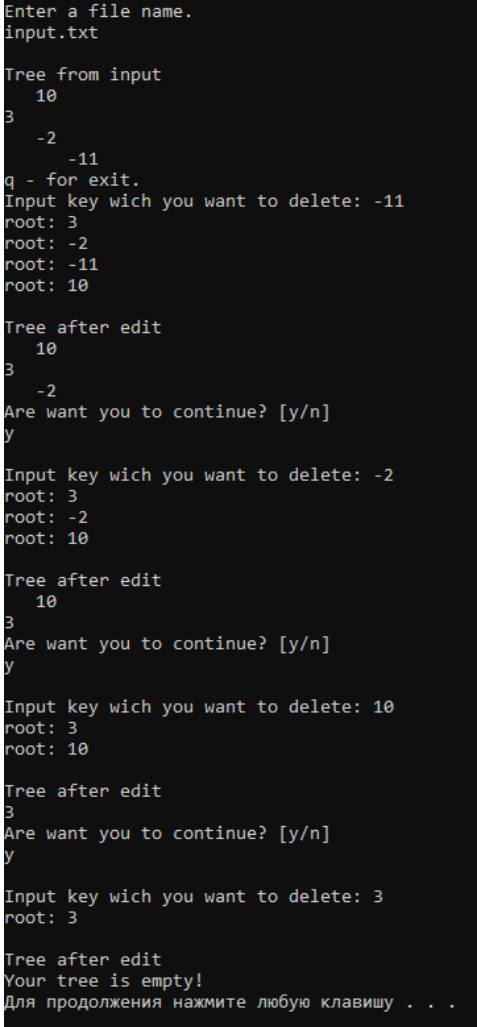


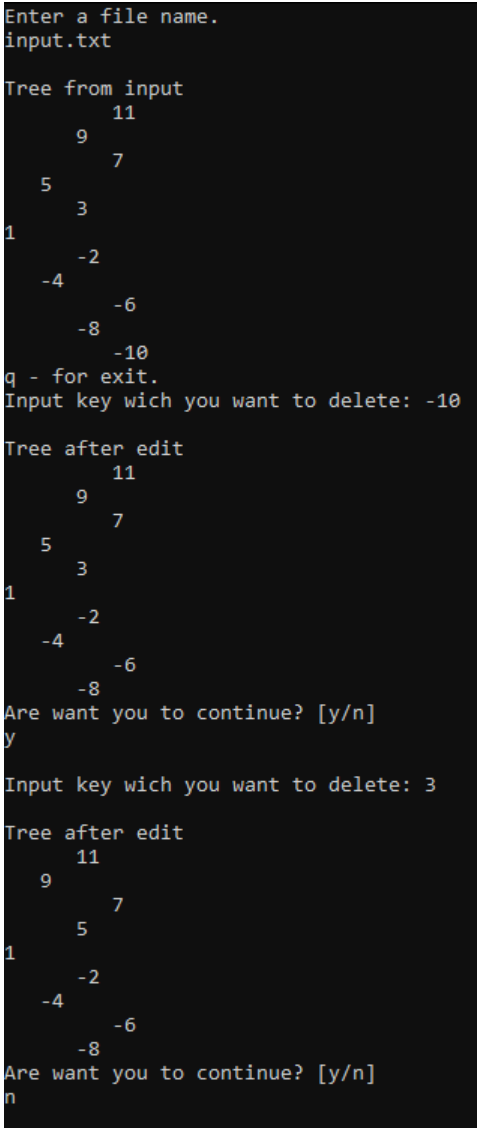
```
        if (!tree.getRoot())
        {
            cout << "Your tree is empty!" << endl;
            break;
        }
        cout << "Are want you to continue? [y/n]" << endl;
        cin >> c;
        std::cout << std::endl;
    }
    return 0;
}
```

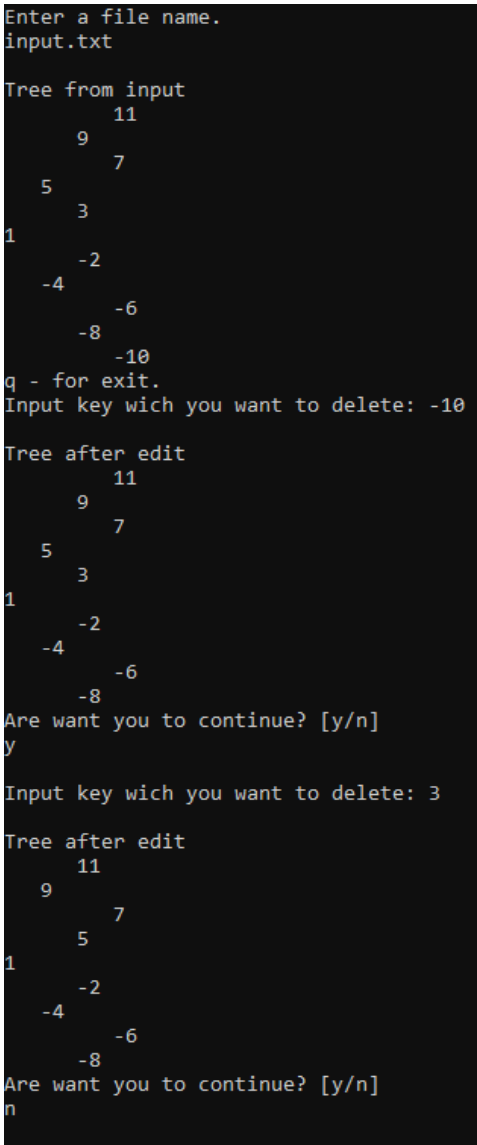
## ПРИЛОЖЕНИЕ Б

### ТЕСТИРОВАНИЕ

Таблица 1 — Примеры тестовых случаев.

№ п/п	Входные данные	Выходные данные	Комментарии
17 1.	3423 34 3221 36656 5436 31412 2314	 <p style="text-align: center;">Рисунок 1 — Пример работы программы.</p>	

2.	<p>20</p> <p>123 34 344 324 32 5432</p> <p>5452324 432 324 3444</p> <p>4324 3211 90 786 1234</p> <p>3432 1111 12 432 91</p>	 <p>The screenshot shows a terminal window with the following text:</p> <pre> Enter a file name. input.txt  Tree from input       11      /  \     9    7    /  \   5    3  / 1 -2 -4   -6   -8   -10 q - for exit. Input key wich you want to delete: -10  Tree after edit       11      /  \     9    7    /  \   5    3  / 1 -2 -4   -6   -8 Are want you to continue? [y/n] y  Input key wich you want to delete: 3  Tree after edit       11      /  \     9    7    /  \   5    3  / 1 -2 -4   -6   -8 Are want you to continue? [y/n] n </pre> <p>Рисунок 2 – Тестирование программы</p>	
----	---	--	--

3.	<p>11</p> <p>1 -2 3 -4 5 -6 7 -8 9 -10</p> <p>11</p>	 <pre> Enter a file name. input.txt  Tree from input       11      /  \     9    7    /  \   5    3  /  \ 1  -2    / \   -4 -6      / \     -8 -10 q - for exit. Input key wich you want to delete: -10  Tree after edit       11      /  \     9    7    /  \   5    3  /  \ 1  -2    / \   -4 -6      / \     -8 Are want you to continue? [y/n] y  Input key wich you want to delete: 3  Tree after edit       11      /  \     9    7    /  \   5    3  /  \ 1  -2    / \   -4 -6      / \     -8 Are want you to continue? [y/n] n </pre> <p>Рисунок 3 – Тестирование программы</p>	
----	--	---	--