МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №5

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема. Бинарное дерево поиска. AVL дерево

Выполнил студент группы ИКБО-25-22 Ракитин В.А.

Принял доцент Бузыкова Ю.С.

Москва 2023

Цель: изучить теорию и научиться программировать бинарные деревья поиска и AVL, применять алгоритмы обхода.

Ход работы

Задание 1

Код программы

Код основной программы представлен в листинге 1.1.

Листинг 1.1 – Код основной программы

```
#include <iostream>
#include <queue>
#include <string>

enum Color { RED, BLACK };

template <typename T>
struct Node {
    T key;
    Node* parent;
    Node* left;
    Node* right;
    Color color;
};

template <typename T>
class RedBlackTree {
private:
    Node<T>* root;

    void leftRotate(Node<T>* x) {
        Node<T>* y = x->right;
        x->right = y->left;

        if (y->left != nullptr) {
            y->left->parent = x;
        }

        y->parent = x->parent;

        if (x->parent == nullptr) {
            root = y;
        }
        else if (x == x->parent->left) {
            x->parent->left = y;
        }
        else {
            x->parent->right = y;
        }

        y->left = x;
        x->parent = y;
    }

    void rightRotate(Node<T>* y) {
        Node<T>* x = y->left;
        y->left = x->right;

        if (x->right != nullptr) {
```

```
            x->right->parent = y;
        }

        x->parent = y->parent;

        if (y->parent == nullptr) {
            root = x;
        }
        else if (y == y->parent->left) {
            y->parent->left = x;
        }
        else {
            y->parent->right = x;
        }

        x->right = y;
        y->parent = x;
    }

    void insertFixup(Node<T>* z) {
        while (z->parent != nullptr && z->parent->color == RED) {
            if (z->parent == z->parent->parent->left) {
                Node<T>* y = z->parent->parent->right;
                if (y != nullptr && y->color == RED) {
                    z->parent->color = BLACK;
                    y->color = BLACK;
                    z->parent->parent->color = RED;
                    z = z->parent->parent;
                }
                else {
                    if (z == z->parent->right) {
                        z = z->parent;
                        leftRotate(z);
                    }
                    z->parent->color = BLACK;
                    z->parent->parent->color = RED;
                    rightRotate(z->parent->parent);
                }
            }
            else {
                Node<T>* y = z->parent->parent->left;
                if (y != nullptr && y->color == RED) {
                    z->parent->color = BLACK;
                    y->color = BLACK;
                    z->parent->parent->color = RED;
                    z = z->parent->parent;
                }
                else {
                    if (z == z->parent->left) {
                        z = z->parent;
                        rightRotate(z);
                    }
                    z->parent->color = BLACK;
                    z->parent->parent->color = RED;
                    leftRotate(z->parent->parent);
                }
            }
        }
        root->color = BLACK;
    }

    void inorderTraversal(Node<T>* x) {
        if (x != nullptr) {
```

```
            inorderTraversal(x->left);
            std::cout << x->key << " ";
            inorderTraversal(x->right);
        }
    }

    void levelOrderTraversal() {
        if (root == nullptr) {
            return;
        }

        std::queue<Node<T>*> q;
        q.push(root);

        while (!q.empty()) {
            Node<T>* temp = q.front();
            q.pop();

            std::cout << temp->key << " ";

            if (temp->left != nullptr) {
                q.push(temp->left);
            }

            if (temp->right != nullptr) {
                q.push(temp->right);
            }
        }
    }

    int findLeafSum(Node<T>* x) {
        if (x == nullptr) {
            return 0;
        }

        if (x->left == nullptr && x->right == nullptr) {
            return x->key.length();  // assuming key is a string
        }

        return findLeafSum(x->left) + findLeafSum(x->right);
    }

    int findHeight(Node<T>* x) {
        if (x == nullptr) {
            return 0;
        }

        int leftHeight = findHeight(x->left);
        int rightHeight = findHeight(x->right);

        return std::max(leftHeight, rightHeight) + 1;
    }

public:
    RedBlackTree() : root(nullptr) {}

    void insert(T key) {
        Node<T>* z = new Node<T>{ key, nullptr, nullptr, nullptr, RED };
        Node<T>* y = nullptr;
        Node<T>* x = root;

        while (x != nullptr) {
            y = x;
```

```
            if (z->key < x->key) {
                x = x->left;
            }
            else {
                x = x->right;
            }
        }

        z->parent = y;
        if (y == nullptr) {
            root = z;
        }
        else if (z->key < y->key) {
            y->left = z;
        }
        else {
            y->right = z;
        }

        insertFixup(z);
    }

    void printInorder() {
        std::cout << "Симметричный обход: ";
        inorderTraversal(root);
        std::cout << std::endl;
    }

     double calculateLeafSum() {
        return findLeafSum(root);
    }

    int calculateHeight() {
        return findHeight(root);
    }
    double findLeavesSum() {
        return sumLeaves(root);
    }
    double sumLeaves(Node<T>* x) {
        if (x == nullptr)
            return 0;
        if (x->left == nullptr && x->right == nullptr)
            return std::stod(x->key);
        return sumLeaves(x->left) + sumLeaves(x->right);
    }
    double findAverage() {
        double sum = sumLeaves(root);
        double count = calculateLeafSum();
        if (count > 0)
            return sum / count;
        return 0;
    }
    void printTreeStructure() {
        std::cout << "Структура дерева:" << std::endl;
        printTreeStructure(root, "", false);
    }

    void printTreeStructure(Node<T>* root, const std::string& prefix, bool
isLeft) {
        if (root != nullptr) {
            std::cout << prefix;
            std::cout << (isLeft ? "|--" : "'--");
```

```
            std::cout << root->key << " (" << (root->color == RED ? "RED" :
"BLACK") << ")" << std::endl;

            printTreeStructure(root->left, prefix + (isLeft ? "|   " : "
"), true);
            printTreeStructure(root->right, prefix + (isLeft ? "|   " : "
"), false);
        }
    }
};

int main() {

    setlocale(LC_ALL, "ru");

    RedBlackTree<std::string> tree;

    std::cout << "Введите 10 элементов для вставки в Красно-Чёрное Дерево:"
<< std::endl;
    for (int i = 0; i < 10; ++i) {
        std::string element;
        std::cin >> element;
        tree.insert(element);
    }

    tree.printInorder();
    double leafSum = tree.calculateLeafSum();

    std::cout << "Сумма значений листьев: " << leafSum << std::endl;
    tree.printTreeStructure();
    // Выводим среднее арифметическое всех узлов
    std::cout << "Среднее арифметическое всех узлов: " << tree.findAverage()
<< std::endl;

    return 0;
}
```

Результаты тестирования

Результат тестирования программы персонального варианта представлен на рисунке 1.1.

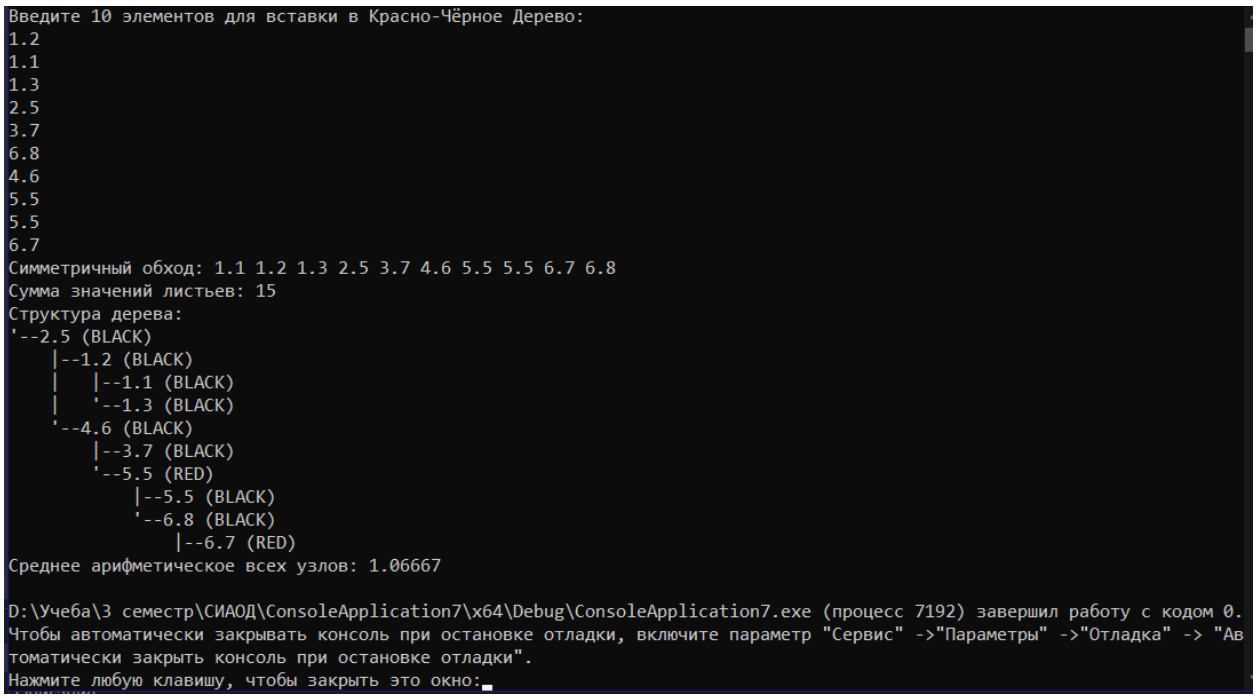

Рисунок 1.1 – Результат тестирования программы

Вывод: в ходе выполнения практической работы были получены навыки разработки бинарных деревьев поиска и AVL, применения алгоритмов обхода.