



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4
Бинарное дерево поиска
по дисциплине
«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-01-21

Луковников Д.Р.

Принял преподаватель

Туманова М.Б.

Практическая

«__»_____2022 г.

работа выполнена

«Зачтено»

«__»_____2022 г.

Москва 2022

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ	3
ХОД РАБОТЫ	4
1.1 Вставка элемента и балансировка	5
1.2 Прямо обход дерева	7
1.3 Симметричный обход дерева	7
1.4 Сумма листьев	7
1.5 Среднее арифметическое всех узлов	8
1.6 Вывод дерева	8
ТЕСТИРОВАНИЕ	10
ВЫВОДЫ	13
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	14
ПРИЛОЖЕНИЯ	15

ЦЕЛЬ РАБОТЫ

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту. Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню.

Провести полное тестирование программы на дереве размером $n=10$ элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

ХОД РАБОТЫ

Индивидуальный вариант:

- Тип значения узла – Вещественное число
- Тип дерева – Красно-чёрно дерево

Реализовать алгоритмы:

- Вставка элемента и балансировка
- Прямой обход дерева
- Симметричный обход дерева
- Найти сумму значений листьев
- Найти среднее арифметическое всех узлов

Математическая модель решения: перед тем, как начинать реализовывать методы, нам необходима реализовать бинарное красно-чёрное дерево (RB tree). Это самобалансирующееся дерево, гарантирующее логарифмический рост высоты дерева от числа узлов и позволяющее.

Свойства красно-чёрных деревьев:

- Каждый узел имеет цвет
- Корень окрашен в чёрный цвет
- Листья (Nil) окрашены в чёрный
- Каждый красный узел, имеет 2 чёрных узла. (тем не менее, у чёрного узла, могут быть чёрные потомки)
- Путь от узла к листьям содержат одинаковое кол-во чёрных узлов.

Реализация: реализация начинается с создания структуры узла, Листинг 1.1.

Листинг 1.1 – Структура узла

```
struct Node {  
    double value; // Node value  
    Node *left, *right, *parent; // Left, right and parent nodes  
    bool color; // false - black, true - red  
};
```

1.1 Вставка элемента и балансировка

Вставка в RB tree начинается так же, как и в обычном бинарном дереве, только здесь, узлы вставляются в позиции null-листьев (nil). Вставленный узел, всегда окрашивается в красный цвет. Затем идёт проверка сохранения свойства Красно-чёрного дерева, Листинг 1.2/1.3.

Листинг 1.2 – Вставка элемента

```
void add(double value) {
    /*
     * Add new node to tree
     */
    Node *newNode = new Node;
    newNode->value = value;
    newNode->left = nil;
    newNode->right = nil;
    newNode->color = true; // Red

    /*
     * Find place for new node
     */

    Node *current = root;
    Node *parent = nil;

    while (current != nil) {
        parent = current;
        if (newNode->value < current->value) {
            current = current->left;
        } else {
            current = current->right;
        }
    }
    newNode->parent = parent;
    if (parent == nil) {
        root = newNode;
    } else if (newNode->value < parent->value) {
        parent->left = newNode;
    } else {
        parent->right = newNode;
    }

    if (newNode->parent == nil) {
        newNode->color = false;
        return;
    }
    if (newNode->parent->parent == nil) {
        return;
    }
    fixTree(newNode);

    if (root == nil) {
        root = newNode;
        root->color = false; // Black
    }
}
```

Листинг 1.3 – Балансировка дерева

```
void fixTree(Node *node) {
    /*
     * Fix tree after adding new node
     */
    Node *parent = nil;
    Node *grandparent = nil;

    while ((node != root) && (node->color != false) && (node->parent->color
== true)) {
        parent = node->parent;
        grandparent = node->parent->parent;

        if (parent == grandparent->left) {
            Node *uncle = grandparent->right;

            if (uncle->color == true) {
                grandparent->color = true;
                parent->color = false;
                uncle->color = false;
                node = grandparent;
            } else {
                if (node == parent->right) {
                    leftRotate(parent);
                    node = parent;
                    parent = node->parent;
                }

                rightRotate(grandparent);
                swap(parent->color, grandparent->color);
                node = parent;
            }
        } else {
            Node *uncle = grandparent->left;

            if (uncle->color == true) {
                grandparent->color = true;
                parent->color = false;
                uncle->color = false;
                node = grandparent;
            } else {
                if (node == parent->left) {
                    rightRotate(parent);
                    node = parent;
                    parent = node->parent;
                }

                leftRotate(grandparent);
                swap(parent->color, grandparent->color);
                node = parent;
            }
        }
    }
    root->color = false;
}
```

1.2 Прямо обход дерева

Прямой обход дерева, это обход сначала корня, затем левого, затем правого узла, Листинг 2.1.

Листинг 2.1 – Прямой обход

```
void showTree(Node *node = nullptr) {
    if (node == nullptr) {
        node = root;
    }
    if (node != nil) {
        cout << node->value << " ";
        showTree(node->left);
        showTree(node->right);
    }
}
```

1.3 Симметричный обход дерева

Симметричный обход подразумевает обход сначала левого узла, затем корня, а затем правого, Листинг 3.1

Листинг 3.1 – Симметричный обход

```
void showTreeSymmetric(Node *node = nullptr) {
    if (node == nullptr) {
        node = root;
    }
    if (node != nil) {
        showTreeSymmetric(node->left);
        cout << node->value << " ";
        showTreeSymmetric(node->right);
    }
}
```

1.4 Сумма листьев

Рекурсивно проходим по всему дереву, проверяя, является ли узел листом, если да увеличиваем сумму, Листинг 4.1

Листинг 4.1 – Сумма листьев

```
double sumLeaves(Node *node = nullptr) {
    if (node == nullptr) {
        node = root;
    }
    if (node != nil) {
        if (node->left == nil && node->right == nil) {
            return node->value;
        }
        return sumLeaves(node->left) + sumLeaves(node->right);
    }
    return 0;
}
```

1.5 Среднее арифметическое всех узлов

Считаем сумму всех узлов, считаем количество узлов и делим одно на другое, Листинг 5.1

Листинг 5.1 – Среднее арифметическое узлов

```
// Количество узлов
int countNodes(Node *node = nullptr) {
    if (node == nullptr) {
        node = root;
    }
    if (node != nil) {
        return 1 + countNodes(node->left) + countNodes(node->right);
    }
    return 0;
}

// Сумма всех узлов
double sumNodes(Node *node = nullptr) {
    if (node == nullptr) {
        node = root;
    }
    if (node != nil) {
        return node->value + sumNodes(node->left) + sumNodes(node->right);
    }
    return 0;
}

// Среднее арифметическое всех узлов
double averageNodes(Node *node = nullptr) {
    if (node == nullptr) {
        node = root;
    }
    if (node != nil) {
        return sumNodes(node) / countNodes(node);
    }
    return 0;
}
```

1.6 Вывод дерева

В целях эффективности проверки и отладки кода, написан дополнительный метод вывода дерева в виде графа с помощью языка Grapviz, Листинг 6.1/6.2.

Листинг 6.1 – Вывод графа

```
void showGraphviz() {
    cout << "digraph G {" << endl;
    showTreeGraphviz();
    cout << "}" << endl;
}
```


Листинг 6.2 – Рекурсивная функция вывода графа

```
void showTreeGraphiz(Node *node = nullptr) {
    /*
     * Show tree Graphiz
     */
    if (node == nullptr) {
        node = root;
    }
    if (node != nil) {
        if (node->left != nil) {
            cout << node->value << " [color=" << (node->color ? "red" :
"black") << "];" << endl;
            cout << node->value << " -> " << node->left->value << ";" <<
endl;
        }
        if (node->right != nil) {
            cout << node->value << " [color=" << (node->color ? "red" :
"black") << "];" << endl;
            cout << node->value << " -> " << node->right->value << ";" <<
endl;
        }
        showTreeGraphiz(node->left);
        showTreeGraphiz(node->right);
    }
}
```

ТЕСТИРОВАНИЕ

Для начала добавим в дерево 10 элементов, рисунок 1.

```
1. Добавление элемента
2. Показать дерево (Прямой ход)
3. Симметричный обход
4. Сумма листьев
5. Среднее арифметическое всех узлов
6. Показать дерево Graphviz
7. Выход
Выберите пункт меню:1
1
Введите значение:13.4
13.4
```

Рисунок 1 – Вставка элементов

Для удобства проверки сгенерируем граф и отрисуем его, в силу того, что это граф, а не дерево, для узлов, которые имеют 1-го `nil` наследника, в графе не будет отображаться левый или правый этот наследник, Рисунок 2-3.

```
Выберите пункт меню:6
6
digraph G {
12.8 [color=black];
12.8 -> 3.5;
12.8 [color=black];
12.8 -> 24.6;
3.5 [color=red];
3.5 -> 2.5;
3.5 [color=red];
3.5 -> 6.8;
2.5 [color=black];
2.5 -> 0.2;
24.6 [color=red];
24.6 -> 16.4;
24.6 [color=red];
24.6 -> 45.1;
16.4 [color=black];
16.4 -> 13.4;
16.4 [color=black];
16.4 -> 23.1;
}
```

Рисунок 2 – Отрисовка графа

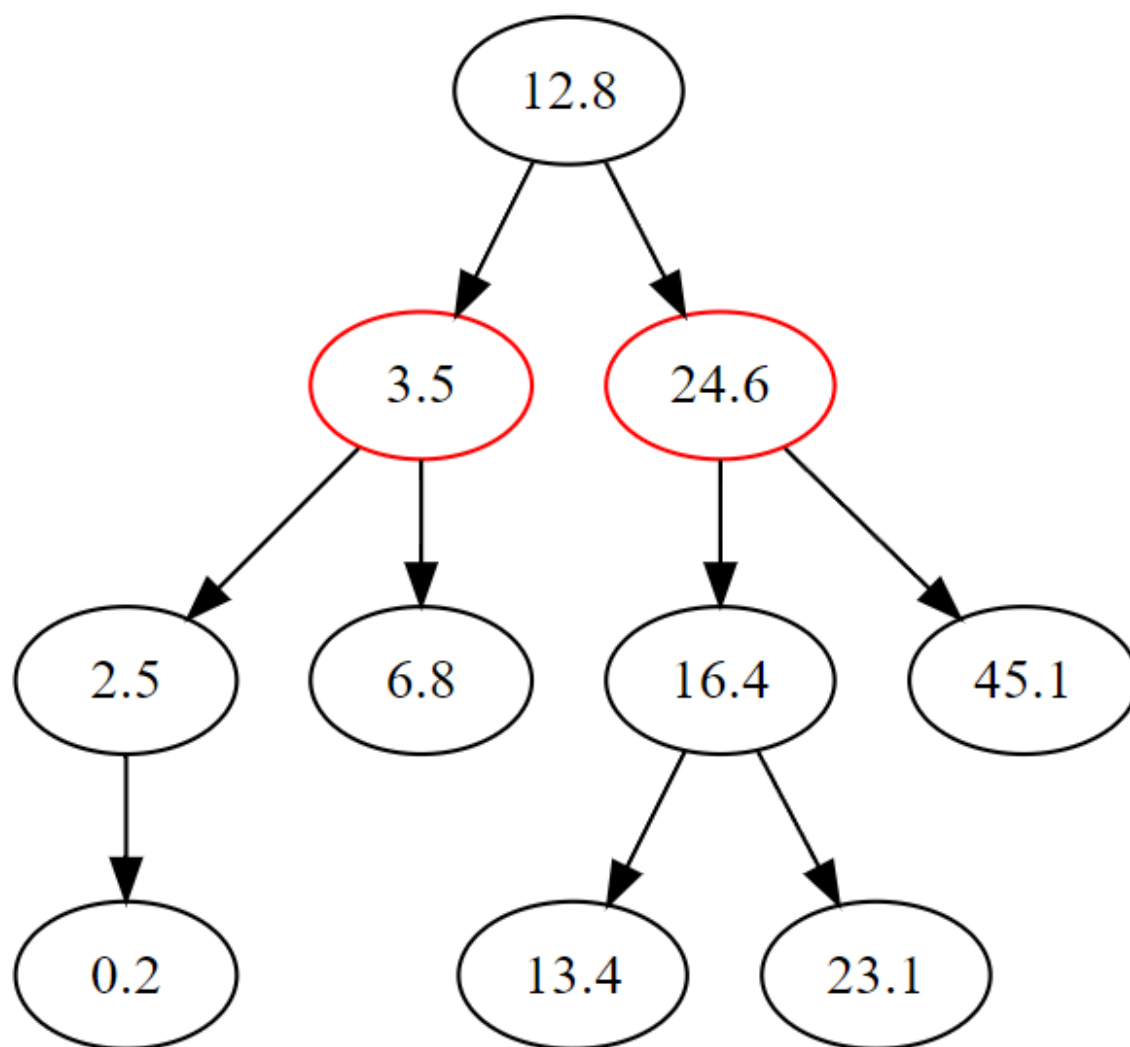


Рисунок 3 – Дерево в виде графа

Далее проверим прямой и симметричный обход, Рисунок 4-5.

```

1. Выход
Выберите пункт меню:2
2
12.8 3.5 2.5 0.2 6.8 24.6 16.4 13.4 23.1 45.1
  
```

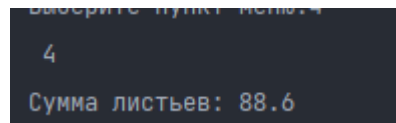
Рисунок 4 – Прямой обход

```

Выберите пункт меню:3
3
0.2 2.5 3.5 6.8 12.8 13.4 16.4 23.1 24.6 45.1
  
```

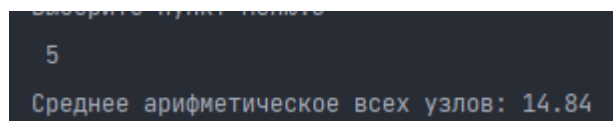
Рисунок 5 – Симметричный обход

Сравнив вывод с графом, видим, что всё работает верно. Сумма листьев и среднее арифметическое всех узлов, Рисунок 6-7.



```
Введите пункт меню:
4
Сумма листьев: 88.6
```

Рисунок 6 – Сумма листьев



```
Введите пункт меню:
5
Среднее арифметическое всех узлов: 14.84
```

Рисунок 7 – Среднее арифметическое всех узлов

Все значения выведены верно, программа работает корректно.

ВЫВОДЫ

При выполнении работы были получены навыки реализации красно-чёрного бинарного дерева поиска и создание процедур для работы с данным деревом. Программа была проверена на работоспособность и полностью протестирована.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
2. Документация по языку C++ [Электронный ресурс]. URL: <https://docs.microsoft.com/ruru/cpp/cpp/> (дата обращения 01.09.2021).
3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: <https://online-edu.mirea.ru/course/view.php?id=4020> (дата обращения 01.09.2021).

ПРИЛОЖЕНИЯ

Приложение А – Исходный код программы Красно-чёрного бинарного дерева

Приложение А

Исходный код программы Красно-чёрного бинарного дерева

Листинг 7.1 – main.cpp

```
/*
 * Составить программу создания двоичного дерева поиска и реализовать
 * процедуры для работы с деревом согласно варианту.
 * Процедуры оформить в виде самостоятельных режимов работы созданного
 * дерева. Выбор режимов производить с помощью пользовательского
 * (иерархического ниспадающего) меню.
 * Провести полное тестирование программы на дереве размером n=10
 * элементов, сформированном вводом с клавиатуры. Тест-примеры определить
 * самостоятельно. Результаты тестирования в виде скриншотов экранов
 * включить в отчет по выполненной работе.
 * Сделать выводы о проделанной работе, основанные на полученных
 * результатах.
 * Оформить отчет с подробным описанием созданного дерева, принципов
 * программной реализации алгоритмов работы с деревом, описанием текста
 * исходного кода и проведенного тестирования программы.
 *
 * Вариант 23
 * Тип значения узла - Вещественное число.
 * Тип дерева - Красно-черное дерево.
 * Процедуры:
 *     1. Вставка узла в дерево и балансировка
 *     2. Прямой обход дерева
 *     3. Симметричный обход дерева
 *     4. Найти сумму значений листьев дерева
 *     5. Найти среднее арифметическое значений всех узлов
 *
 * Правила:
 *     1. Корень дерева всегда черный.
 *     2. Все листья дерева (NIL) черные.
 *     3. Если узел красный, то оба его потомка черные.
 *     4. Глубина в черных узлах одинакова.
 *
 * https://dreampuf.github.io/GraphvizOnline/
 */

#include <iostream>
#include <vector>

using namespace std;

struct Node {
    double value; // Node value
    Node *left, *right, *parent; // Left, right and parent nodes
    bool color; // false - black, true - red
};

class Tree {
    Node *root;
    Node *nil;

public:
    Tree() {
        /*
         * Constructor for Tree class
         */
    }
};
```


Продолжение Листинг 7.1

```
    nil = new Node;
    nil->color = false;
    nil->left = nil;
    nil->right = nil;
    nil->parent = nil;
    root = nil;
}

void add(double value) {
    /*
     * Add new node to tree
     */
    Node *newNode = new Node;
    newNode->value = value;
    newNode->left = nil;
    newNode->right = nil;
    newNode->color = true; // Red

    /*
     * Find place for new node
     */

    Node *current = root;
    Node *parent = nil;

    while (current != nil) {
        parent = current;
        if (newNode->value < current->value) {
            current = current->left;
        } else {
            current = current->right;
        }
    }

    newNode->parent = parent;
    if (parent == nil) {
        root = newNode;
    } else if (newNode->value < parent->value) {
        parent->left = newNode;
    } else {
        parent->right = newNode;
    }

    if (newNode->parent == nil) {
        newNode->color = false;
        return;
    }

    if (newNode->parent->parent == nil) {
        return;
    }

    fixTree(newNode);

    if (root == nil) {
        root = newNode;
        root->color = false; // Black
    }
}
```

Продолжение Листинг 7.1

```
void fixTree(Node *node) {
    /*
     * Fix tree after adding new node
     */
    Node *parent = nil;
    Node *grandparent = nil;

    while ((node != root) && (node->color != false) && (node->parent-
>color == true)) {
        parent = node->parent;
        grandparent = node->parent->parent;

        if (parent == grandparent->left) {
            Node *uncle = grandparent->right;

            if (uncle->color == true) {
                grandparent->color = true;
                parent->color = false;
                uncle->color = false;
                node = grandparent;
            } else {
                if (node == parent->right) {
                    leftRotate(parent);
                    node = parent;
                    parent = node->parent;
                }

                rightRotate(grandparent);
                swap(parent->color, grandparent->color);
                node = parent;
            }
        } else {
            Node *uncle = grandparent->left;

            if (uncle->color == true) {
                grandparent->color = true;
                parent->color = false;
                uncle->color = false;
                node = grandparent;
            } else {
                if (node == parent->left) {
                    rightRotate(parent);
                    node = parent;
                    parent = node->parent;
                }

                leftRotate(grandparent);
                swap(parent->color, grandparent->color);
                node = parent;
            }
        }
    }
    root->color = false;
}

void leftRotate(Node *node) {
    Node *right = node->right;
    node->right = right->left;

    if (node->right != nil) {
        node->right->parent = node;
    }
}
```

```
    }

    right->parent = node->parent;

    if (node->parent == nil) {
        root = right;
    } else if (node == node->parent->left) {
        node->parent->left = right;
    } else {
        node->parent->right = right;
    }

    right->left = node;
    node->parent = right;
}

void rightRotate(Node *node) {
    Node *left = node->left;
    node->left = left->right;

    if (node->left != nil) {
        node->left->parent = node;
    }

    left->parent = node->parent;

    if (node->parent == nil) {
        root = left;
    } else if (node == node->parent->left) {
        node->parent->left = left;
    } else {
        node->parent->right = left;
    }

    left->right = node;
    node->parent = left;
}

void showGraphiz() {
    cout << "digraph G {" << endl;
    showTreeGraphiz();
    cout << "}" << endl;
}

// Прямой обход
void showTree(Node *node = nullptr) {
    if (node == nullptr) {
        node = root;
    }
    if (node != nil) {
        cout << node->value << " ";
        showTree(node->left);
        showTree(node->right);
    }
}

// Симметричный обход
void showTreeSymmetric(Node *node = nullptr) {
    if (node == nullptr) {
        node = root;
    }
}
```

Продолжение Листинг 7.1

```
        if (node != nil) {
            showTreeSymmetric(node->left);
            cout << node->value << " ";
            showTreeSymmetric(node->right);
        }
    }

    // Сумма листьев
    double sumLeaves(Node *node = nullptr) {
        if (node == nullptr) {
            node = root;
        }
        if (node != nil) {
            if (node->left == nil && node->right == nil) {
                return node->value;
            }
            return sumLeaves(node->left) + sumLeaves(node->right);
        }
        return 0;
    }

    // Среднее арифметическое всех узлов
    double averageNodes(Node *node = nullptr) {
        if (node == nullptr) {
            node = root;
        }
        if (node != nil) {
            return sumNodes(node) / countNodes(node);
        }
        return 0;
    }
}

private:
    // Количество узлов
    int countNodes(Node *node = nullptr) {
        if (node == nullptr) {
            node = root;
        }
        if (node != nil) {
            return 1 + countNodes(node->left) + countNodes(node->right);
        }
        return 0;
    }

    // Сумма всех узлов
    double sumNodes(Node *node = nullptr) {
        if (node == nullptr) {
            node = root;
        }
        if (node != nil) {
            return node->value + sumNodes(node->left) + sumNodes(node->right);
        }
        return 0;
    }

    // Вывод дерева в формате Graphviz
    void showTreeGraphviz(Node *node = nullptr) {
        /*
         * Show tree Graphviz
         */
    }
```

Продолжение Листинг 7.1

```
        if (node == nullptr) {
            node = root;
        }
        if (node != nil) {
            if (node->left != nil) {
                cout << node->value << " [color=" << (node->color ? "red" :
"black") << "];" << endl;
                cout << node->value << " -> " << node->left->value << ";" <<
endl;
            }
            if (node->right != nil) {
                cout << node->value << " [color=" << (node->color ? "red" :
"black") << "];" << endl;
                cout << node->value << " -> " << node->right->value << ";" <<
endl;
            }
            showTreeGraphiz(node->left);
            showTreeGraphiz(node->right);
        }
    }
};

int main() {
    system("chcp 65001");

    Tree tree;
    vector<double> values = {
        1.3, 2.3, 3.3, 4.3, 5.3, 6.3, 7.3, 8.3, 9.3, 10.3,
//        11.3, 12.3, 13.3, 14.3, 15.3, 16.3, 17.3, 18.3, 19.3, 20.3,
//        21.3, 22.3, 23.3, 24.3, 25.3, 26.3, 27.3, 28.3, 29.3, 30.3,
//        31.3, 32.3, 33.3, 34.3, 35.3, 36.3, 37.3, 38.3, 39.3, 40.3,
    };
    //    // Перемешать значения
    //    for (int i = 0; i < values.size(); i++) {
    //        int index = rand() % values.size();
    //        double temp = values[i];
    //        values[i] = values[index];
    //        values[index] = temp;
    //    }
    //    for (double value: values) {
    //        tree.add(value);
    //    }

    // Текстовое меню
    int choice = 0;
    while (choice != 7) {
        cout << "1. Добавление элемента" << endl;
        cout << "2. Показать дерево (Прямой ход)" << endl;
        cout << "3. Симметричный обход" << endl;
        cout << "4. Сумма листьев" << endl;
        cout << "5. Среднее арифметическое всех узлов" << endl;
        cout << "6. Показать дерево Graphiz" << endl;
        cout << "7. Выход" << endl;
        cout << "Выберите пункт меню: ";
        cin >> choice;
        switch (choice) {
            case 1: {
                double value;
                cout << "Введите значение: ";
```

Продолжение Листинг 7.1

```
        cin >> value;
        tree.add(value);
        break;
    }
    case 2: {
        tree.showTree();
        cout << endl;
        break;
    }
    case 3: {
        tree.showTreeSymmetric();
        cout << endl;
        break;
    }
    case 4: {
        cout << "Сумма листьев: " << tree.sumLeaves() << endl;
        break;
    }
    case 5: {
        cout << "Среднее арифметическое всех узлов: " <<
tree.averageNodes() << endl;
        break;
    }
    case 6: {
        tree.showGraphiz();
        break;
    }
    case 7: {
        break;
    }
    default: {
        cout << "Неверный пункт меню" << endl;
        break;
    }
}

return 0;
}
```