|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | |
|  | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | |
| по лабораторной работе №1.2  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | |  |  | Студент гр. Б9123-09.03.04прогин(2) | | | |
|  |  |  | | Д.А. Долгорук | |
|  | | |  |  | (подпись) | |  | |
|  | | | | | | | | |
|  | | |  |  | Преподаватель | | |
|  | | |  |  | Ассистент | | | |
|  |  |  |  |  | (Ученая степень, должность) |  | А.А Шулятьев | |
|  |  |  |  |  | (подпись) |  | (И.О. Фамилия) | |
| г. Владивосток | | | | | | | | |
| 2024 | | | | | | | | |

# Неформальная постановка задачи

Реализовать пакет подпрограмм для работы с AVL деревом, ключ в котором представлен в виде структуры - номер группы (1 поле - бакалавриат/магистратура/специалитет (1 буква, соответственно: B/M/S), 2 поле - номер (4 цифры)).

Основные операции:

1. Инициализация (пустого дерева)

2. Добавление нового элемента

3. Удаление заданного элемента с заменой на минимальный правый элемент дерева (при полном совпадении ключа и номера строки)

4. Поиск заданного элемента

5. Печать

6. Прямой обход AVL дерева

7. Освобождение памяти (удаление всего дерева)

Входные данные: текстовый файл, каждая из строк которого содержит данные заданной предметной области - номер группы (1 поле - бакалавриат/магистратура/специалитет (1 буква, соответственно: B/M/S), 2 поле - номер (4 цифры)).

Выходные данные: текстовый файл, строки которого содержат данные входного файла в порядке заданного обхода дерева поиска, в нашем случае в прямом обходе.

1. Загрузить данные из текстового файла в дерево поиска. При загрузке вызывать процедуру добавления с проверкой ключа на корректность. Список номеров строк входного файла, содержащих заданный ключ, хранить в цепочке (списке) задания 1.1.

2. Вызвать подпрограммы работы с деревом.

3. Сохранить итоговое дерево в текстовый файл в порядке заданного обхода.

# Описание типа, спецификация подпрограмм и тестирование программы

1. **Структура GroupNumber**

*«struct GroupNumber {*

*char type; // Тип (бакалавриат 'B', магистратура 'M', специалитет 'S')*

*int number; // Номер группы (4-значный)*

*};»*

Описание:

Структура для хранения информации о группе. Поле type указывает на тип образования, а поле number — на номер группы, который должен быть 4-значным числом.

Входные данные: Буквы и цифры

Выходные данные: Объект структуры GroupNumber.

Тестовые ситуации:

Корректные данные:

Ввод: GroupNumber group = {'B', 1234}; (В файле: B1234 или при вводе через консоль)

Ожидаемый результат: создаётся объект GroupNumber с полями type = 'B' и number = 1234.

Ввод: GroupNumber group = {'S', 5687 };( в файле: S5687 или при вводе через консоль)

Ожидаемый результат: создаётся объект GroupNumber с полями type = 'S' и number = 5687.

Ввод: GroupNumber group = {‘M', 8898}; (В файле: M8898 или при вводе через консоль)

Ожидаемый результат: создается объект GroupNumber с полями type = 'M' и number = 8898.

Некорректные данные:

Ввод: GroupNumber group = {'X', 1234}; (в файле X1234 или при вводе через консоль)

Ожидаемый результат: Ошибка! Некорректный ввод. Так как type должен быть 'B', 'M' или 'S'.

Ввод: GroupNumber group = {‘B', 123};( в файле B123 или при вводе через консоль)

Ожидаемый результат: Ошибка! Некорректный ввод. Так как number должен быть 4-значным.

Ввод: GroupNumber group = {‘пробел', 1235};( в файле 1235 или при вводе через консоль)

Ожидаемый результат: Ошибка! Некорректный ввод. Так как type должен быть 'B', 'M' или 'S'.

Ввод: GroupNumber group = {‘B', пробел};( в файле B или при вводе через консоль)

Ожидаемый результат: Ошибка! Некорректный ввод. Так как number должен быть 4-значным.

1. **Структура TreeNode**

*«struct TreeNode {*

*GroupNumber key; // Ключ узла*

*TreeNode\* left; // Указатель на левое поддерево*

*TreeNode\* right; // Указатель на правое поддерево*

*int balanceFactor; // Коэффициент балансировки*

*std::vector<int> lineNumbers; // Список номеров строк, где встречался элемент*

*TreeNode(GroupNumber k, int lineNumber) : key(k), left(nullptr), right(nullptr), balanceFactor(0) {*

*lineNumbers.push\_back(lineNumber);*

*}*

*};»*

Описание:

Узел дерева, содержащий ключ и указатели на дочерние узлы. Также хранит коэффициент балансировки и список номеров строк, где встречался элемент.

Входные данные: Буквы и цифры

Выходные данные: Объект структуры TreeNode.

Тестовые ситуации:

Ввод: TreeNode node({ 'B', 1234 }, 1);

Ожидаемый результат: Узел создается с key и lineNumbers = {1}.

Ввод: TreeNode node({ 'B', 1234 }, 1); (добавление повторно)

Ожидаемый результат: Узел не дублируется остается без изменений, lineNumbers записывается {1,2}.

1. **Класс AVLTree**
   1. **Конструктор AVLTree()**

*«AVLTree();»*

Описание:

Создает пустое AVL-дерево.

Входные данные: Н/Д

Выходные данные: Пустой объект класса AVLTree.

Тестовые ситуации:

Ввод: AVLTree tree;

Ожидаемый результат: Дерево инициализировано, root указывает на nullptr.

* 1. **Метод void insert(GroupNumber key, int lineNumber)**

*«void insert(GroupNumber key, int lineNumber);2*

Описание:

Добавляет новый элемент в дерево.

Входные данные:

key — ключ группы (объект GroupNumber),

lineNumber — номер строки (целое число).

Выходные данные: Объект класса AVLTree с добавленным узлом.

Тестовые ситуации:

Ввод: tree.insert({ 'B', 1111 }, 1);

Ожидаемый результат: Узел B1111 добавляется в корень дерева.

Ввод: tree.insert({ 'S', 5687 }, 2); (дерево уже содержит B1111)

Ожидаемый результат: Узел S5687 добавляется к правому поддереву.

Ввод: tree.insert({ 'B', 1111 }, 3); (дерево содержит B1111)

Ожидаемый результат: Узел B1111 сохраняет номер строки 1 и добавляет 3.

Ввод: tree.insert({ 'X', 1111 }, 1);

Ожидаемый результат: Ошибка, так как type некорректен.

Ввод: tree.insert({ 'B', 111 }, 1);

Ожидаемый результат: Ошибка, так как number должен быть 4-значным.

* 1. **Метод void remove(GroupNumber key)**

*«void remove(GroupNumber key);»*

Описание:

Удаляет элемент из дерева.

Входные данные: key — ключ группы для удаления (объект GroupNumber).

Выходные данные: Объект класса AVLTree без удаленного узла.

Тестовые ситуации:

Ввод: tree.remove({ 'B', 1111 }); (дерево содержит B1111)

Ожидаемый результат: Дерево становится пустым.

Ввод: tree.remove({ 'S', 5687 }); (дерево содержит только B1111)

Ожидаемый результат: Дерево остается без изменений.

Ввод: tree.remove({ 'B', 1111 }); (дерево содержит B1111 и S5687)

Ожидаемый результат: Узел S5687 остается в дереве и становится на место корня.

Ввод: tree.remove({ 'M', 5687}); (дерево содержит B1111 и S5687 и M5689, где M5687 – корень)

Ожидаемый результат: Узел M5687 удаляется, а на место него становится элемент минимальный справа, в нашем случае это элемент S5687

Ввод: tree.remove({ 'X', 5687 });

Ожидаемый результат: Ошибка, так как type некорректен.

* 1. **Метод TreeNode\* search(GroupNumber key)**

*«TreeNode\* search(GroupNumber key);»*

Описание:

Ищет элемент в дереве.

Входные данные: key — ключ группы для поиска (объект GroupNumber).

Выходные данные: Указатель на найденный узел или nullptr, если не найден.

Тестовые ситуации:

Ввод: tree.search({ 'S', 5687 }); (дерево содержит B1111 и S5687)

Ожидаемый результат: возвращается указатель на узел S5687.

Ввод: tree.search({ 'M', 8898 }); (дерево содержит B1111)

Ожидаемый результат: возвращается nullptr.

Ввод: tree.search({ 'X', 8898 });

Ожидаемый результат: Ошибка, так как type некорректен.

* 1. **Метод void print() const**

*«void print() const;»*

Описание:

Печатает дерево.

Входные данные: (список элементов в определенном порядке)

Выходные данные: (вывод дерева в консоль – элементы расположены в виде дерева).

Тестовые ситуации:

Ввод: tree.print(); (пустое дерево)

Ожидаемый результат: (пустой вывод)

Ввод: tree.print(); (дерево содержит B1111 и S5687 и M8989, где M8989 корень)

Ожидаемый результат:

S5687

M8989

B1111

Ввод: tree.print(); (дерево содержит B1111, где B1111 корень)

Ожидаемый результат:

B1111

Ввод: tree.print(); (дерево содержит B1111 и S5687, где B1111 корень)

Ожидаемый результат:

S5687

B1111

* 1. **Метод void preOrderTraversal(std::ostream& out)**

*«void preOrderTraversal(std::ostream& out);»*

Описание:

Выполняет прямой обход дерева и выводит результат.

Входные данные: Элементы в определённом порядке.

Выходные данные: Элементы в порядке прямого обхода.

Тестовые ситуации:

Ввод: tree.preOrderTraversal(out); (пустое дерево)

Ожидаемый результат: (пустой вывод)

Ввод: tree.preOrderTraversal(out); (дерево содержит B1111 и S5687, где B1111 – корень)

Ожидаемый результат:

B1111

S5687

Ввод: tree.preOrderTraversal(out); (дерево содержит B1111 и S5687, M8989 где M8989– корень)

Ожидаемый результат:

M8989

B1111

S5687

* 1. **Метод void clear()**

*«void clear();»*

Описание:

Освобождает память, занятую деревом.

Входные данные: Дерево с элементами или пустое

Выходные данные: - (Удаление всего дерева)

Тестовые ситуации:

Очистка пустого дерева:

Ввод: Н/Д

Ожидаемый результат: (пустой вывод)

Очистка дерева с элементами:

Ввод: Дерево: B1111, S5687, M8898

Ожидаемый результат: Дерево очищено, root указывает на nullptr.

# Текст программы

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <algorithm>

#include <iomanip> // Для std::setw

// Определяем структуру для хранения информации о группе

struct GroupNumber {

char type; // Тип (бакалавриат 'B', магистратура 'M', специалитет 'S')

int number; // Номер группы (4-значный)

};

// Определяем структуру для узла дерева

struct TreeNode {

GroupNumber key; // Ключ узла

TreeNode\* left; // Указатель на левое поддерево

TreeNode\* right; // Указатель на правое поддерево

int balanceFactor; // Коэффициент балансировки

std::vector<int> lineNumbers; // Список номеров строк, где встречался элемент

// Конструктор для инициализации узла

TreeNode(GroupNumber k, int lineNumber) : key(k), left(nullptr), right(nullptr), balanceFactor(0) {

lineNumbers.push\_back(lineNumber);

}

};

// Определяем класс для AVL-дерева

class AVLTree {

public:

AVLTree() : root(nullptr) {} // Конструктор, инициализирующий корень дерева как nullptr

void insert(GroupNumber key, int lineNumber) { // Метод для вставки нового элемента

root = insert(root, key, lineNumber); // Вставляем элемент в дерево, начиная с корня

}

void remove(GroupNumber key) { // Метод для удаления элемента

root = remove(root, key); // Удаляем элемент из дерева, начиная с корня

}

TreeNode\* search(GroupNumber key) { // Метод для поиска элемента

return search(root, key); // Ищем элемент, начиная с корня

}

void print() const { // Метод для печати дерева

int height = getHeight(root);

print(root, 0, height); // Печатаем дерево, начиная с корня

}

void preOrderTraversal(std::ostream& out) { // Метод для прямого обхода дерева

preOrderTraversal(root, out); // Обходим дерево, начиная с корня

}

void clear() { // Метод для очистки дерева

clear(root); // Освобождаем память, начиная с корня

root = nullptr; // Устанавливаем корень как nullptr

}

// Метод для загрузки данных из файла

void loadFromFile(const std::string& filename) {

std::ifstream inFile(filename); // Открываем файл для чтения

if (!inFile) { // Проверяем, удалось ли открыть файл

std::cerr << "Ошибка при открытии файла " << filename << std::endl; // Сообщаем об ошибке

return; // Выходим из метода

}

std::string line; // Переменная для хранения строки из файла

int lineNumber = 0; // Счетчик строк

while (std::getline(inFile, line)) { // Читаем файл построчно

lineNumber++; // Увеличиваем номер строки

if (line.length() != 5 ||

(line[0] != 'B' && line[0] != 'M' && line[0] != 'S') ||

!std::isdigit(line[1]) || !std::isdigit(line[2]) || !std::isdigit(line[3]) || !std::isdigit(line[4])) {

std::cerr << "Ошибка в строке " << lineNumber << ": некорректный формат строки." << std::endl;

continue; // Пропускаем некорректную строку

}

GroupNumber group; // Создаем переменную для хранения группы

group.type = line[0]; // Извлекаем тип группы из первой буквы строки

group.number = std::stoi(line.substr(1, 4)); // Извлекаем 4-цифровый номер группы

// Проверка на значение номера группы

if (group.number < 1000 || group.number > 9999) {

std::cerr << "Ошибка в строке " << lineNumber << ": номер группы должен быть 4-значным числом." << std::endl;

continue; // Пропускаем некорректную строку

}

insert(group, lineNumber); // Добавляем элемент в дерево

}

inFile.close(); // Закрываем файл

}

// Метод для сохранения данных в файл

void saveToFile(const std::string& filename) {

std::ofstream outFile(filename); // Открываем файл для записи

if (!outFile) { // Проверяем, удалось ли открыть файл

std::cerr << "Ошибка при открытии файла " << filename << std::endl; // Сообщаем об ошибке

return; // Выходим из метода

}

preOrderTraversal(outFile); // Записываем дерево в файл

outFile.close(); // Закрываем файл

}

private:

TreeNode\* root; // Указатель на корень дерева

// Метод для получения коэффициента балансировки узла

int balanceFactor(TreeNode\* node) {

return node ? (height(node->left) - height(node->right)) : 0; // Разница высот левого и правого поддеревьев

}

// Метод для получения высоты узла

int height(TreeNode\* node) {

if (!node) return 0; // Если узел пуст, высота 0

return std::max(height(node->left), height(node->right)) + 1; // Высота узла

}

// Метод для правого вращения узла

TreeNode\* rotateRight(TreeNode\* y) {

TreeNode\* x = y->left; // Сохраняем левое поддерево

TreeNode\* T2 = x->right; // Сохраняем правое поддерево левого узла

x->right = y; // Поворачиваем узел вправо

y->left = T2; // Присоединяем правое поддерево к левому узлу

// Обновляем коэффициенты балансировки

y->balanceFactor = balanceFactor(y);

x->balanceFactor = balanceFactor(x);

return x; // Возвращаем новый корень поддерева

}

// Метод для левого вращения узла

TreeNode\* rotateLeft(TreeNode\* x) {

TreeNode\* y = x->right; // Сохраняем правое поддерево

TreeNode\* T2 = y->left; // Сохраняем левое поддерево правого узла

y->left = x; // Поворачиваем узел влево

x->right = T2; // Присоединяем левое поддерево к правому узлу

// Обновляем коэффициенты балансировки

x->balanceFactor = balanceFactor(x);

y->balanceFactor = balanceFactor(y);

return y; // Возвращаем новый корень поддерева

}

// Метод для вставки узла в дерево

TreeNode\* insert(TreeNode\* node, GroupNumber key, int lineNumber) {

if (!node) return new TreeNode(key, lineNumber); // Если узел пуст, создаем новый узел

// Сравниваем ключи и вставляем в соответствующее поддерево

if (key.type < node->key.type || (key.type == node->key.type && key.number < node->key.number)) {

node->left = insert(node->left, key, lineNumber); // Вставляем в левое поддерево

}

else if (key.type > node->key.type || (key.type == node->key.type && key.number > node->key.number)) {

node->right = insert(node->right, key, lineNumber); // Вставляем в правое поддерево

}

else {

// Если элемент уже существует, добавляем номер строки в список

if (std::find(node->lineNumbers.begin(), node->lineNumbers.end(), lineNumber) == node->lineNumbers.end()) {

node->lineNumbers.push\_back(lineNumber);

}

return node; // Элемент уже существует

}

// Обновляем коэффициент балансировки

node->balanceFactor = balanceFactor(node);

// Балансировка дерева

if (node->balanceFactor > 1 && (key.type < node->left->key.type || (key.type == node->left->key.type && key.number < node->left->key.number))) {

return rotateRight(node); // LL

}

if (node->balanceFactor < -1 && (key.type > node->right->key.type || (key.type == node->right->key.type && key.number > node->right->key.number))) {

return rotateLeft(node); // RR

}

if (node->balanceFactor > 1 && (key.type > node->left->key.type || (key.type == node->left->key.type && key.number > node->left->key.number))) {

node->left = rotateLeft(node->left); // LR

return rotateRight(node);

}

if (node->balanceFactor < -1 && (key.type < node->right->key.type || (key.type == node->right->key.type && key.number < node->right->key.number))) {

node->right = rotateRight(node->right); // RL

return rotateLeft(node);

}

return node; // Возвращаем узел

}

// Метод для удаления узла из дерева

TreeNode\* remove(TreeNode\* root, GroupNumber key) {

if (!root) return root; // Если узел пуст, возвращаем его

// Сравниваем ключи и удаляем из соответствующего поддерева

if (key.type < root->key.type || (key.type == root->key.type && key.number < root->key.number)) {

root->left = remove(root->left, key); // Удаляем из левого поддерева

}

else if (key.type > root->key.type || (key.type == root->key.type && key.number > root->key.number)) {

root->right = remove(root->right, key); // Удаляем из правого поддерева

}

else {

// Узел с одним дочерним элементом или без них

if (!root->left || !root->right) {

TreeNode\* temp = root->left ? root->left : root->right; // Узел, который заменит удаляемый

if (!temp) { // Если нет дочерних узлов

temp = root; // Удаляемый узел

root = nullptr; // Узел удален

}

else { // Копируем содержимое

\*root = \*temp;

}

delete temp; // Освобождаем память

}

else {

// Узел с двумя дочерними элементами

TreeNode\* temp = getMinValueNode(root->right); // Находим минимальный узел в правом поддереве

root->key = temp->key; // Копируем значение

root->right = remove(root->right, temp->key); // Удаляем узел

}

}

// Если дерево стало пустым

if (!root) return root;

// Обновление коэффициента балансировки

root->balanceFactor = balanceFactor(root);

// Балансировка дерева

if (root->balanceFactor > 1 && balanceFactor(root->left) >= 0)

return rotateRight(root); // LL

if (root->balanceFactor > 1 && balanceFactor(root->left) < 0) {

root->left = rotateLeft(root->left); // LR

return rotateRight(root);

}

if (root->balanceFactor < -1 && balanceFactor(root->right) <= 0)

return rotateLeft(root); // RR

if (root->balanceFactor < -1 && balanceFactor(root->right) > 0) {

root->right = rotateRight(root->right); // RL

return rotateLeft(root);

}

return root; // Возвращаем узел

}

// Метод для получения узла с минимальным значением

TreeNode\* getMinValueNode(TreeNode\* node) {

TreeNode\* current = node; // Начинаем с данного узла

while (current && current->left) current = current->left; // Идем до самого левого узла

return current; // Возвращаем узел

}

// Метод для поиска узла в дереве

TreeNode\* search(TreeNode\* node, GroupNumber key) {

if (node == nullptr || (node->key.type == key.type && node->key.number == key.number))

return node; // Если узел пуст или ключ совпадает, возвращаем узел

// Сравниваем ключи и ищем в соответствующем поддереве

if (key.type < node->key.type || (key.type == node->key.type && key.number < node->key.number))

return search(node->left, key); // Поиск в левом поддереве

return search(node->right, key); // Поиск в правом поддереве

}

// Метод для получения высоты дерева

int getHeight(TreeNode\* node) const {

if (!node) return 0;

return std::max(getHeight(node->left), getHeight(node->right)) + 1;

}

// Метод для печати дерева

void print(TreeNode\* node, int indent, int height) const {

if (node) {

// Печатаем правое поддерево

if (node->right) {

print(node->right, indent + 4, height);

}

// Печатаем текущий узел

if (indent) {

std::cout << std::setw(indent) << ' '; // Отступ

}

std::cout << node->key.type << node->key.number; // Печатаем узел

// Печатаем номера строк

if (!node->lineNumbers.empty()) {

std::cout << " (строки: ";

for (size\_t i = 0; i < node->lineNumbers.size(); ++i) {

std::cout << node->lineNumbers[i];

if (i < node->lineNumbers.size() - 1) {

std::cout << ", ";

}

}

std::cout << ")";

}

std::cout << std::endl; // Переход на новую строку

// Печатаем левое поддерево

if (node->left) {

print(node->left, indent + 4, height);

}

}

}

// Метод для прямого обхода дерева

void preOrderTraversal(TreeNode\* node, std::ostream& out) {

if (node) {

out << node->key.type << node->key.number << " (строки: ";

for (const auto& line : node->lineNumbers) {

out << line << " ";

}

out << ")" << std::endl; // Запись узла

preOrderTraversal(node->left, out); // Обход левого поддерева

preOrderTraversal(node->right, out); // Обход правого поддерева

}

}

// Метод для очистки дерева

void clear(TreeNode\* node) {

if (node) {

clear(node->left); // Освобождаем левое поддерево

clear(node->right); // Освобождаем правое поддерево

delete node; // Освобождение памяти

}

}

};

void showMenu() {

std::cout << "Выберите действие:\n";

std::cout << "1. Инициализация (пустое дерево)\n";

std::cout << "2. Добавление нового элемента\n";

std::cout << "3. Удаление заданного элемента\n";

std::cout << "4. Поиск заданного элемента\n";

std::cout << "5. Печать дерева\n";

std::cout << "6. Прямой обход дерева\n";

std::cout << "7. Освобождение памяти (удаление всего дерева)\n";

std::cout << "0. Выход\n";

}

int main() {

setlocale(LC\_ALL, "Russian");

AVLTree tree; // Создание объекта дерева

tree.loadFromFile("input.txt"); // Автоматическая загрузка данных из файла при старте

int choice; // Переменная для выбора действия

do {

showMenu(); // Показать меню

std::cin >> choice; // Ввод пользователем выбора

switch (choice) { // В зависимости от выбора выполняем соответствующее действие

case 1:

tree.clear(); // Очистка дерева (инициализация пустого дерева)

std::cout << "Дерево инициализировано.\n"; // Сообщаем об инициализации

break;

case 2: { // Добавление нового элемента

GroupNumber group; // Создаем переменную для группы

std::cout << "Введите тип (B/M/S) и номер группы (4 цифры): "; // Запрашиваем ввод пользователя

std::cin >> group.type >> group.number; // Считываем тип и номер группы

// Проверка на корректность ввода

if (group.number < 1000 || group.number > 9999 || (group.type != 'B' && group.type != 'M' && group.type != 'S')) {

std::cout << "Некорректный ввод!" << std::endl; // Сообщаем об ошибке

}

else {

tree.insert(group, 0); // Добавить элемент

std::cout << "Элемент добавлен.\n";

}

break;

}

case 3: { // Удаление заданного элемента

GroupNumber group; // Создаем переменную для группы

std::cout << "Введите тип (B/M/S) и номер группы (4 цифры) для удаления: "; // Запрашиваем ввод

std::cin >> group.type >> group.number; // Считываем данные

// Проверка на корректность ввода

if (group.number < 1000 || group.number > 9999 || (group.type != 'B' && group.type != 'M' && group.type != 'S')) {

std::cout << "Некорректный ввод!" << std::endl; // Сообщаем об ошибке

}

else {

tree.remove(group); // Удалить элемент

std::cout << "Элемент удалён.\n";

}

break;

}

case 4: { // Поиск заданного элемента

GroupNumber group; // Создаем переменную для группы

std::cout << "Введите тип (B/M/S) и номер группы (4 цифры) для поиска: "; // Запрашиваем ввод

std::cin >> group.type >> group.number; // Считываем данные

// Проверка на корректность ввода

if (group.number < 1000 || group.number > 9999 || (group.type != 'B' && group.type != 'M' && group.type != 'S')) {

std::cout << "Некорректный ввод!" << std::endl; // Сообщаем об ошибке

}

else {

TreeNode\* result = tree.search(group); // Поиск элемента

if (result) {

std::cout << "Элемент найден: " << result->key.type << result->key.number << " (строки: ";

for (const auto& line : result->lineNumbers) {

std::cout << line << " ";

}

std::cout << ")" << std::endl; // Сообщаем о найденном элементе

}

else {

std::cout << "Элемент не найден.\n";

}

}

break;

}

case 5: // Печать дерева

std::cout << "Дерево:\n"; // Сообщаем, что будет печататься дерево

tree.print(); // Печатаем дерево

break;

case 6: // Прямой обход дерева

std::cout << "Прямой обход дерева:\n"; // Сообщаем о прямом обходе

tree.preOrderTraversal(std::cout); // Прямой обход дерева (посредством preOrderTraversal)

break;

case 7: // Освобождение памяти

tree.clear(); // Освобождение памяти

std::cout << "Дерево очищено.\n";

break;

case 0: // Выход из программы

tree.saveToFile("output.txt"); // Сохранение данных в файл перед выходом

std::cout << "Данные сохранены в файл output.txt.\n"; // Сообщаем о сохранении данных

std::cout << "Выход из программы.\n";

break;

default: // Неправильный ввод

std::cout << "Некорректный выбор, попробуйте снова.\n";

}

} while (choice != 0); // Продолжать до выбора выхода

tree.clear(); // Освобождение памяти перед выходом

return 0;

}