Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(ФГАОУ ВО ПНИПУ)

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ

ОТЧЁТ

О ЛАБОРАТОРНОЙ РАБОТЕ

На тему: ДЕРЕВЬЯ

Выполнил:

Студент группы РИС-23-3б

Асташин Д.А.

Проверил:

доц. кафедры ИТАС

Полякова О.А.

Пермь 2024

**Постановка задачи**

1. Самостоятельно придумать вид Дерева и

реализовать алгоритмы для этого собственного варианта бинарного дерева поиска, имеющего не менее трёх уровней.

2. Алгоритмы:

2.1. Необходимо реализовать функции для редактирования дерева:

- Вставка узла.

- Удаление узла.

- Поиск элемента по ключу.

2.2 Реализовать алгоритмы обхода дерева:

2.3 Прямой

2.4 Симметричный

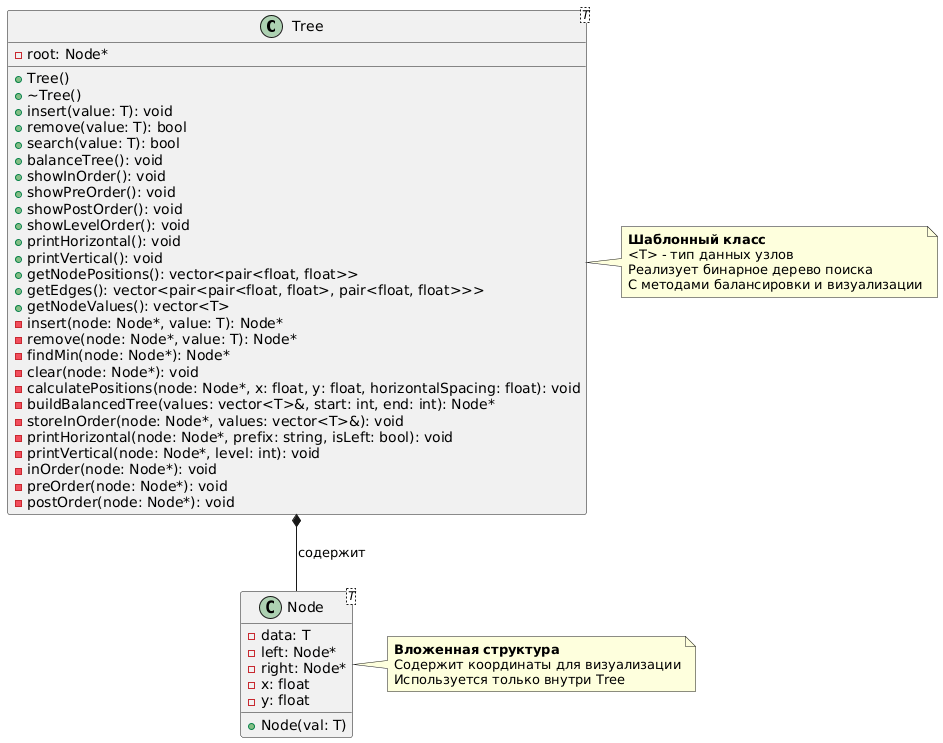
2.5 Обратный

3. Реализовать алгоритм балансировки дерева.

4. Реализовать вертикальную и горизонтальную печать.

5. Визуализацию дерева выполнить с использованием любой доступной графической библиотеки – SFML, SDL, OpenGL…

**UML Диаграмма**



**Код программы**

**Файл Tree.h**

#ifndef TREE\_H

#define TREE\_H

#include <iostream>

#include <queue>

#include <stack>

#include <vector>

#include <string>

#include <algorithm>

template<typename T>

class Tree {

private:

struct Node {

T data;

Node\* left;

Node\* right;

float x, y;

Node(T val) : data(val), left(nullptr), right(nullptr), x(0), y(0) {}

};

Node\* root;

// Вспомогательные методы

Node\* insert(Node\* node, T value);

Node\* remove(Node\* node, T value);

Node\* findMin(Node\* node);

void clear(Node\* node);

void calculatePositions(Node\* node, float x, float y, float horizontalSpacing);

// Методы для балансировки

Node\* buildBalancedTree(std::vector<T>& values, int start, int end);

void storeInOrder(Node\* node, std::vector<T>& values);

// Методы для печати

void printHorizontal(Node\* node, const std::string& prefix, bool isLeft);

void printVertical(Node\* node, int level = 0);

public:

Tree() : root(nullptr) {}

~Tree() { clear(root); }

// Основные операции

void insert(T value);

bool remove(T value);

bool search(T value);

// Балансировка

void balanceTree();

// Обходы

void showInOrder();

void showPreOrder();

void showPostOrder();

void showLevelOrder();

// Печать

void printHorizontal();

void printVertical();

// Для визуализации

std::vector<std::pair<float, float>> getNodePositions();

std::vector<std::pair<std::pair<float, float>, std::pair<float, float>>> getEdges();

std::vector<T> getNodeValues();

private:

void inOrder(Node\* node);

void preOrder(Node\* node);

void postOrder(Node\* node);

};

// Реализация основных методов

template<typename T>

typename Tree<T>::Node\* Tree<T>::insert(Node\* node, T value) {

if (node == nullptr) {

return new Node(value);

}

if (value < node->data) {

node->left = insert(node->left, value);

}

else if (value > node->data) {

node->right = insert(node->right, value);

}

return node;

}

template<typename T>

void Tree<T>::insert(T value) {

root = insert(root, value);

}

template<typename T>

typename Tree<T>::Node\* Tree<T>::remove(Node\* node, T value) {

if (node == nullptr) return nullptr;

if (value < node->data) {

node->left = remove(node->left, value);

}

else if (value > node->data) {

node->right = remove(node->right, value);

}

else {

if (node->left == nullptr) {

Node\* temp = node->right;

delete node;

return temp;

}

else if (node->right == nullptr) {

Node\* temp = node->left;

delete node;

return temp;

}

Node\* temp = findMin(node->right);

node->data = temp->data;

node->right = remove(node->right, temp->data);

}

return node;

}

template<typename T>

bool Tree<T>::remove(T value) {

if (!search(value)) return false;

root = remove(root, value);

return true;

}

template<typename T>

typename Tree<T>::Node\* Tree<T>::findMin(Node\* node) {

while (node && node->left != nullptr) {

node = node->left;

}

return node;

}

template<typename T>

bool Tree<T>::search(T value) {

Node\* current = root;

while (current != nullptr) {

if (value == current->data) return true;

else if (value < current->data) current = current->left;

else current = current->right;

}

return false;

}

template<typename T>

void Tree<T>::clear(Node\* node) {

if (node != nullptr) {

clear(node->left);

clear(node->right);

delete node;

}

}

// Методы балансировки

template<typename T>

void Tree<T>::storeInOrder(Node\* node, std::vector<T>& values) {

if (node == nullptr) return;

storeInOrder(node->left, values);

values.push\_back(node->data);

storeInOrder(node->right, values);

}

template<typename T>

typename Tree<T>::Node\* Tree<T>::buildBalancedTree(std::vector<T>& values, int start, int end) {

if (start > end) return nullptr;

int mid = (start + end) / 2;

Node\* node = new Node(values[mid]);

node->left = buildBalancedTree(values, start, mid - 1);

node->right = buildBalancedTree(values, mid + 1, end);

return node;

}

template<typename T>

void Tree<T>::balanceTree() {

std::vector<T> values;

storeInOrder(root, values); // Получаем отсортированные значения

// Очищаем старое дерево

clear(root);

// Строим сбалансированное дерево

root = buildBalancedTree(values, 0, values.size() - 1);

}

// Методы печати

template<typename T>

void Tree<T>::printHorizontal() {

std::cout << "\n=== ГОРИЗОНТАЛЬНАЯ ПЕЧАТЬ ===" << std::endl;

printHorizontal(root, "", true);

}

template<typename T>

void Tree<T>::printVertical() {

std::cout << "\n=== ВЕРТИКАЛЬНАЯ ПЕЧАТЬ ===" << std::endl;

printVertical(root);

std::cout << std::endl;

}

template<typename T>

void Tree<T>::printHorizontal(Node\* node, const std::string& prefix, bool isLeft) {

if (node != nullptr) {

std::cout << prefix;

std::cout << (isLeft ? "|--" : "'--");

std::cout << node->data << std::endl;

printHorizontal(node->left, prefix + (isLeft ? "| " : " "), true);

printHorizontal(node->right, prefix + (isLeft ? "| " : " "), false);

}

}

template<typename T>

void Tree<T>::printVertical(Node\* node, int level) {

if (node != nullptr) {

printVertical(node->right, level + 1);

for (int i = 0; i < level; i++) {

std::cout << " ";

}

std::cout << node->data << std::endl;

printVertical(node->left, level + 1);

}

}

// Методы обхода

template<typename T>

void Tree<T>::inOrder(Node\* node) {

if (node != nullptr) {

inOrder(node->left);

std::cout << node->data << " ";

inOrder(node->right);

}

}

template<typename T>

void Tree<T>::preOrder(Node\* node) {

if (node != nullptr) {

std::cout << node->data << " ";

preOrder(node->left);

preOrder(node->right);

}

}

template<typename T>

void Tree<T>::postOrder(Node\* node) {

if (node != nullptr) {

postOrder(node->left);

postOrder(node->right);

std::cout << node->data << " ";

}

}

template<typename T>

void Tree<T>::showInOrder() {

std::cout << "Симметричный обход: ";

inOrder(root);

std::cout << std::endl;

}

template<typename T>

void Tree<T>::showPreOrder() {

std::cout << "Прямой обход: ";

preOrder(root);

std::cout << std::endl;

}

template<typename T>

void Tree<T>::showPostOrder() {

std::cout << "Обратный обход: ";

postOrder(root);

std::cout << std::endl;

}

template<typename T>

void Tree<T>::showLevelOrder() {

std::cout << "Обход в ширину: ";

if (root == nullptr) {

std::cout << "Дерево пустое" << std::endl;

return;

}

std::queue<Node\*> q;

q.push(root);

while (!q.empty()) {

Node\* current = q.front();

q.pop();

std::cout << current->data << " ";

if (current->left) q.push(current->left);

if (current->right) q.push(current->right);

}

std::cout << std::endl;

}

// Методы для визуализации

template<typename T>

void Tree<T>::calculatePositions(Node\* node, float x, float y, float horizontalSpacing) {

if (node == nullptr) return;

node->x = x;

node->y = y;

calculatePositions(node->left, x - horizontalSpacing, y + 100, horizontalSpacing / 2);

calculatePositions(node->right, x + horizontalSpacing, y + 100, horizontalSpacing / 2);

}

template<typename T>

std::vector<std::pair<float, float>> Tree<T>::getNodePositions() {

std::vector<std::pair<float, float>> positions;

calculatePositions(root, 400, 100, 200);

std::queue<Node\*> q;

if (root) q.push(root);

while (!q.empty()) {

Node\* current = q.front();

q.pop();

positions.push\_back({ current->x, current->y });

if (current->left) q.push(current->left);

if (current->right) q.push(current->right);

}

return positions;

}

template<typename T>

std::vector<std::pair<std::pair<float, float>, std::pair<float, float>>> Tree<T>::getEdges() {

std::vector<std::pair<std::pair<float, float>, std::pair<float, float>>> edges;

calculatePositions(root, 400, 100, 200);

std::queue<Node\*> q;

if (root) q.push(root);

while (!q.empty()) {

Node\* current = q.front();

q.pop();

if (current->left) {

edges.push\_back({ {current->x, current->y}, {current->left->x, current->left->y} });

q.push(current->left);

}

if (current->right) {

edges.push\_back({ {current->x, current->y}, {current->right->x, current->right->y} });

q.push(current->right);

}

}

return edges;

}

template<typename T>

std::vector<T> Tree<T>::getNodeValues() {

std::vector<T> values;

calculatePositions(root, 400, 100, 200);

std::queue<Node\*> q;

if (root) q.push(root);

while (!q.empty()) {

Node\* current = q.front();

q.pop();

values.push\_back(current->data);

if (current->left) q.push(current->left);

if (current->right) q.push(current->right);

}

return values;

}

#endif

**Файл main.cpp**

#include <SFML/Graphics.hpp>

#include <iostream>

#include <thread>

#include <string>

#include <sstream>

#include "tree.h"

using namespace std;

Tree<int> tree;

bool treeUpdated = false;

// Переменные для камеры

sf::View camera;

float cameraX = 0.0f;

float cameraY = 0.0f;

float cameraZoom = 1.0f;

sf::RenderWindow window(sf::VideoMode(800, 600), "Binary Tree - Camera Control");

// Функция для отрисовки дерева

void drawTree() {

window.clear(sf::Color::White);

// Устанавливаем камеру

camera.setCenter(cameraX, cameraY);

camera.setSize(800 \* cameraZoom, 600 \* cameraZoom);

window.setView(camera);

sf::Font font;

if (!font.loadFromFile("arial.ttf")) {

// Если шрифт не загружен, продолжаем без текста

}

// Рисуем ребра

auto edges = tree.getEdges();

for (const auto& edge : edges) {

sf::Vertex line[] = {

sf::Vertex(sf::Vector2f(edge.first.first, edge.first.second), sf::Color::Black),

sf::Vertex(sf::Vector2f(edge.second.first, edge.second.second), sf::Color::Black)

};

window.draw(line, 2, sf::Lines);

}

// Рисуем узлы

auto nodePositions = tree.getNodePositions();

auto nodeValues = tree.getNodeValues();

for (size\_t i = 0; i < nodePositions.size(); i++) {

// Рисуем круг (узел)

sf::CircleShape nodeCircle(25);

nodeCircle.setFillColor(sf::Color(173, 216, 230));

nodeCircle.setOutlineThickness(2);

nodeCircle.setOutlineColor(sf::Color::Blue);

nodeCircle.setPosition(nodePositions[i].first - 25, nodePositions[i].second - 25);

window.draw(nodeCircle);

// Рисуем текст

if (font.getInfo().family != "") {

sf::Text text;

text.setFont(font);

text.setString(to\_string(nodeValues[i]));

text.setCharacterSize(20);

text.setFillColor(sf::Color::Black);

text.setStyle(sf::Text::Bold);

sf::FloatRect textBounds = text.getLocalBounds();

text.setOrigin(textBounds.width / 2, textBounds.height / 2);

text.setPosition(nodePositions[i].first, nodePositions[i].second - 5);

window.draw(text);

}

}

// Возвращаем обычный вид для интерфейса

window.setView(window.getDefaultView());

window.display();

}

// Функция для обработки ввода камеры

void handleCameraInput() {

float moveSpeed = 10.0f / cameraZoom; // Скорость движения зависит от зума

float zoomSpeed = 0.1f;

// Движение камеры

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {

cameraX -= moveSpeed;

treeUpdated = true;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {

cameraX += moveSpeed;

treeUpdated = true;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {

cameraY -= moveSpeed;

treeUpdated = true;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {

cameraY += moveSpeed;

treeUpdated = true;

}

// Зум

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Add) || sf::Keyboard::isKeyPressed(sf::Keyboard::Equal)) {

cameraZoom -= zoomSpeed;

if (cameraZoom < 0.1f) cameraZoom = 0.1f;

treeUpdated = true;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Subtract) || sf::Keyboard::isKeyPressed(sf::Keyboard::Dash)) {

cameraZoom += zoomSpeed;

if (cameraZoom > 3.0f) cameraZoom = 3.0f;

treeUpdated = true;

}

}

void showMenu() {

cout << "\n=== БИНАРНОЕ ДЕРЕВО ===" << endl;

cout << "1. Добавить узел" << endl;

cout << "2. Удалить узел" << endl;

cout << "3. Найти узел" << endl;

cout << "4. Показать все обходы" << endl;

cout << "5. Горизонтальная печать" << endl;

cout << "6. Вертикальная печать" << endl;

cout << "7. Сбалансировать дерево" << endl;

cout << "0. Выход" << endl;

cout << "\nВыберите действие: ";

}

void consoleMenu() {

int choice;

int value;

// Тестовые данные

tree.insert(50);

tree.insert(30);

tree.insert(70);

tree.insert(20);

tree.insert(40);

tree.insert(60);

tree.insert(80);

treeUpdated = true;

cout << "Добавлены тестовые узлы: 50, 30, 70, 20, 40, 60, 80" << endl;

cout << "Управление камерой в графическом окне:" << endl;

cout << "Стрелки - движение камеры" << endl;

cout << "+ - увеличение, - - уменьшение" << endl;

cout << "R - сброс камеры" << endl;

while (true) {

showMenu();

cin >> choice;

switch (choice) {

case 1:

cout << "Введите значение: ";

cin >> value;

tree.insert(value);

cout << "Узел " << value << " добавлен." << endl;

treeUpdated = true;

break;

case 2:

cout << "Введите значение: ";

cin >> value;

if (tree.remove(value)) {

cout << "Узел " << value << " удален." << endl;

}

else {

cout << "Узел " << value << " не найден." << endl;

}

treeUpdated = true;

break;

case 3:

cout << "Введите значение: ";

cin >> value;

if (tree.search(value)) {

cout << "Узел " << value << " найден." << endl;

}

else {

cout << "Узел " << value << " не найден." << endl;

}

break;

case 4:

cout << "\n--- Обходы дерева ---" << endl;

tree.showInOrder();

tree.showPreOrder();

tree.showPostOrder();

tree.showLevelOrder();

break;

case 5:

tree.printHorizontal();

break;

case 6:

tree.printVertical();

break;

case 7:

tree.balanceTree();

cout << "Дерево сбалансировано!" << endl;

treeUpdated = true;

break;

case 0:

cout << "Выход..." << endl;

window.close();

return;

default:

cout << "Неверный выбор!" << endl;

}

if (treeUpdated) {

treeUpdated = false;

}

this\_thread::sleep\_for(chrono::milliseconds(100));

}

}

int main() {

system("chcp 1251 > nul");

window.setFramerateLimit(60);

// Инициализация камеры

camera = window.getDefaultView();

// Запуск консольного меню в отдельном потоке

thread consoleThread(consoleMenu);

cout << "SFML окно создано. Используйте консоль для управления." << endl;

cout << "Управление камерой:" << endl;

cout << "Стрелки - движение" << endl;

cout << "+/- - зум" << endl;

cout << "R - сброс камеры" << endl;

// Главный цикл SFML

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

}

// Обработка клавиш для камеры

if (event.type == sf::Event::KeyPressed) {

if (event.key.code == sf::Keyboard::Escape) {

window.close();

}

// Сброс камеры

if (event.key.code == sf::Keyboard::R) {

cameraX = 0;

cameraY = 0;

cameraZoom = 1.0f;

treeUpdated = true;

}

}

// Зум колесиком мыши

if (event.type == sf::Event::MouseWheelScrolled) {

if (event.mouseWheelScroll.wheel == sf::Mouse::VerticalWheel) {

float zoomAmount = 0.1f;

if (event.mouseWheelScroll.delta > 0) {

cameraZoom -= zoomAmount;

}

else {

cameraZoom += zoomAmount;

}

// Ограничения зума

if (cameraZoom < 0.1f) cameraZoom = 0.1f;

if (cameraZoom > 3.0f) cameraZoom = 3.0f;

treeUpdated = true;

}

}

}

// Обработка непрерывного ввода для камеры

handleCameraInput();

// Перерисовываем если было изменение

if (treeUpdated) {

drawTree();

treeUpdated = false;

}

else {

// Или просто перерисовываем с текущим состоянием

drawTree();

}

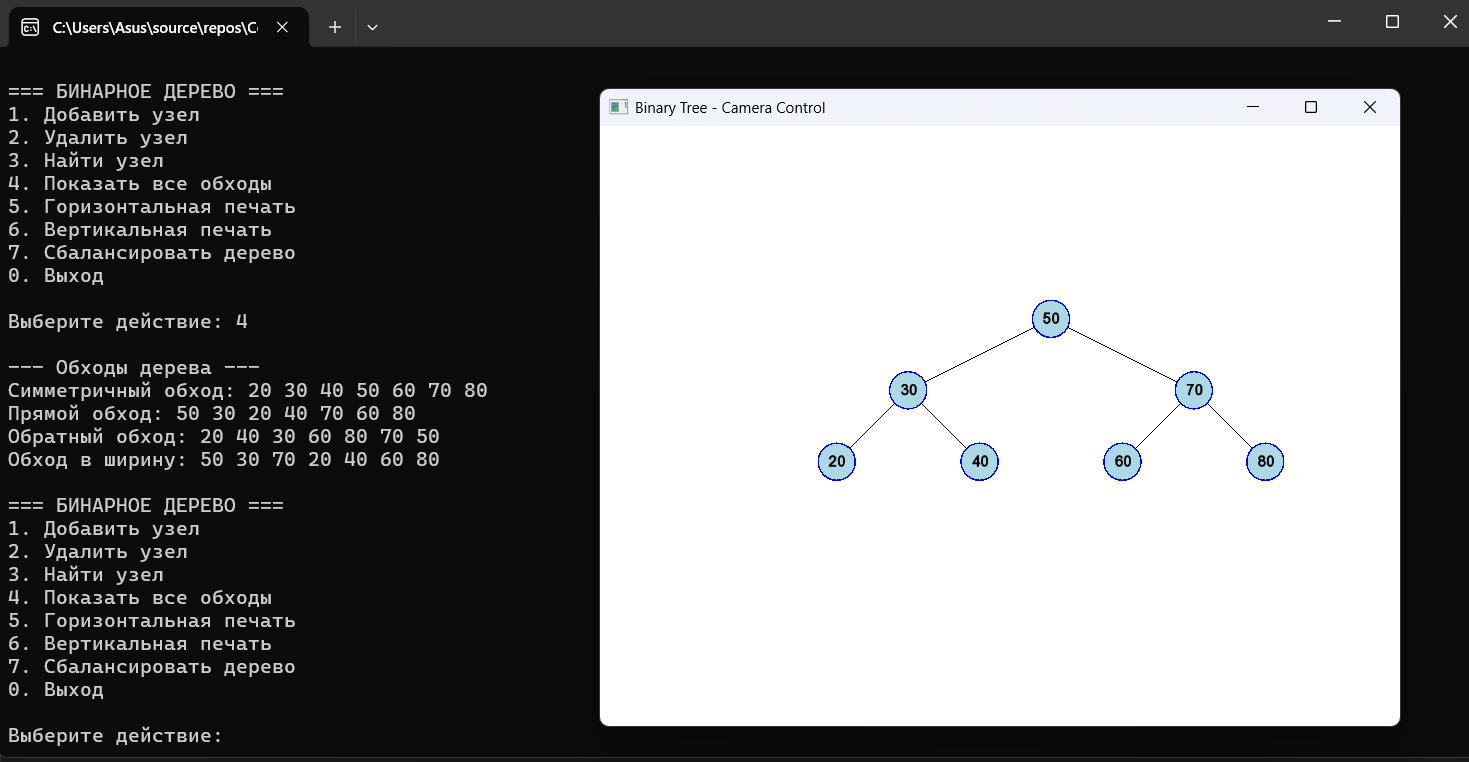
}

consoleThread.join();

return 0;

}

**Результаты работы программы**

****

**Ссылка GitHub**

https://github.com/astidii/PNRPU/tree/main/1%20semester/ConsoleApplication2