**Пермский Национальный Исследовательский Политехнический Университет**

Творческая работа:

«Бинарные деревья»

*Выполнил:*

*Коротаев Александр Дмитриевич РИС-23-3Б*

*Проверила:*

*доцент кафедры ИТАС О.А. Полякова*

2024

**Цель работы**

Получить практические навыки работы с бинарными деревьями.

**Постановка задачи**

1) Сформировать идеально сбалансированное бинарное дерево, тип

информационного поля указан в варианте.

2) Распечатать полученное дерево.

3) Выполнить обработку дерева в соответствии с заданием, вывести

полученный результат.

4) Преобразовать идеально сбалансированное дерево в дерево

поиска.

5) Распечатать полученное дерево

**Анализ задачи**

1) Бинарное дерево – это динамическая структура данных, состоящая из узлов, каждый из которых содержит, кроме данных, не более двух ссылок на различные бинарные деревья. Для описания дерева используется класс.

class Tree

{

private:

Tree\* left;//левая ветка

Tree\* right;//правая ветка

Tree\* parent;//родительская ветка

double data;//данные в узле

int Node\_radius;//радиус узла

};

2) Для визуализации дерева и интерфейса будет использована библиотека SFML.

#include <SFML/Graphics.hpp>

3) Для вставки элемента в дерево используется функция insert. В методе используется класс Tree, который содержит указатели на данные и ссылки на дочерние узлы (left и right). В методе insert выполняется поиск места для вставки нового элемента data. Если элемент уже присутствует в дереве, то он не будет добавлен повторно.

4) Для того, чтобы выполнить определенную операцию над всеми узлами дерева, все узлы надо обойти. Такая задача называется обходом дерева. При обходе узлы должны посещаться в определенном порядке.

5) Метод NLT класса Tree выполняет прямой обход дерева (то есть, посещение каждого узла ровно один раз), добавляя элементы данных в вектор vect. Обход начинается с текущего узла current и продолжается рекурсивно для левого и правого поддеревьев. Если текущий узел равен nullptr, то обход возвращается без изменений вектора.

6) Метод LNT класса Tree выполняет симметричный обход дерева, добавляя элементы данных в вектор vect. Обход начинается с левого поддерева и продолжается к правому поддереву, затем добавляются данные текущего узла.

7) Метод LRN класса Tree выполняет обратный обход дерева, добавляя элементы данных в вектор vect. Обход начинается с правого поддерева и продолжается к левому поддереву, затем добавляются данные текущего узла.

8) Функция get\_height возвращает целое число, которое является количеством уровней дерева.

9) Функция replace\_NULL\_for\_Empty() преобразую дерево из

неполного в полное, посредством добавления пустых узлов в копию текущего

дерева.

10) Метод erase класса Tree удаляет узел из двоичного дерева поиска,

основанного на данных. Сначала находится узел, который нужно удалить.

Затем, в зависимости от наличия дочерних узлов, либо просто удаляется узел,

если он листовой, либо удаляются все узлы, соединенные с выбранным.

11) Функция find находит узел дерева по ключу. Метод рекурсивно проверяет сначала левую ветвь дерева, затем правую. Если узел с заданным ключом найден, возвращается указатель на этот узел. В противном случае возвращается nullptr.

12) Функция draw выводит текущее бинарное дерево в окне интерфейса. Код выполняет следующие задачи:

1. Создание экземпляра класса RenderWindow с заданными размерами и заголовком.

2. Загрузка шрифта для кнопки закрытия окна.

3. Создание экземпляра класса RectButton, который представляет кнопку с возможностью обработки событий мыши.

4. Цикл обработки событий, включая обработку закрытия окна и нажатий кнопок мыши.

5. Очистка окна перед каждым новым кадром.

6. Построение и отрисовка дерева поиска с использованием рекурсии и логики, основанной на структуре данных levels.

7. Обновление и отображение содержимого окна. Функция draw\_node выводит узел, и связующую его ветвь.

13) Для получения значения узла, который надо добавить или удалить, необходимо текстовое поле, куда пользователь будет вводить данные. Класс TextBox в этом коде представляет собой текстовый редактор, который может быть использован в приложениях, созданных с помощью библиотеки SFML. Он предоставляет различные методы для настройки размера, положения, текста и других параметров. Также класс имеет внутренние структуры для рисования рамки и мигающего курсора. Методы класса включают:

- draw: для отрисовки текстового поля и его содержимого.

- handleEvent: для обработки событий ввода, таких как нажатия клавиш.

- getCurrentText: для получения текущего текста в текстовом поле.

Кроме того, класс TextBox содержит вложенный класс Text, который управляет отображением текста внутри текстового поля. Этот класс предоставляет методы для установки текста, его позиции и размера.

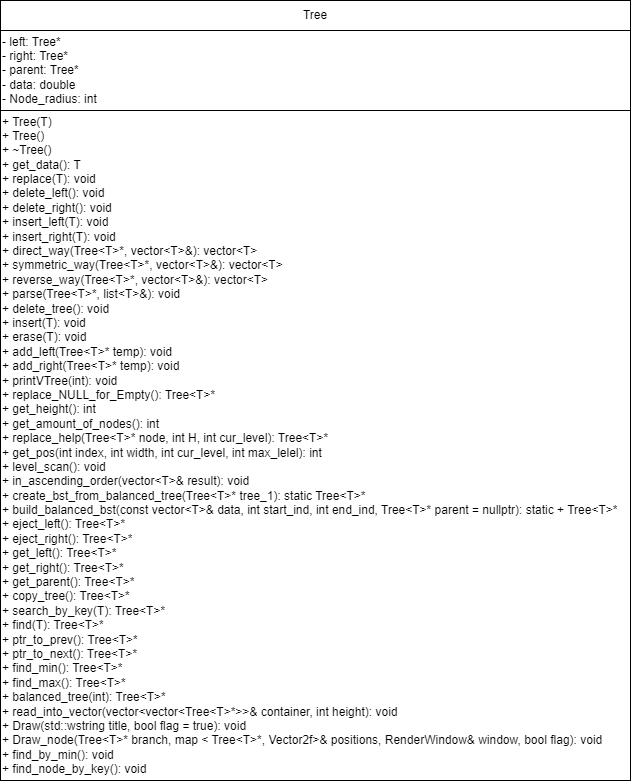
14) Класс Button представляет собой абстрактный класс, который определяет базовые методы для всех типов кнопок. Подклассы должны реализовать эти методы для конкретной реализации кнопки.

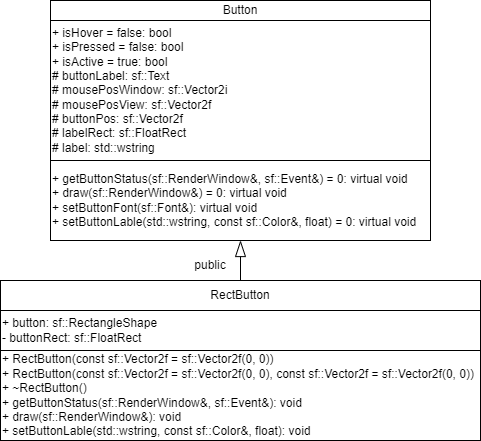
15) Класс RectButton наследуется от Button и реализует конкретные методы для прямоугольной кнопки. Он также содержит дополнительные члены данных, такие как sf::RectangleShape button, которые используются для рисования самой кнопки. Конструктор RectButton принимает параметры для определения размеров и положения кнопки. Деструктор не делает ничего особенного, так как он пустой. Метод getButtonStatus обрабатывает события мыши и обновляет состояние кнопки (isHover, isPressed). Метод draw отвечает за отрисовку кнопки на экране. Метод setButtonLable устанавливает надпись на кнопке.

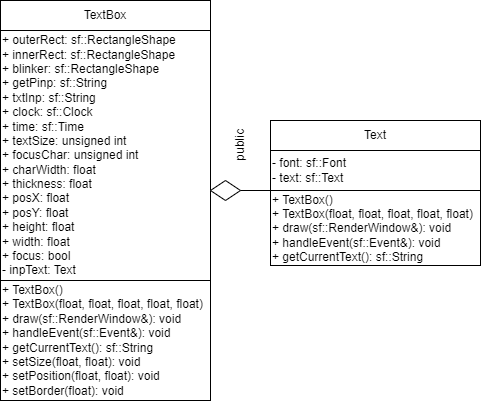
16) Функция MessageBox создает окно отображает сообщение message в центре окна. Она также добавляет кнопку "Ok" внизу окна. При нажатии на кнопку "Ok" окно закрывается.

17) Для использования какого-либо шрифта в интерфейсе необходимо загрузить в папку с проектом файл с расширением .ttf, в котором хранится необходимый шрифт.

**UML Диаграммы**







**Код**

Tree.h

#pragma once

#include <SFML/Graphics.hpp>

#include <vector>

#include <cmath>

#include <stdlib.h>

#include <algorithm>

#include <iomanip>

#include <sstream>

#include "textbox.h"

#include "button.h"

using namespace sf;

class Tree

{

private:

int font\_size = 20; // размер шрифта надписи значения узла

float shift\_x = 20; // смещение по x для следующего узла

float shift\_y = 60; // смещение по y для следующего узла

int node\_radius = 20; // размер окружности узла

Tree\* left;

Tree\* right;

Tree\* parent;

int height = 1;

double data;

Vector2f position;

public:

Tree();

Tree(double value);

~Tree();

// конструктор создания дерева из элементов вектора

Tree(const std::vector<double>& values);

// геттеры и сеттеры приватных полей

void setFontSize(int font\_size) { this->font\_size = font\_size; }

void setShiftX(float shift\_x) { this->shift\_x = shift\_x; }

void setShiftY(float shift\_y) { this->shift\_y = shift\_y; }

void setNodeRadius(int node\_radius) { this->node\_radius = node\_radius; }

int getFontSize() { return this->font\_size; }

float getShiftX() { return this->shift\_x; }

float getShiftY() { return this->shift\_y; }

int getNodeRadius() { return this->node\_radius; }

int getHeight()

{

if (this == nullptr) return 0;

return this->height;

}

void setLeft(Tree\* left) { this->left = left; }

void setRight(Tree\* right) { this->right = right; }

void setParent(Tree\* parent) { this->parent = parent; }

void setData(double data) { this->data = data; }

void setHeight(int height) { this->height = height; }

Tree\* getLeft() { return this->left; }

Tree\* getRight() { return this->right; }

Tree\* getParent() { return this->parent; }

double getData() { return this->data; }

Vector2f getPosition() { return this->position; }

void setPosition(Vector2f position) { this->position = position; }

// вычисление высоты дерева

int find\_height();

// вычисление максимальной длины вправо | влево от данного узла

int find\_left\_lenght(int level);

int find\_right\_lenght(int level);

// вставка значения в дерево

void insert(double value);

// окно с проходами по дереву

void showTraversals();

// Прямой обход (NLR)

void NLR(std::vector<double>\* vect);

// симметричный обход (LNR)

void LNR(std::vector<double>\* vect);

// обратный обход (LRN)

void LRN(std::vector<double>\* vect);

// удаление дерева

void delete\_tree();

// отрисовка дерева

void draw(const std::wstring title, bool isHorosontal = false);

int getBalance()

{

int leftHeight = (left ? left->height : 0);

int rightHeight = (right ? right->height : 0);

return leftHeight - rightHeight;

}

void updateHeight()

{

int leftHeight = (left ? left->height : 0);

int rightHeight = (right ? right->height : 0);

height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

}

Tree\* rotateLeft()

{

Tree\* newRoot = right;

right = newRoot->left;

newRoot->left = this;

parent = newRoot;

updateHeight();

newRoot->updateHeight();

return newRoot;

}

Tree\* rotateRight()

{

Tree\* newRoot = left;

left = newRoot->right;

newRoot->right = this;

parent = newRoot;

updateHeight();

newRoot->updateHeight();

return newRoot;

}

Tree\* balance()

{

int balanceFactor = getBalance();

if (balanceFactor > 1)

{

if (left && left->getBalance() < 0)

{

left = left->rotateLeft();

}

return rotateRight();

}

if (balanceFactor < -1)

{

if (right && right->getBalance() > 0)

{

right = right->rotateRight();

}

return rotateLeft();

}

return this;

}

void addNode();

void delNode();

// нахождение элемента по значению

Tree\* find(double key);

};

void MessageBox(std::wstring message);

// Отрисовка дерева

void drawTree(sf::RenderWindow& window, Tree\* node, float x, float y, int level, bool isHorisontal);

// сбалансированнное добавлние элемента

Tree\* insertNode(Tree\* root, double data);

// окно добавления

std::string getDataWindow(std::wstring mes);

Tree.cpp

#include "tree.h"

#include "button.h"

Tree::Tree() {}

Tree::Tree(double value)

{

left = nullptr;

right = nullptr;

parent = nullptr;

data = value;

}

Tree::~Tree() {}

Tree::Tree(const std::vector<double>& values)

{

if (values.empty()) { return; }

this->left = nullptr;

this->right = nullptr;

this->parent = nullptr;

this->data = values[0];

for (size\_t i = 1; i < values.size(); i++)

{

this->insert(values[i]);

}

}

void Tree::delete\_tree()

{

if (this == nullptr)

{

return;

}

// Рекурсивно удаляем левое поддерево

this->left->delete\_tree();

// Рекурсивно удаляем правое поддерево

this->right->delete\_tree();

// Удаляем текущий узел

delete this;

}

int Tree::find\_height()

{

int height\_left = 0, height\_right = 0;

if (this == nullptr)

return 0;

if (this->left != nullptr)

{

height\_left = this->left->find\_height();

}

if (this->right != nullptr)

{

height\_right = this->right->find\_height();

}

return std::max(height\_right, height\_left) + 1;

}

int Tree::find\_left\_lenght(int level)

{

int lenght\_left = 0, lenght\_right = lenght\_left;

if (this == nullptr)

return 0;

if (this->left != nullptr)

{

lenght\_left += shift\_x \* level \* 1.5;

lenght\_left += this->left->find\_left\_lenght(level - 1);

}

if (this->right != nullptr)

{

lenght\_right -= shift\_x \* level \* 1.5;

lenght\_right += this->right->find\_left\_lenght(level - 1);

}

return std::max(lenght\_left, lenght\_right);

}

int Tree::find\_right\_lenght(int level)

{

int lenght\_right = 0, lenght\_left = lenght\_right;

if (this == nullptr)

return 0;

if (this->left != nullptr)

{

lenght\_left -= shift\_x \* level \* 1.5;

lenght\_left += this->left->find\_right\_lenght(level - 1);

}

if (this->right != nullptr)

{

lenght\_right += shift\_x \* level \* 1.5;

lenght\_right += this->right->find\_right\_lenght(level - 1);

}

return std::max(lenght\_left, lenght\_right);

}

void Tree::insert(double value)

{

if (value < data)

{

if (left == nullptr)

{

left = new Tree(value);

left->parent = this;

}

else

{

left->insert(value);

}

}

else

{

if (right == nullptr)

{

right = new Tree(value);

right->parent = this;

}

else

{

right->insert(value);

}

}

}

void Tree::draw(const std::wstring title, bool isHorisontal)

{

int height\_tree = this->find\_height();

int lenght\_left = this->find\_left\_lenght(height\_tree);

int lenght\_right = this->find\_right\_lenght(height\_tree);

float window\_width, window\_height;

if (!isHorisontal)

{

window\_width = 100 + lenght\_left + lenght\_right;

window\_height = 150 + this->node\_radius \* 2 + shift\_y \* (height\_tree - 1);

}

else

{

window\_width = 100 + this->node\_radius \* 2 + shift\_y \* (height\_tree - 1);

window\_height = 150 + lenght\_left + lenght\_right;

}

// Создания окна

sf::RenderWindow window(sf::VideoMode(window\_width, window\_height), title);

window.setVerticalSyncEnabled(true);

// Установление иконки

sf::Image icon;

icon.loadFromFile("tree\_icon.png");

window.setIcon(512, 512, icon.getPixelsPtr());

// Установления шрифта

sf::Font font;

font.loadFromFile("gta.ttf");

RectButton button\_back({ 150, 40 }, { window\_width / 2 - 75, window\_height - 70 });

button\_back.setButtonFont(font);

button\_back.setButtonLable(L"Назад", sf::Color::White, 30);

while (window.isOpen())

{

sf::Event event;

button\_back.getButtonStatus(window, event);

while (window.pollEvent(event))

{

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

{

window.close();

}

if (event.type == sf::Event::MouseButtonPressed)

{

if (event.key.code == Mouse::Left)

{

if (button\_back.isPressed == true)

{

window.close();

}

}

}

}

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

button\_back.draw(window);

// отрисовка дерева

if (!isHorisontal)

drawTree(window, this, 50 + lenght\_left, 50, height\_tree, isHorisontal);

else

drawTree(window, this, 50, 50 + lenght\_right, height\_tree, isHorisontal);

window.display();

}

}

void drawLine(sf::RenderWindow& window, int radius, Vector2f pos\_start, Vector2f pos\_finish)

{

float x1 = pos\_start.x, y1 = pos\_start.y, x2 = pos\_finish.x, y2 = pos\_finish.y;

float delta\_x = fabs(x1 - x2);

float delta\_y = fabs(y1 - y2);

float lenght = sqrt(pow(x1 - x2, 2) + pow(y2 - y1, 2));

RectangleShape rect({ 1, lenght });

rect.setFillColor(Color(0, 0, 0));

rect.setPosition({ x1, y1 });

float angle;

if (x2 < x1 && y2 > y1)

{

angle = 57.2958 \* asin(delta\_x / lenght);

}

if (x2 > x1 && y2 > y1)

{

angle = 270 + 57.2958 \* asin(delta\_y / lenght);

}

if (x2 > x1 && y2 < y1)

{

angle = 180 + 57.2958 \* asin(delta\_x / lenght);

}

rect.setRotation(angle);

window.draw(rect);

}

void drawTree(sf::RenderWindow& window, Tree\* node, float x, float y, int level, bool isHorisontal)

{

if (node == nullptr) return;

int radius = node->getNodeRadius();

int shift\_x = node->getShiftX();

int shift\_y = node->getShiftY();

int font\_size = node->getFontSize();

Vector2f position(x - radius, y - radius);

node->setPosition({ x, y });

// Отображение узла как окружности

sf::CircleShape circle(radius, 30);

circle.setPosition(position);

circle.setFillColor(sf::Color::Yellow);

circle.setOutlineThickness(2);

circle.setOutlineColor(sf::Color::Black);

// Отображение значения узла внутри окружности

sf::Font font;

font.loadFromFile("gta.ttf"); // шрифт

// Форматирование значения узла

std::ostringstream buf;

buf << std::fixed << std::setprecision(1) << node->getData();

sf::Text text(buf.str(), font);

text.setCharacterSize(font\_size);

text.setOutlineThickness(1);

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

text.setPosition(x - (font\_size / 2.5 \* float(buf.str().size()) / 2) + 1, y - (font\_size \* 0.75) + 1);

if (node->getParent() != nullptr)

{

Vector2f pos\_parent, pos\_node;

pos\_parent = node->getParent()->getPosition();

pos\_node = node->getPosition();

if (!isHorisontal)

{

pos\_parent.y += radius;

pos\_node.y -= radius;

}

else

{

pos\_parent.x += radius;

pos\_node.x -= radius;

}

drawLine(window, node->getNodeRadius(), pos\_parent, pos\_node);

}

window.draw(circle);

window.draw(text);

// Рекурсивные вызовы для левого и правого потомка узла

if (!isHorisontal)

{

if (node->getLeft() != nullptr)

{

drawTree(window, node->getLeft(), x - shift\_x \* level \* 1.5, y + shift\_y, level - 1, isHorisontal);

}

if (node->getRight() != nullptr)

{

drawTree(window, node->getRight(), x + shift\_x \* level \* 1.5, y + shift\_y, level - 1, isHorisontal);

}

}

else

{

if (node->getLeft() != nullptr)

{

drawTree(window, node->getLeft(), x + shift\_y, y + shift\_x \* level \* 1.5, level - 1, isHorisontal);

}

if (node->getRight() != nullptr)

{

drawTree(window, node->getRight(), x + shift\_y, y - shift\_x \* level \* 1.5, level - 1, isHorisontal);

}

}

}

void Tree::showTraversals()

{

// Установления шрифта

sf::Font font;

font.loadFromFile("gta.ttf");

int font\_size = 30;

float window\_width, window\_height = 330;

int count\_symb = 0;

std::vector<double> values;

this->NLR(&values);

for (int i = 0; i < values.size(); i++)

{

std::ostringstream str;

str << std::fixed << std::setprecision(1) << values[i];

count\_symb += str.str().size();

}

if (count\_symb < 25)

window\_width = font\_size / 2.5 \* 25;

else

window\_width = font\_size / 2.5 \* count\_symb;

// Создание окна

sf::RenderWindow window(sf::VideoMode(window\_width + 100, window\_height), L"Обходы дерева");

window.setVerticalSyncEnabled(true);

// Установление иконки

sf::Image icon;

icon.loadFromFile("tree\_icon.png");

window.setIcon(512, 512, icon.getPixelsPtr());

RectButton button\_back({ 150, 40 }, { window\_width / 2 - 30, window\_height - 70 });

button\_back.setButtonFont(font);

button\_back.setButtonLable(L"Назад", sf::Color::White, 30);

while (window.isOpen())

{

sf::Event event;

button\_back.getButtonStatus(window, event);

while (window.pollEvent(event))

{

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

{

window.close();

}

if (event.type == sf::Event::MouseButtonPressed)

{

if (event.key.code == Mouse::Left)

{

if (button\_back.isPressed == true)

{

window.close();

}

}

}

}

std::vector <double> vect;

std::wstring text\_to\_show = L"";

// прямой обход (NLR)

text\_to\_show = L"";

this->NLR(&vect);

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

{

std::ostringstream buf;

buf << std::fixed << std::setprecision(1) << vect[i];

text\_to\_show = text\_to\_show + " " + buf.str();

}

sf::Text text\_NLR(L"прямой обход (nlr):\n" + text\_to\_show, font);

text\_NLR.setCharacterSize(font\_size);

text\_NLR.setPosition({50, 50});

text\_NLR.setOutlineThickness(1);

text\_NLR.setOutlineColor(Color::Black);

vect.clear();

// симметричный обход (LNR)

text\_to\_show = L"";

this->LNR(&vect);

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

{

std::ostringstream buf;

buf << std::fixed << std::setprecision(1) << vect[i];

text\_to\_show = text\_to\_show + " " + buf.str();

}

sf::Text text\_LNR(L"Симметричный обход (lnr):\n" + text\_to\_show, font);

text\_LNR.setCharacterSize(font\_size);

text\_LNR.setPosition({ 50, 110});

text\_LNR.setOutlineThickness(1);

text\_LNR.setOutlineColor(Color::Black);

vect.clear();

// обратный проход (LRN)

text\_to\_show = L"";

this->LRN(&vect);

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

{

std::ostringstream buf;

buf << std::fixed << std::setprecision(1) << vect[i];

text\_to\_show = text\_to\_show + " " + buf.str();

}

sf::Text text\_LRN(L"Обратный проход (lrn):\n" + text\_to\_show, font);

text\_LRN.setCharacterSize(font\_size);

text\_LRN.setPosition({ 50, 170 });

text\_LRN.setOutlineThickness(1);

text\_LRN.setOutlineColor(Color::Black);

vect.clear();

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

// отрисовка виджетов

button\_back.draw(window);

window.draw(text\_NLR);

window.draw(text\_LNR);

window.draw(text\_LRN);

window.display();

}

}

// прямой проход (NLR)

void Tree::NLR(std::vector<double>\* vect)

{

if (this == nullptr)

{

return;

}

vect->push\_back(this->data);

if (this->left != nullptr)

{

this->left->NLR(vect);

}

if (this->right != nullptr)

{

this->right->NLR(vect);

}

}

// симметричный проход (LNR)

void Tree::LNR(std::vector<double>\* vect)

{

if (this == nullptr)

{

return;

}

if (this->left != nullptr)

{

this->left->LNR(vect);

}

vect->push\_back(this->data);

if (this->right != nullptr)

{

this->right->LNR(vect);

}

}

// обратный проход (LRN)

void Tree::LRN(std::vector<double>\* vect)

{

if (this == nullptr)

{

return;

}

if (this->left != nullptr)

{

this->left->LRN(vect);

}

if (this->right != nullptr)

{

this->right->LRN(vect);

}

vect->push\_back(this->data);

}

int getBalanceFactor(Tree\* node)

{

if (node == nullptr) return 0;

return node->getLeft()->getHeight() - node->getRight()->getHeight();

}

void updateHeight(Tree\* node)

{

if (node != nullptr)

{

node->setHeight(1 + std::max(node->getLeft()->find\_height(), node->getRight()->find\_height()));

}

}

Tree\* rotateRight(Tree\* node)

{

Tree\* newRoot = node->getLeft();

node->setLeft(newRoot->getRight());

newRoot->setRight(node);

newRoot->setParent(node->getParent());

node->setParent(newRoot);

if (node->getLeft() != nullptr)

node->getLeft()->setParent(node);

updateHeight(node);

updateHeight(newRoot);

return newRoot;

}

Tree\* rotateLeft(Tree\* node)

{

Tree\* newRoot = node->getRight();

node->setRight(newRoot->getLeft());

newRoot->setLeft(node);

newRoot->setParent(node->getParent());

node->setParent(newRoot);

if (node->getRight() != nullptr)

node->getRight()->setParent(node);

updateHeight(node);

updateHeight(newRoot);

return newRoot;

}

Tree\* balanceTree(Tree\* node)

{

if (node == nullptr) return nullptr;

updateHeight(node);

int balanceFactor = getBalanceFactor(node);

if (balanceFactor > 1)

{

if (getBalanceFactor(node->getLeft()) < 0)

{

node->setLeft(rotateLeft(node->getLeft()));

}

return rotateRight(node);

}

else if (balanceFactor < -1)

{

if (getBalanceFactor(node->getRight()) > 0)

{

node->setRight(rotateRight(node->getRight()));

}

return rotateLeft(node);

}

return node;

}

Tree\* insertNode(Tree\* root, double data)

{

if (root == nullptr) return new Tree(data);

if (data < root->getData())

{

root->setLeft(insertNode(root->getLeft(), data));

root->getLeft()->setParent(root);

}

else if (data >= root->getData())

{

root->setRight(insertNode(root->getRight(), data));

root->getRight()->setParent(root);

}

return balanceTree(root);

}

std::string getDataWindow(std::wstring text\_message)

{

int window\_width = 540, window\_height = 300;

RenderWindow window(VideoMode(window\_width, window\_height), L"Ввод данных");

// Установление иконки

sf::Image icon;

icon.loadFromFile("tree\_icon.png");

window.setIcon(512, 512, icon.getPixelsPtr());

Font font;

font.loadFromFile("gta.ttf");

RectButton button\_accept(sf::Vector2f(195, 40), sf::Vector2f(window\_width-195-50, window\_height-40-50));

button\_accept.setButtonFont(font);

button\_accept.setButtonLable(L"Продолжить", Color::White, 30);

RectButton button\_exit(sf::Vector2f(195, 40), sf::Vector2f(50, window\_height - 40 - 50));

button\_exit.setButtonFont(font);

button\_exit.setButtonLable(L"Назад", Color::White, 30);

Text text\_msg;

text\_msg.setFont(font);

text\_msg.setString(text\_message);

text\_msg.setFillColor(Color::White);

text\_msg.setOutlineColor(Color::Black);

text\_msg.setOutlineThickness(1);

text\_msg.setCharacterSize(30);

text\_msg.setPosition(180, 50);

sdx::TextBox textBox(440, 32, 50, 100, 2);

//textBox.setPosition(0, 100);

std::string text\_to\_return = "";

while (window.isOpen())

{

Vector2i mousePoz = Mouse::getPosition(window); // позиция мыши в окне

sf::Event event;

button\_accept.getButtonStatus(window, event);

button\_exit.getButtonStatus(window, event);

while (window.pollEvent(event))

{

textBox.handleEvent(event);

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

{

window.close();

}

if (event.type == Event::MouseButtonPressed)

{

if (event.key.code == Mouse::Left)

{

if (button\_exit.isPressed)

{

window.close();

}

if (button\_accept.isPressed)

{

text\_to\_return = textBox.getCurrentText(); // получение текста из TextBox

window.close();

}

}

}

}

window.clear(Color::White);

button\_exit.draw(window);

button\_accept.draw(window);

textBox.draw(window);

window.draw(text\_msg);

window.display();

}

return text\_to\_return;

}

void Tree::addNode()

{

std::string user\_data = getDataWindow(L"Добавление узла");

if (user\_data.size() != 0)

{

double newData = std::stod(user\_data);

this->insert(newData);

}

}

void Tree::delNode()

{

std::string user\_data = getDataWindow(L"Удаление узла");

if (user\_data.size() != 0)

{

double delData = std::stod(user\_data);

// нахождение указателя на элемент

Tree\* deleted\_node = this->find(delData);

if (deleted\_node != nullptr)

{

Tree\* temp = deleted\_node->getParent();

if (temp != nullptr)

{

if (temp->left == deleted\_node)

{

temp->left = nullptr;

}

else

{

temp->right = nullptr;

}

}

deleted\_node->delete\_tree();

}

else

{

MessageBox(L"Элемент не найден");

}

}

}

void MessageBox(std::wstring message)

{

Font font;

font.loadFromFile("gta.ttf");

Text text\_msg(message, font, 30);

text\_msg.setFillColor(Color::White);

text\_msg.setOutlineColor(Color::Black);

text\_msg.setOutlineThickness(1);

float text\_size = text\_msg.getCharacterSize() / 2.5 \* message.size();

float window\_width = 100 + text\_size;

float window\_height = 100 + 40 + 30 + 30; // Отступы (сверху и снизу) + высота кнопки + высота текста + отступ между текстом и кнопкой

RenderWindow window(VideoMode(window\_width, window\_height), L"Уведомление");

text\_msg.setPosition(window\_width/2 - text\_size/2, 50);

// Установление иконки

sf::Image icon;

icon.loadFromFile("tree\_icon.png");

window.setIcon(512, 512, icon.getPixelsPtr());

RectButton button\_OK({150, 40}, {window\_width/2 - 75, window\_height - 40 - 50});

button\_OK.setButtonFont(font);

button\_OK.setButtonLable(L"OK", sf::Color::White, 30);

while (window.isOpen())

{

sf::Event event;

button\_OK.getButtonStatus(window, event);

while (window.pollEvent(event))

{

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

{

window.close();

}

if (event.type == sf::Event::MouseButtonPressed)

{

if (event.key.code == Mouse::Left)

{

if (button\_OK.isPressed)

{

window.close();

}

}

}

}

window.clear(Color::White);

button\_OK.draw(window);

window.draw(text\_msg);

window.display();

}

}

// нахождение элемента по значению key в поле data

Tree\* Tree::find(double key)

{

if (data == key)

{

return this;

}

else if (key < data && left != nullptr)

{

return left->find(key);

}

else if (key > data && right != nullptr)

{

return right->find(key);

}

else

{

return nullptr;

}

}

Button.h

#pragma once

#ifndef BUTTON\_HPP\_INCLUDED

#define BUTTON\_HPP\_INCLUDED

#include <SFML/Graphics.hpp>

#include <iostream>

#include <string>

const sf::Color defaultHovered = sf::Color(145, 145, 145); // цвет кнопки по умолчанию

const sf::Color defaultPressed = sf::Color(100, 100, 100); // цвет кнопки при нажатии

class Button

{

public:

virtual void getButtonStatus(sf::RenderWindow&, sf::Event&) = 0; //статус кнопки

virtual void draw(sf::RenderWindow&) = 0; //отображение кнопки

virtual void setButtonFont(sf::Font&); //шрифт текста на кнопке

virtual void setButtonLable(std::wstring, const sf::Color&, float) = 0; //установка надписи на кнопке

bool isHover = false; // наведения курсора

bool isPressed = false; // нажатия кнопки

bool isActive = true; // доступность для нажатия

protected:

sf::Text buttonLabel; // надпись на кнопке

sf::Vector2i mousePosWindow; // позиция мыши

sf::Vector2f buttonPos; // позиция кнопки

std::wstring label;//надпись

sf::Vector2f mousePosView; // ------------------------------------------------------------------------------------------------------------------

sf::FloatRect labelRect; // --------------------------------------------------------------------------------------------------------------------

};

class RectButton : public Button // подкласс прямоугольных кнопок

{

public:

RectButton(const sf::Vector2f = sf::Vector2f(0, 0));

RectButton(const sf::Vector2f = sf::Vector2f(0, 0), const sf::Vector2f = sf::Vector2f(0, 0));

~RectButton();

void getButtonStatus(sf::RenderWindow&, sf::Event&); // получение статуса нажатия кнопки

void draw(sf::RenderWindow&); // отрисовка кнопки

void setButtonLable(std::wstring, const sf::Color&, float); // отрисовка надписи

sf::RectangleShape button;

private:

sf::FloatRect buttonRect;

};

#endif

Button.cpp

#include "button.h"

RectButton::RectButton(const sf::Vector2f size)

{

this->button.setSize(size);

this->buttonRect = this->button.getLocalBounds();

}

RectButton::RectButton(const sf::Vector2f size, const sf::Vector2f position)

{

this->button.setSize(size);

this->button.setOutlineThickness(1);

this->button.setOutlineColor(sf::Color::Black);

this->button.setPosition(position);

this->buttonPos = position;

this->buttonRect = this->button.getLocalBounds();

}

RectButton::~RectButton() {}

//статус кнопки

void RectButton::getButtonStatus(sf::RenderWindow& window, sf::Event& event)

{

this->mousePosWindow = sf::Mouse::getPosition(window);//позиция мыши в окне

this->mousePosView = window.mapPixelToCoords(this->mousePosWindow);

this->isHover = false; // наведение курсора

this->isPressed = false; // нажатие кнопки

// если кнопка активна

if (isActive)

{

if (button.getGlobalBounds().contains(this->mousePosView)) //курсор наведен

{

this->isHover = true;

}

if (button.getGlobalBounds().contains(this->mousePosView)) // после 1 нажатия кнопки

{

this->isPressed = true; // изменение состояния нажатия кнопки

}

if (isHover) // при наведении курсора

{

button.setFillColor(defaultPressed);

}

else button.setFillColor(defaultHovered);

if (isPressed) //кнопка нажата

{

button.setFillColor(defaultPressed);

}

}

else //обычный цвет

{

button.setFillColor(defaultPressed);

}

}

void RectButton::draw(sf::RenderWindow& window) { //отображаю кнопку в окне

window.draw(button);//кнопка

window.draw(buttonLabel);//надпись

}

void RectButton::setButtonLable(std::wstring label, const sf::Color& color, float charSize) // изменение надписи

{

this->buttonLabel.setString(label); // надпись

this->buttonLabel.setCharacterSize(charSize);//размер символов

this->buttonLabel.setFillColor(color);//устанавливаю цвет

this->buttonLabel.setOutlineThickness(1);

this->label = label;//присваиваю надпись

this->labelRect = this->buttonLabel.getLocalBounds();

this->buttonLabel.setOrigin(this->labelRect.width / 2.0f, this->labelRect.height / 1.2f);//ставлю координаты

this->buttonLabel.setPosition(this->buttonPos.x + (this->buttonRect.width / 2.0f),//ставлю координаты

this->buttonPos.y + (this->buttonRect.height / 4.0f) + 7);

}

void Button::setButtonFont(sf::Font& font)

{

buttonLabel.setFont(font);

}

Main.cpp

#include "tree.h"

#include "button.h"

#include <iostream>

#define TREE\_SIZE 10

std::vector<double> tree\_rand(int size);

int main()

{

srand(time(0));

sf::Font font;

font.loadFromFile("gta.ttf");

// Анти-алиасинг

ContextSettings settings;

settings.antialiasingLevel = 8;

// Создание и инициализация дерева

std::vector<double> values = tree\_rand(TREE\_SIZE);

Tree\* tree = new Tree(values);

// Создание окна SFML

sf::RenderWindow window(sf::VideoMode(540, 700), L"Меню");

// Установление иконки

sf::Image icon;

icon.loadFromFile("tree\_icon.png");

window.setIcon(512, 512, icon.getPixelsPtr());

//window.setVerticalSyncEnabled(true);

window.setFramerateLimit(60);

CircleShape circle(40);

circle.setPosition({230, 480});

circle.setFillColor(sf::Color::Yellow);

circle.setOutlineThickness(2);

circle.setOutlineColor(sf::Color::Black);

Text text\_eyes(L"O.O", font, 60);

text\_eyes.setFillColor(sf::Color::White);

text\_eyes.setOutlineColor(Color::Black);

text\_eyes.setOutlineThickness(1);

text\_eyes.setPosition({ circle.getPosition().x - 10, circle.getPosition().y - 10 });

sf::Clock clock;

float move\_interval = 0.1f;

RectButton button\_vertical\_print({ 400, 40 }, { 70, 50 });

button\_vertical\_print.setButtonFont(font);

button\_vertical\_print.setButtonLable(L"Вертикальная печать", sf::Color::White, 30);

RectButton button\_horizontal\_print({ 400, 40 }, { 70, 100 });

button\_horizontal\_print.setButtonFont(font);

button\_horizontal\_print.setButtonLable(L"Горизонтальная печать", sf::Color::White, 30);

RectButton button\_tree\_traversal({ 400, 40 }, { 70, 150 });

button\_tree\_traversal.setButtonFont(font);

button\_tree\_traversal.setButtonLable(L"Обходы дерева", sf::Color::White, 30);

RectButton button\_add\_node({ 400, 40 }, { 70, 200 });

button\_add\_node.setButtonFont(font);

button\_add\_node.setButtonLable(L"Добавить узел", sf::Color::White, 30);

RectButton button\_delete\_node({ 400, 40 }, { 70, 250 });

button\_delete\_node.setButtonFont(font);

button\_delete\_node.setButtonLable(L"Удалить узел", sf::Color::White, 30);

RectButton button\_balanced\_tree({ 400, 40 }, { 70, 300 });

button\_balanced\_tree.setButtonFont(font);

button\_balanced\_tree.setButtonLable(L"Сбалансированное дерево", sf::Color::White, 30);

RectButton button\_find\_element({ 400, 40 }, { 70, 350 });

button\_find\_element.setButtonFont(font);

button\_find\_element.setButtonLable(L"Найти элемент", sf::Color::White, 30);

RectButton button\_find\_min({ 400, 40 }, { 70, 400 });

button\_find\_min.setButtonFont(font);

button\_find\_min.setButtonLable(L"Найти минимальный элемент", sf::Color::White, 30);

RectButton button\_exit({ 400, 40 }, { 70, 600 });

button\_exit.setButtonFont(font);

button\_exit.setButtonLable(L"Выход", sf::Color::White, 30);

bool is\_came = false;

int point\_start = text\_eyes.getPosition().x;

int point\_finish = text\_eyes.getPosition().x + 40;

while (window.isOpen())

{

sf::Event event;

button\_vertical\_print.getButtonStatus(window, event);

button\_horizontal\_print.getButtonStatus(window, event);

button\_tree\_traversal.getButtonStatus(window, event);

button\_add\_node.getButtonStatus(window, event);

button\_delete\_node.getButtonStatus(window, event);

button\_balanced\_tree.getButtonStatus(window, event);

button\_find\_element.getButtonStatus(window, event);

button\_find\_min.getButtonStatus(window, event);

button\_exit.getButtonStatus(window, event);

while (window.pollEvent(event))

{

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

window.close();

if (event.type == sf::Event::MouseButtonPressed)

{

if (event.key.code == Mouse::Left)

{

if (button\_vertical\_print.isPressed)

{

tree->draw(L"Вертикальная печать");

}

if (button\_horizontal\_print.isPressed)

{

tree->draw(L"Горизонтальная печать", true);

}

if (button\_tree\_traversal.isPressed)

{

tree->showTraversals();

}

if (button\_balanced\_tree.isPressed)

{

values.clear();

tree->LNR(&values);

Tree\* new\_tree = new Tree(values[0]);

for (size\_t i = 1; i < values.size(); i++)

{

new\_tree = insertNode(new\_tree, values[i]);

}

new\_tree->draw(L"Сбалансированное дерево");

}

if (button\_add\_node.isPressed)

{

tree->addNode();

}

if (button\_delete\_node.isPressed)

{

tree->delNode();

}

if (button\_find\_element.isPressed)

{

std::string key = getDataWindow(L"Найти элемент");

if (key != "")

{

Tree\* founded\_element = tree->find(std::stod(key));

if (founded\_element == nullptr)

{

MessageBox(L"Элемент не найден");

}

else

{

std::wstring text = L"Элемент найден: " + std::to\_wstring(founded\_element->getData());

MessageBox(text);

}

}

}

if (button\_find\_min.isPressed)

{

std::vector<double> vect;

tree->LNR(&vect);

Tree\* min\_element = tree->find(vect[0]);

std::wstring text = L"Элемент найден: " + std::to\_wstring(min\_element->getData());

MessageBox(text);

}

if (button\_exit.isPressed)

{

window.close();

}

}

}

}

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

button\_vertical\_print.draw(window);

button\_horizontal\_print.draw(window);

button\_tree\_traversal.draw(window);

button\_add\_node.draw(window);

button\_delete\_node.draw(window);

button\_balanced\_tree.draw(window);

button\_find\_element.draw(window);

button\_find\_min.draw(window);

button\_exit.draw(window);

window.draw(circle);

if (clock.getElapsedTime().asSeconds() >= move\_interval)

{

if (text\_eyes.getPosition().x > point\_start && is\_came)

{

text\_eyes.move(-1, 0);

if (text\_eyes.getPosition().x == point\_start)

{

is\_came = false;

}

}

if (text\_eyes.getPosition().x < point\_finish && !is\_came)

{

text\_eyes.move(1, 0);

if (text\_eyes.getPosition().x == point\_finish)

{

is\_came = true;

}

}

clock.restart();

}

window.draw(text\_eyes);

window.display();

}

return 0;

}

std::vector<double> tree\_rand(int size)

{

std::vector<double> vect;

for (int i = 0; i < size; i++)

{

double element = rand() % 1000;

vect.push\_back(element / 10.0);

}

return vect;

}

**Выводы**

В ходе работы я применил знания о работе с классами, и интерфейсами.

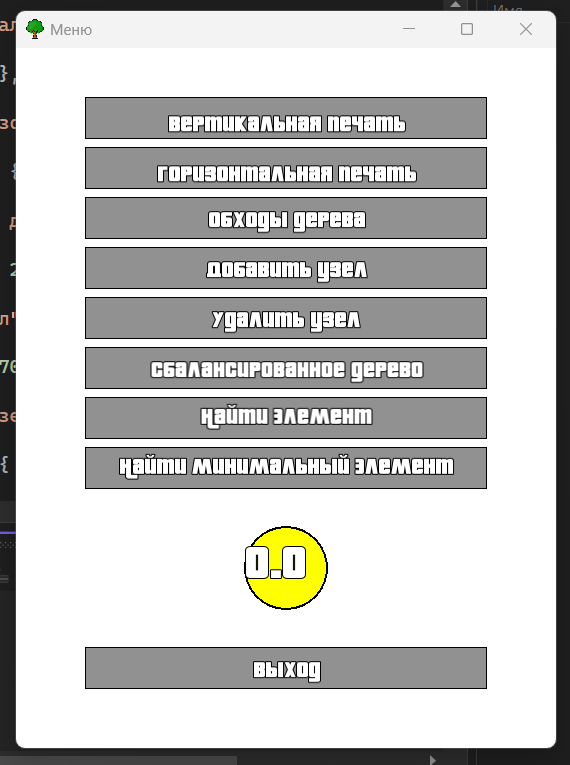
По ходу работы было разработано бинарное дерево, операции с которым

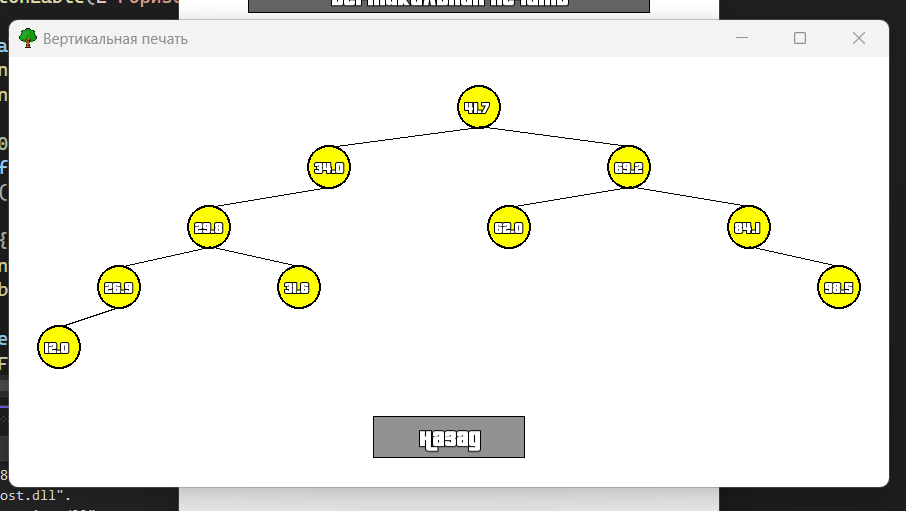
выполняются посредством с интерфейсом, разработанным с помощью библиотеки SFML. Были реализованы разные методы обхода дерева, функции печати дерева (вертикальная и горизонтальная), функции добавления и удаление узла дерева, а также функции поиска необходимого элемента. В коде были реализованы особые классы, которые реализуют кнопки и текстовые боксы для упрощения реализации интерфейса. По итогу работы было реализовано бинарное дерево, с меню, которое позволяет управлять им.

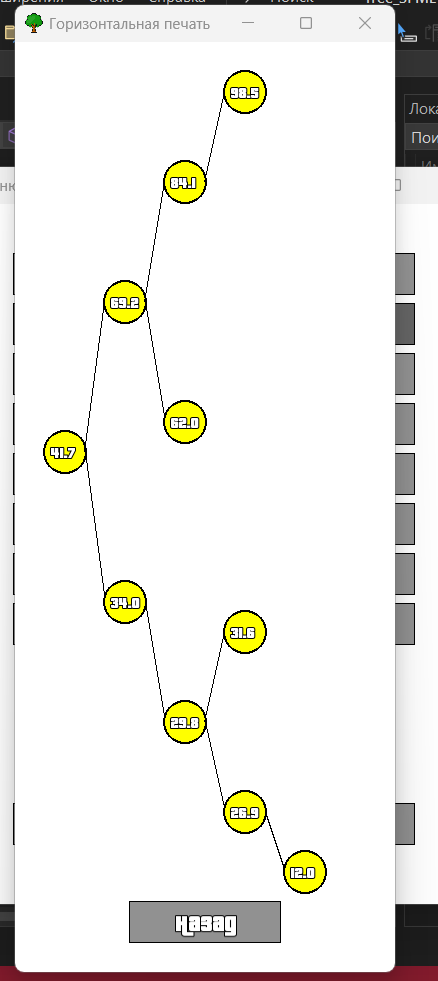
**GitHub**

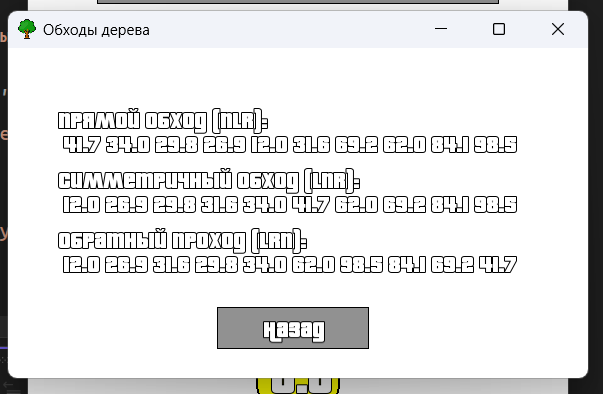
<https://github.com/Korovay4ik/Laboratory-works>

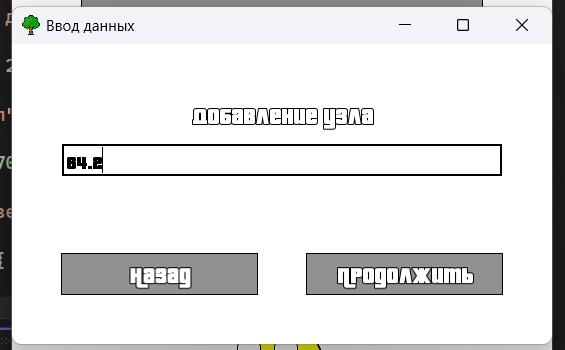
**Результат работы программы**

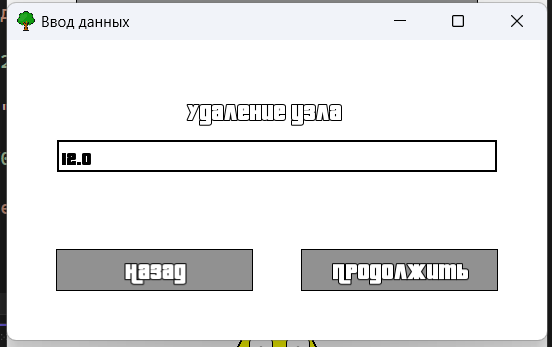


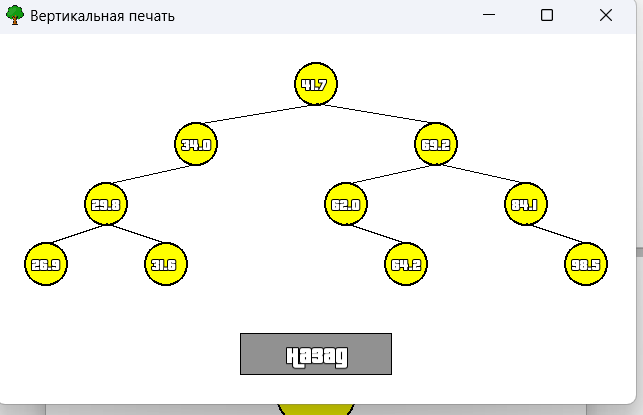


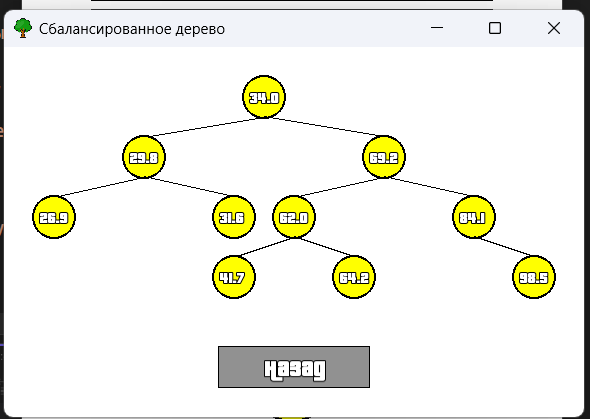


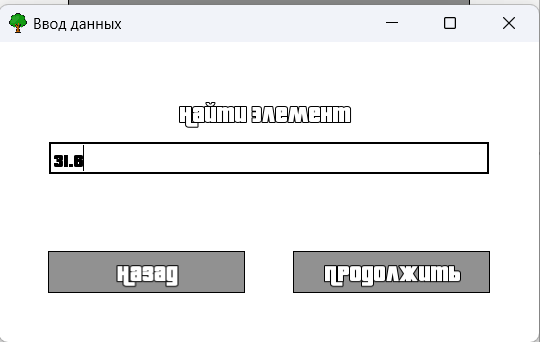
****

****

****

****

****

****

