Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление подготовки: Разработка информационных систем

**Лабораторная работа по бинарным деревьям**

Выполнил студент гр. РИС-24-3б

Килин Сергей Вадимович

Проверил:

Доц. каф. ИТАС

Полякова Ольга Андреевна

Пермь 2025

**Постановка задачи.**

1. Необходимо реализовать функции для редактирования дерева:

- Вставка узла.

- Удаление узла.

- Поиск элемента по ключу.

2. Реализовать алгоритмы обхода дерева:

2.1 Прямой

2.2 Симметричный

2.3 Обратный

3. Реализовать алгоритм балансировки дерева.

4. Реализовать вертикальную и горизонтальную печать.

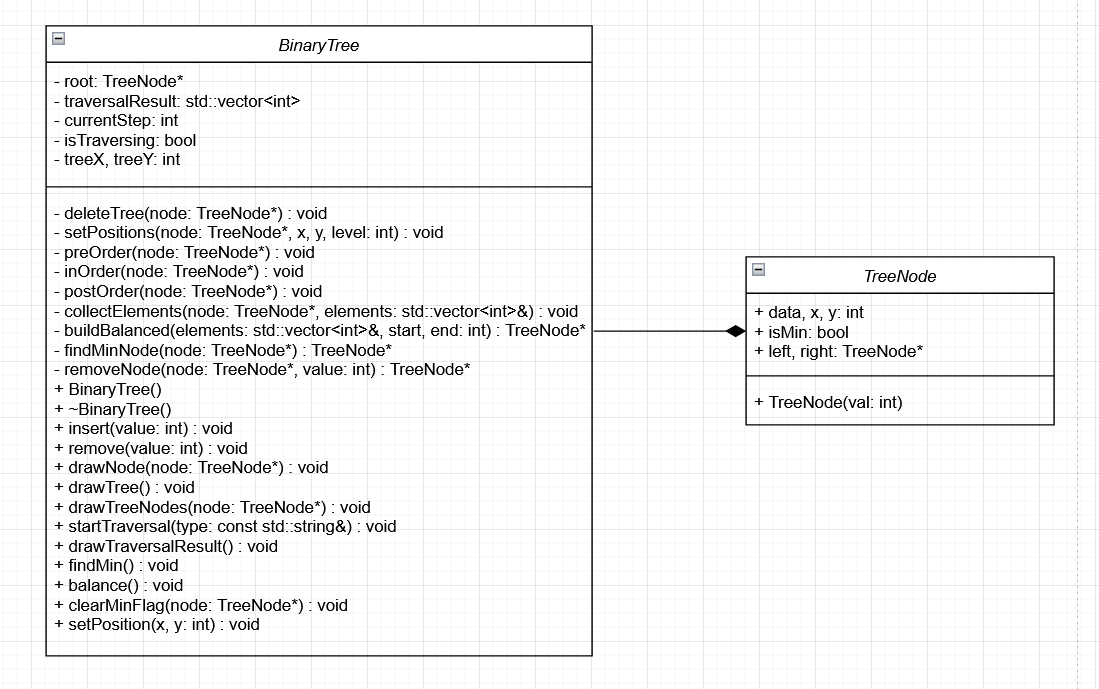
5. Визуализация дерева должна быть выполнена с использованием любой доступной графической библиотеки – SFML, SDL, OpenGL и подобных.

6. Пользовательский интерфейс по усмотрению разработчика - с условием кроссплатформенности (поощряется использование Qt или иных фреймворков).

6. Построить UML- диаграмму классов

7. Выполнить отчёт

**UML-диаграмма**

****

**Код в c++**

#include <GL/glut.h>

#include <vector>

#include <algorithm>

#include <string>

const int WINDOW\_WIDTH = 1100;

const int WINDOW\_HEIGHT = 800;

const int UI\_HEIGHT = 150;

const int BUTTON\_HEIGHT = 40;

const int BUTTON\_START\_Y = 110;

const int TRAVERSAL\_Y = 600;

class BinaryTree {

private:

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

int x, y;

bool isMin;

TreeNode(int val) : data(val), left(nullptr), right(nullptr),

x(0), y(0), isMin(false) {}

};

TreeNode\* root;

std::vector<int> traversalResult;

int currentStep;

bool isTraversing;

int treeX, treeY;

void deleteTree(TreeNode\* node) {

if (node) {

deleteTree(node->left);

deleteTree(node->right);

delete node;

}

}

void setPositions(TreeNode\* node, int x, int y, int level) {

if (!node) return;

node->x = x;

node->y = y;

int offset = 300 / (level + 1);

setPositions(node->left, x - offset, y + 80, level + 1);

setPositions(node->right, x + offset, y + 80, level + 1);

}

void preOrder(TreeNode\* node) {

if (!node) return;

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

preOrder(node->left);

preOrder(node->right);

}

void inOrder(TreeNode\* node) {

if (!node) return;

inOrder(node->left);

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

inOrder(node->right);

}

void postOrder(TreeNode\* node) {

if (!node) return;

postOrder(node->left);

postOrder(node->right);

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

}

void collectElements(TreeNode\* node, std::vector<int>& elements) {

if (!node) return;

collectElements(node->left, elements);

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

collectElements(node->right, elements);

}

TreeNode\* buildBalanced(std::vector<int>& elements, int start, int end) {

if (start > end) return nullptr;

int mid = (start + end) / 2;

TreeNode\* node = new TreeNode(elements[mid]);

node->left = buildBalanced(elements, start, mid - 1);

node->right = buildBalanced(elements, mid + 1, end);

return node;

}

TreeNode\* findMinNode(TreeNode\* node) {

while (node && node->left) node = node->left;

return node;

}

TreeNode\* removeNode(TreeNode\* node, int value) {

if (!node) return nullptr;

if (value < node->data) {

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

}

else if (value > node->data) {

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

}

else {

if (!node->left) {

TreeNode\* temp = node->right;

delete node;

return temp;

}

else if (!node->right) {

TreeNode\* temp = node->left;

delete node;

return temp;

}

TreeNode\* temp = findMinNode(node->right);

node->data = temp->data;

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

}

return node;

}

public:

BinaryTree() : root(nullptr), currentStep(0), isTraversing(false),

treeX(400), treeY(150) {}

~BinaryTree() {

deleteTree(root);

}

void insert(int value) {

if (!root) {

root = new TreeNode(value);

return;

}

TreeNode\* current = root;

while (true) {

if (value < current->data) {

if (!current->left) {

current->left = new TreeNode(value);

break;

}

current = current->left;

}

else {

if (!current->right) {

current->right = new TreeNode(value);

break;

}

current = current->right;

}

}

}

void remove(int value) {

root = removeNode(root, value);

}

void drawNode(TreeNode\* node) {

if (!node) return;

// Рисуем связи

if (node->left) {

glColor3f(1, 1, 1);

glBegin(GL\_LINES);

glVertex2i(node->x, node->y);

glVertex2i(node->left->x, node->left->y);

glEnd();

}

if (node->right) {

glColor3f(1, 1, 1);

glBegin(GL\_LINES);

glVertex2i(node->x, node->y);

glVertex2i(node->right->x, node->right->y);

glEnd();

}

// Рисуем узел

if (node->isMin) glColor3f(0.0f, 1.0f, 0.0f); // Зеленый для минимального

else glColor3f(0.4f, 0.7f, 1.0f); // Синий для обычных

glBegin(GL\_POLYGON);

for (int i = 0; i < 360; i++) {

float angle = i \* 3.14159f / 180;

glVertex2f(node->x + 20 \* cos(angle), node->y + 20 \* sin(angle));

}

glEnd();

// Текст

glColor3f(0, 0, 0);

glRasterPos2i(node->x - (node->data < 10 ? 5 : 10), node->y - 5);

std::string num = std::to\_string(node->data);

for (char c : num) glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, c);

}

void drawTree() {

if (root) {

setPositions(root, treeX, treeY, 1);

drawTreeNodes(root);

}

}

void drawTreeNodes(TreeNode\* node) {

if (!node) return;

drawNode(node);

drawTreeNodes(node->left);

drawTreeNodes(node->right);

}

void startTraversal(const std::string& type) {

traversalResult.clear();

currentStep = 0;

if (type == "PreOrder") preOrder(root);

else if (type == "InOrder") inOrder(root);

else if (type == "PostOrder") postOrder(root);

isTraversing = true;

}

void drawTraversalResult() {

if (!isTraversing || traversalResult.empty()) return;

glColor3f(1, 1, 1);

glRasterPos2i(20, TRAVERSAL\_Y);

std::string label = "Результат: ";

for (char c : label) glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, c);

// Выводим уже пройденные элементы

glColor3f(0, 1, 0); // Зеленый для пройденных

int x = 120;

for (int i = 0; i < currentStep && i < traversalResult.size(); i++) {

glRasterPos2i(x, TRAVERSAL\_Y);

std::string num = std::to\_string(traversalResult[i]) + " ";

for (char c : num) glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, c);

x += 30;

}

// Выводим текущий элемент (если есть)

if (currentStep < traversalResult.size()) {

glColor3f(1, 0, 0); // Красный для текущего

glRasterPos2i(x, TRAVERSAL\_Y);

std::string num = std::to\_string(traversalResult[currentStep]);

for (char c : num) glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, c);

currentStep++;

}

else {

isTraversing = false;

}

}

void findMin() {

clearMinFlag(root);

if (!root) return;

TreeNode\* current = root;

while (current->left) current = current->left;

current->isMin = true;

}

void balance() {

std::vector<int> elements;

collectElements(root, elements);

deleteTree(root);

root = buildBalanced(elements, 0, elements.size() - 1);

}

void clearMinFlag(TreeNode\* node) {

if (!node) return;

node->isMin = false;

clearMinFlag(node->left);

clearMinFlag(node->right);

}

void setPosition(int x, int y) {

treeX = x;

treeY = y;

}

};

BinaryTree tree;

std::string message = "Введите число и нажмите Enter";

std::string traversalType = "";

std::string inputStr = "";

void drawButton(int x, int y, int w, int h, const std::string& text) {

glColor3f(0.7f, 0.7f, 0.7f);

glBegin(GL\_QUADS);

glVertex2i(x, y);

glVertex2i(x + w, y);

glVertex2i(x + w, y + h);

glVertex2i(x, y + h);

glEnd();

glColor3f(0, 0, 0);

glRasterPos2i(x + 10, y + h / 2 + 5);

for (char c : text) glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, c);

}

void drawInterface() {

// Фон интерфейса

glColor3f(0.1f, 0.1f, 0.2f);

glBegin(GL\_QUADS);

glVertex2i(0, 0);

glVertex2i(WINDOW\_WIDTH, 0);

glVertex2i(WINDOW\_WIDTH, UI\_HEIGHT);

glVertex2i(0, UI\_HEIGHT);

glEnd();

// Текст

glColor3f(1, 1, 1);

glRasterPos2i(20, 20);

for (char c : message) glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, c);

glRasterPos2i(20, 50);

std::string inputText = "Ввод: " + inputStr;

for (char c : inputText) glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, c);

glRasterPos2i(20, 80);

for (char c : traversalType) glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, c);

// Кнопки

drawButton(20, BUTTON\_START\_Y, 100, BUTTON\_HEIGHT, "Add (A)");

drawButton(140, BUTTON\_START\_Y, 120, BUTTON\_HEIGHT, "PreOrder (P)");

drawButton(280, BUTTON\_START\_Y, 120, BUTTON\_HEIGHT, "InOrder (I)");

drawButton(420, BUTTON\_START\_Y, 120, BUTTON\_HEIGHT, "PostOrder (O)");

drawButton(560, BUTTON\_START\_Y, 120, BUTTON\_HEIGHT, "Find Min (M)");

drawButton(700, BUTTON\_START\_Y, 120, BUTTON\_HEIGHT, "Balance (B)");

drawButton(840, BUTTON\_START\_Y, 120, BUTTON\_HEIGHT, "Remove (D)");

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawInterface();

// Рисуем дерево

glPushMatrix();

glTranslatef(0, UI\_HEIGHT + 10, 0);

tree.drawTree();

glPopMatrix();

// Рисуем результат обхода

tree.drawTraversalResult();

glutSwapBuffers();

}

void keyboard(unsigned char key, int x, int y) {

switch (tolower(key)) {

case 13: // Enter

case 'a':

if (!inputStr.empty()) {

try {

int value = std::stoi(inputStr);

tree.insert(value);

message = "Добавлено: " + inputStr;

inputStr = "";

}

catch (...) {

message = "Ошибка ввода числа";

}

}

break;

case 'd':

if (!inputStr.empty()) {

try {

int value = std::stoi(inputStr);

tree.remove(value);

message = "Удалено: " + inputStr;

inputStr = "";

}

catch (...) {

message = "Ошибка ввода числа";

}

}

break;

case 'p':

tree.startTraversal("PreOrder");

traversalType = "PreOrder Traversal";

break;

case 'i':

tree.startTraversal("InOrder");

traversalType = "InOrder Traversal";

break;

case 'o':

tree.startTraversal("PostOrder");

traversalType = "PostOrder Traversal";

break;

case 'm':

tree.findMin();

message = "Найден минимальный элемент";

break;

case 'b':

tree.balance();

message = "Дерево сбалансировано";

break;

case 8: // Backspace

if (!inputStr.empty()) inputStr.pop\_back();

break;

default:

if (isdigit(key)) inputStr += key;

}

glutPostRedisplay();

}

void reshape(int width, int height) {

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, width, height, 0);

glMatrixMode(GL\_MODELVIEW);

tree.setPosition(width / 2, 150);

}

void update(int value) {

glutPostRedisplay();

glutTimerFunc(500, update, 0);

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

glutCreateWindow("Binary Tree Visualizer");

glClearColor(0.2f, 0.2f, 0.3f, 1.0f);

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutReshapeFunc(reshape);

glutTimerFunc(500, update, 0);

glutMainLoop();

return 0;

}

Ссылка на гитхаб: [https://github.com/asyehoo/](https://github.com/asyehoo/-)Labs