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.**

Реализовать алгоритмы для собственного варианта графа, имеющего не менее 6 вершин.

Алгоритмы:

1. Обход в ширину.

2. Обход в глубину.

3. Алгоритм Дейкстры.

4. Алгоритм Флойда.

Требования:

1. Пользовательский интерфейс на усмотрение разработчика с условием кроссплатформенности (поощряется использование Qt или иных фреймворков)

2. Визуализация графа с использованием любой доступной графической библиотеки (SFML, SDL, OpenGL и подобных)

3. Реализованные алгоритмы должны справляться как с графом, представленным в задании варианта, так и с другими на усмотрение проверяющего.

4. Необходимо реализовать функции для редактирования графа:

- Создание новой вершины.

- Удаление вершины.

- Добавление и удаление ребра.

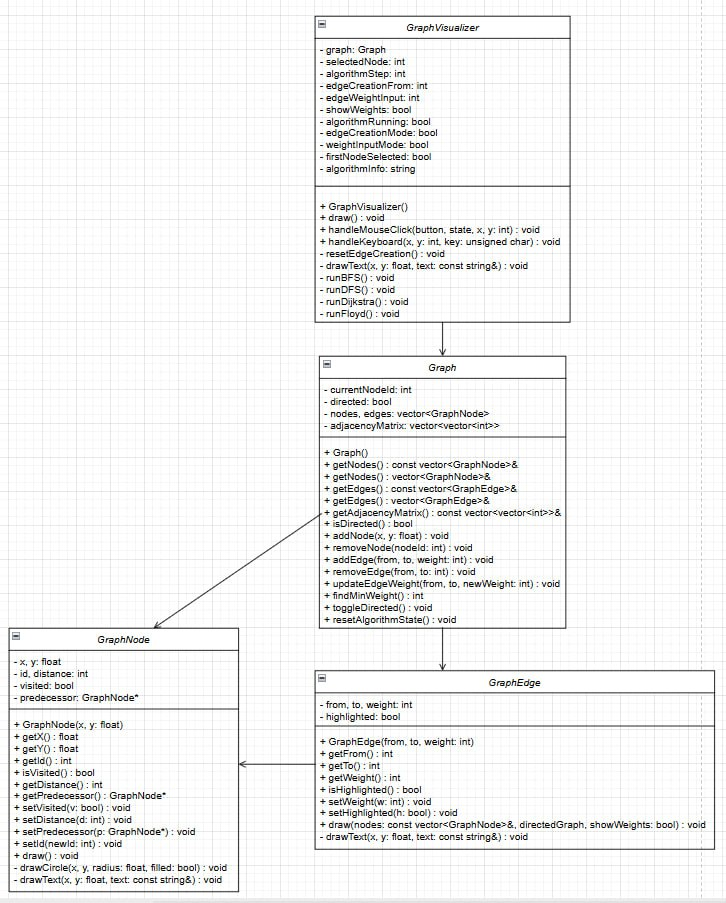
- Редактирование весов ребер.

- Редактирование матрицы смежности (или инцидентности в зависимости от реализации).

5. Построить UML- диаграммы классов

6. Выполнить отчет.

**UML-диаграмма**



**Код в c++**

#include <GL/glut.h>

#include <vector>

#include <queue>

#include <algorithm>

#include <limits>

#include <string>

#include <sstream>

#include <iostream>

using namespace std;

const int WINDOW\_WIDTH = 800;

const int WINDOW\_HEIGHT = 600;

const float NODE\_RADIUS = 20.0f;

const int INF = numeric\_limits<int>::max();

class GraphNode {

private:

float x, y;

int id;

bool visited;

int distance;

GraphNode\* predecessor;

public:

GraphNode(float x, float y, int id) : x(x), y(y), id(id), visited(false), distance(INF), predecessor(nullptr) {}

float getX() const { return x; }

float getY() const { return y; }

int getId() const { return id; }

bool isVisited() const { return visited; }

int getDistance() const { return distance; }

GraphNode\* getPredecessor() const { return predecessor; }

void setVisited(bool v) { visited = v; }

void setDistance(int d) { distance = d; }

void setPredecessor(GraphNode\* p) { predecessor = p; }

void setId(int newId) { id = newId; }

void draw() const {

if (visited) {

glColor3f(0.0f, 1.0f, 0.0f);

}

else {

glColor3f(0.0f, 0.0f, 1.0f);

}

drawCircle(x, y, NODE\_RADIUS, true);

glColor3f(1.0f, 1.0f, 1.0f);

drawCircle(x, y, NODE\_RADIUS, false);

glColor3f(1.0f, 1.0f, 1.0f);

string idText = to\_string(id);

float textWidth = idText.size() \* 6.0f;

drawText(x - textWidth / 2, y - 4, idText);

if (distance != INF) {

string distText = to\_string(distance);

drawText(x - textWidth / 2, y + NODE\_RADIUS + 10, distText);

}

}

private:

void drawCircle(float x, float y, float radius, bool filled) const {

glBegin(filled ? GL\_TRIANGLE\_FAN : GL\_LINE\_LOOP);

for (int i = 0; i < 360; i += 10) {

float angle = i \* 3.14159f / 180.0f;

glVertex2f(x + radius \* cos(angle), y + radius \* sin(angle));

}

glEnd();

}

void drawText(float x, float y, const string& text) const {

glRasterPos2f(x, y);

for (char c : text) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12, c);

}

}

};

class GraphEdge {

private:

int from, to;

int weight;

bool highlighted;

public:

GraphEdge(int from, int to, int weight) : from(from), to(to), weight(weight), highlighted(false) {}

int getFrom() const { return from; }

int getTo() const { return to; }

int getWeight() const { return weight; }

bool isHighlighted() const { return highlighted; }

void setWeight(int w) { weight = w; }

void setHighlighted(bool h) { highlighted = h; }

void draw(const vector<GraphNode>& nodes, bool directedGraph, bool showWeights) const {

const GraphNode& fromNode = nodes[from];

const GraphNode& toNode = nodes[to];

if (highlighted) {

glColor3f(1.0f, 0.0f, 0.0f);

}

else {

glColor3f(0.5f, 0.5f, 0.5f);

}

glBegin(GL\_LINES);

glVertex2f(fromNode.getX(), fromNode.getY());

glVertex2f(toNode.getX(), toNode.getY());

glEnd();

if (directedGraph) {

float angle = atan2(toNode.getY() - fromNode.getY(), toNode.getX() - fromNode.getX());

float arrowX = toNode.getX() - NODE\_RADIUS \* cos(angle);

float arrowY = toNode.getY() - NODE\_RADIUS \* sin(angle);

glBegin(GL\_TRIANGLES);

glVertex2f(arrowX, arrowY);

glVertex2f(arrowX - 10 \* cos(angle + 0.3), arrowY - 10 \* sin(angle + 0.3));

glVertex2f(arrowX - 10 \* cos(angle - 0.3), arrowY - 10 \* sin(angle - 0.3));

glEnd();

}

if (showWeights) {

float midX = (fromNode.getX() + toNode.getX()) / 2;

float midY = (fromNode.getY() + toNode.getY()) / 2;

glColor3f(0.0f, 0.0f, 0.0f);

drawText(midX, midY, to\_string(weight));

}

}

private:

void drawText(float x, float y, const string& text) const {

glRasterPos2f(x, y);

for (char c : text) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12, c);

}

}

};

class Graph {

private:

vector<GraphNode> nodes;

vector<GraphEdge> edges;

vector<vector<int>> adjacencyMatrix;

int currentNodeId;

bool directed;

public:

Graph() : currentNodeId(0), directed(false) {}

const vector<GraphNode>& getNodes() const { return nodes; }

vector<GraphNode>& getNodes() { return nodes; }

const vector<GraphEdge>& getEdges() const { return edges; }

vector<GraphEdge>& getEdges() { return edges; }

const vector<vector<int>>& getAdjacencyMatrix() const { return adjacencyMatrix; }

bool isDirected() const { return directed; }

void addNode(float x, float y) {

nodes.emplace\_back(x, y, currentNodeId++);

for (auto& row : adjacencyMatrix) {

row.push\_back(0);

}

adjacencyMatrix.emplace\_back(nodes.size(), 0);

}

void removeNode(int nodeId) {

nodes.erase(remove\_if(nodes.begin(), nodes.end(),

[nodeId](const GraphNode& n) { return n.getId() == nodeId; }), nodes.end());

edges.erase(remove\_if(edges.begin(), edges.end(),

[nodeId](const GraphEdge& e) { return e.getFrom() == nodeId e.getTo() == nodeId; }), edges.end());

if (!adjacencyMatrix.empty()) {

adjacencyMatrix.erase(adjacencyMatrix.begin() + nodeId);

for (auto& row : adjacencyMatrix) {

if (row.size() > static\_cast<size\_t>(nodeId)) {

row.erase(row.begin() + nodeId);

}

}

}

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

nodes[i].setId(static\_cast<int>(i));

}

currentNodeId = static\_cast<int>(nodes.size());

}

void addEdge(int from, int to, int weight) {

if (from >= 0 && from < static\_cast<int>(nodes.size()) &&

to >= 0 && to < static\_cast<int>(nodes.size())) {

edges.emplace\_back(from, to, weight);

adjacencyMatrix[from][to] = weight;

if (!directed) {

adjacencyMatrix[to][from] = weight;

}

}

}

void removeEdge(int from, int to) {

edges.erase(remove\_if(edges.begin(), edges.end(),

[from, to, this](const GraphEdge& e) {

return (e.getFrom() == from && e.getTo() == to)

(!this->directed && e.getFrom() == to && e.getTo() == from);

}), edges.end());

adjacencyMatrix[from][to] = 0;

if (!directed) {

adjacencyMatrix[to][from] = 0;

}

}

void updateEdgeWeight(int from, int to, int newWeight) {

for (auto& edge : edges) {

if ((edge.getFrom() == from && edge.getTo() == to)

(!directed && edge.getFrom() == to && edge.getTo() == from)) {

edge.setWeight(newWeight);

break;

}

}

adjacencyMatrix[from][to] = newWeight;

if (!directed) {

adjacencyMatrix[to][from] = newWeight;

}

}

int findMinWeight() const {

if (edges.empty()) return INF;

int minWeight = edges[0].getWeight();

for (const auto& edge : edges) {

if (edge.getWeight() < minWeight) {

minWeight = edge.getWeight();

}

}

return minWeight;

}

void toggleDirected() {

directed = !directed;

}

void resetAlgorithmState() {

for (auto& node : nodes) {

node.setVisited(false);

node.setDistance(INF);

node.setPredecessor(nullptr);

}

for (auto& edge : edges) {

edge.setHighlighted(false);

}

}

};

class GraphVisualizer {

private:

Graph graph;

int selectedNode;

bool showWeights;

string algorithmInfo;

bool algorithmRunning;

int algorithmStep;

bool edgeCreationMode;

int edgeCreationFrom;

int edgeWeightInput;

bool weightInputMode;

bool firstNodeSelected;

public:

GraphVisualizer() : selectedNode(-1), showWeights(true), algorithmRunning(false),

algorithmStep(0), edgeCreationMode(false), edgeCreationFrom(-1),

edgeWeightInput(1), weightInputMode(false), firstNodeSelected(false) {}

void draw() {

// Рисуем ребра

for (const auto& edge : graph.getEdges()) {

edge.draw(graph.getNodes(), graph.isDirected(), showWeights);

}

// Рисуем узлы

for (const auto& node : graph.getNodes()) {

node.draw();

}

// Отображаем информацию об алгоритме

glColor3f(0.0f, 0.0f, 0.0f);

drawText(10, WINDOW\_HEIGHT - 20, algorithmInfo);

// Инструкции

drawText(10, 20, "Left click: Add node | Right click: Select node");

drawText(10, 40, "E: Add edge | W: Edit weight | D: Delete node");

drawText(10, 60, "T: Toggle directed | 1-4: Algorithms | ESC: Cancel");

// Режим создания ребра

if (edgeCreationMode) {

if (firstNodeSelected) {

drawText(10, 80, "Select second node for edge");

}

else {

drawText(10, 80, "Select first node for edge");

}

}

// Режим ввода веса

if (weightInputMode) {

drawText(10, 100, "Enter weight (1-9): " + to\_string(edgeWeightInput));

}

// Выбранная вершина

if (selectedNode != -1) {

drawText(10, 120, "Selected node: " + to\_string(selectedNode));

}

}

void handleMouseClick(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

if (weightInputMode) return;

// Проверяем, не кликнули ли по существующей вершине

int clickedNode = -1;

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

float dx = graph.getNodes()[i].getX() - x;

float dy = graph.getNodes()[i].getY() - y;

if (dx \* dx + dy \* dy <= NODE\_RADIUS \* NODE\_RADIUS) {

clickedNode = static\_cast<int>(i);

break;

}

}

if (edgeCreationMode) {

if (clickedNode != -1) {

if (!firstNodeSelected) {

// Выбрали первую вершину для ребра

edgeCreationFrom = clickedNode;

firstNodeSelected = true;

}

else {

// Выбрали вторую вершину, переходим к вводу веса

weightInputMode = true;

edgeWeightInput = 1;

}

}

else if (!firstNodeSelected) {

// Добавляем новую вершину

graph.addNode(static\_cast<float>(x), static\_cast<float>(y));

}

}

else {

// Обычный режим - добавляем новую вершину

if (clickedNode == -1) {

graph.addNode(static\_cast<float>(x), static\_cast<float>(y));

}

}

}

else if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN) {

// Выбор вершины правой кнопкой

selectedNode = -1;

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

float dx = graph.getNodes()[i].getX() - x;

float dy = graph.getNodes()[i].getY() - y;

if (dx \* dx + dy \* dy <= NODE\_RADIUS \* NODE\_RADIUS) {

selectedNode = static\_cast<int>(i);

break;

}

}

}

}

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

if (weightInputMode) {

if (key >= '1' && key <= '9') {

edgeWeightInput = key - '0';

}

else if (key == 13) { // Enter

// Завершаем создание ребра

if (firstNodeSelected && selectedNode != -1 && selectedNode != edgeCreationFrom) {

graph.addEdge(edgeCreationFrom, selectedNode, edgeWeightInput);

}

resetEdgeCreation();

}

else if (key == 27) { // Escape

resetEdgeCreation();

}

glutPostRedisplay();

return;

}

switch (key) {

case 'e': case 'E':

if (!edgeCreationMode) {

edgeCreationMode = true;

firstNodeSelected = false;

algorithmInfo = "Edge creation mode";

}

break;

case 'w': case 'W':

if (selectedNode != -1) {

// Находим первое ребро, связанное с выбранной вершиной

for (auto& edge : graph.getEdges()) {

if (edge.getFrom() == selectedNode edge.getTo() == selectedNode) {

weightInputMode = true;

edgeWeightInput = edge.getWeight();

break;

}

}

}

break;

case 'd': case 'D':

if (selectedNode != -1) {

graph.removeNode(selectedNode);

selectedNode = -1;

}

break;

case 't': case 'T':

graph.toggleDirected();

algorithmInfo = graph.isDirected() ? "Directed graph" : "Undirected graph";

break;

case '1': runBFS(); break;

case '2': runDFS(); break;

case '3': runDijkstra(); break;

case '4': runFloyd(); break;

case 27: // Escape

resetEdgeCreation();

break;

}

glutPostRedisplay();

}

private:

void resetEdgeCreation() {

edgeCreationMode = false;

weightInputMode = false;

firstNodeSelected = false;

edgeCreationFrom = -1;

}

void drawText(float x, float y, const string& text) {

glRasterPos2f(x, y);

for (char c : text) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12, c);

}

}

void runBFS() {

if (!graph.getNodes().empty() && !algorithmRunning) {

graph.resetAlgorithmState();

algorithmRunning = true;

algorithmStep = 0;

algorithmInfo = "BFS started - press SPACE for next step";

}

}

void runDFS() {

if (!graph.getNodes().empty() && !algorithmRunning) {

graph.resetAlgorithmState();

algorithmRunning = true;

algorithmStep = 1;

algorithmInfo = "DFS started - press SPACE for next step";

}

}

void runDijkstra() {

if (!graph.getNodes().empty() && !algorithmRunning) {

graph.resetAlgorithmState();

algorithmRunning = true;

algorithmStep = 2;

algorithmInfo = "Dijkstra started - press SPACE for next step";

}

}

void runFloyd() {

if (!graph.getNodes().empty() && !algorithmRunning) {

graph.resetAlgorithmState();

algorithmRunning = true;

algorithmStep = 3;

algorithmInfo = "Floyd started - press SPACE for next step";

}

}

};

GraphVisualizer visualizer;

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

visualizer.draw();

glutSwapBuffers();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, w, h, 0);

glMatrixMode(GL\_MODELVIEW);

}

void mouse(int button, int state, int x, int y) {

visualizer.handleMouseClick(button, state, x, y);

glutPostRedisplay();

}

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

visualizer.handleKeyboard(key, x, y);

}

void timer(int value) {

glutPostRedisplay();

glutTimerFunc(1000, timer, 0);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

glutCreateWindow("Graph Builder - Start from scratch");

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

glutTimerFunc(1000, timer, 0);

glutMainLoop();

return 0;

}

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