Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учереждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление подготовки: Разработка информационных систем (РИС)

**Лабораторная работа “Бинарные деревья”**

Выполнил студент гр. РИС-24-3б

Караваев Артем Андреевич

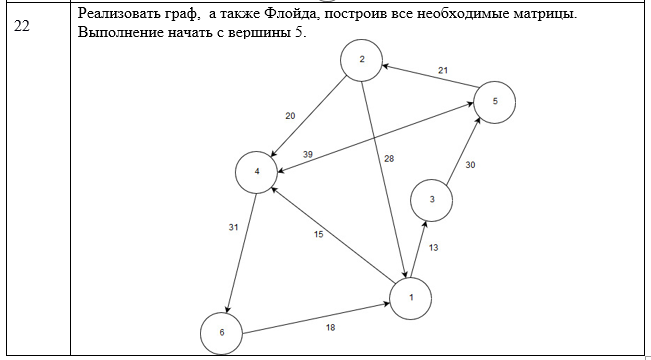
Проверил:

Доц. каф. ИТАС

Ольга Андреевна Полякова

г. Пермь, 2024

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.

Код программы:

#include <SFML/Graphics.hpp>

#include <iostream>

#include <vector>

#include <queue>

#include <sstream>

#include <algorithm>

#include <cmath>

#include <map>

#include <climits>

using namespace std;

using namespace sf;

class Graph {

private:

int maxSize;

struct Route {

vector<int> Verts;

int weight = 0;

bool isVertin(int val) {

return find(Verts.begin(), Verts.end(), val) != Verts.end();

}

};

public:

int\*\* Floidmatrix;

int\*\* Floidmatrix1;

vector<int> vertList;

int\*\* matrix;

Graph(int s) :maxSize(s) {

matrix = new int\* [maxSize];

Floidmatrix = new int\* [maxSize];

Floidmatrix1 = new int\* [maxSize];

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

matrix[i] = new int[maxSize]();

Floidmatrix[i] = new int[maxSize]();

Floidmatrix1[i] = new int[maxSize]();

}

vertList = { 6,4,2,3,5,1 };

int tempmatrix[6][6]{

{0, 0, 0, 0, 0, 18} ,

{31, 0, 0, 0, 0, 0},

{0, 20, 0, 39, 0, 28},

{0, 0, 0, 0, 30, 0},

{0, 0, 21, 28, 0, 30 },

{0, 15, 0, 13, 30, 0}

};

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

for (int j = 0; j < maxSize; j++) {

matrix[i][j] = tempmatrix[i][j];

}

}

}

~Graph() {

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

delete[] matrix[i];

delete[] Floidmatrix[i];

delete[] Floidmatrix1[i];

}

delete[] matrix;

delete[] Floidmatrix;

delete[] Floidmatrix1;

}

int GetVertPos(int val) {

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

if (vertList[i] == val) {

return i;

}

}

return -1;

}

bool isEmpty() {

return vertList.empty();

}

int GetEdges() {

int k = 0;

if (!isEmpty()) {

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

for (int j = 0; j < maxSize; j++) {

if (matrix[i][j] != 0) k += 1;

}

}

return k;

}

return 0;

}

int GetWeight(int val, int val1) {

if (!isEmpty()) {

int x1 = GetVertPos(val);

int x2 = GetVertPos(val1);

return matrix[x1][x2];

}

return 0;

}

vector<int> GetN(int val) {

vector<int> vec;

int x = GetVertPos(val);

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

if (matrix[x][i] != 0) {

int y = vertList[i];

vec.push\_back(y);

}

}

return vec;

}

void Insert(int val) {

if (vertList.size() < maxSize && GetVertPos(val) == -1) {

int newSize = vertList.size() + 1;

int\*\* newMatrix = new int\* [newSize];

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

newMatrix[i] = new int[newSize]();

for (int j = 0; j < newSize; j++) {

if (i < vertList.size() && j < vertList.size()) {

newMatrix[i][j] = matrix[i][j];

}

}

}

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

delete[] matrix[i];

}

delete[] matrix;

matrix = newMatrix;

vertList.push\_back(val);

}

}

void Remove(int val) {

int pos = GetVertPos(val);

if (pos == -1) return;

int newSize = vertList.size() - 1;

int\*\* newMatrix = new int\* [newSize];

for (int i = 0, new\_i = 0; i < vertList.size(); i++) {

if (i == pos) continue;

newMatrix[new\_i] = new int[newSize];

for (int j = 0, new\_j = 0; j < vertList.size(); j++) {

if (j == pos) continue;

newMatrix[new\_i][new\_j] = matrix[i][j];

new\_j++;

}

new\_i++;

}

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

delete[] matrix[i];

}

delete[] matrix;

matrix = newMatrix;

vertList.erase(vertList.begin() + pos);

}

void InsertEdge(int val, int val1, int weight) {

int x1 = GetVertPos(val);

int x2 = GetVertPos(val1);

if (x1 != -1 && x2 != -1) {

matrix[x1][x2] = weight;

}

}

void RemoveEdge(int val, int val1) {

int x1 = GetVertPos(val);

int x2 = GetVertPos(val1);

if (x1 != -1 && x2 != -1) {

matrix[x1][x2] = 0;

}

}

vector<int> initisVis() {

return vector<int>(vertList.size(), 0);

}

void glub(int val, vector<int>& isVis, vector<int>& out) {

out.push\_back(val);

isVis[GetVertPos(val)] = -1;

vector<int> neighbors = GetN(val);

for (int neighbor : neighbors) {

int pos = GetVertPos(neighbor);

if (isVis[pos] != -1) {

glub(neighbor, isVis, out);

}

}

}

void shir(int val, vector<int>& isVis, vector<int>& out) {

queue<int> q;

q.push(val);

isVis[GetVertPos(val)] = -1;

while (!q.empty()) {

int cur = q.front();

out.push\_back(cur);

q.pop();

vector<int> neighbors = GetN(cur);

for (int neighbor : neighbors) {

int pos = GetVertPos(neighbor);

if (isVis[pos] != -1) {

q.push(neighbor);

isVis[pos] = -1;

}

}

}

}

bool Vis(const vector<int>& isVis) {

return any\_of(isVis.begin(), isVis.end(), [](int v) { return v != -1; });

}

int min(const vector<int>& label, const vector<int>& isVis) {

int minVal = INT\_MAX;

int minIndex = -1;

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

if (label[i] < minVal && isVis[i] != -1) {

minVal = label[i];

minIndex = i;

}

}

return minIndex;

}

vector<int> initlabel() {

return vector<int>(vertList.size(), INT\_MAX);

}

int Deikstri(int val, int val1, vector<int>& label, vector<int>& isVis) {

int x = GetVertPos(val);

label[x] = 0;

isVis[x] = -1;

while (Vis(isVis)) {

int h = GetVertPos(min(label, isVis));

vector<int> vec;

vec = GetN(h);

isVis[h] = -1;

for (auto z : vec) {

int g = GetVertPos(z);

if (isVis[g] != -1) {

if (matrix[h][g] > 0) {

label[g] = (matrix[h][g] + label[h] < label[g] ? matrix[h][g] + label[h] : label[g]);

}

}

}

}

int y = GetVertPos(val1);

return label[y];

}

void initFloidmatrix() {

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

for (int j = 0; j < vertList.size(); j++) {

if (i == j) {

Floidmatrix[i][j] = 0;

}

else if (matrix[i][j] == 0) {

Floidmatrix[i][j] = INT\_MAX;

}

else {

Floidmatrix[i][j] = matrix[i][j];

}

}

}

}

void initFloidmatrix1() {

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

for (int j = 0; j < vertList.size(); j++) {

if (matrix[i][j] == 0) {

Floidmatrix1[i][j] = -1;

}

else {

Floidmatrix1[i][j] = j;

}

}

}

}

void Floid() {

initFloidmatrix();

initFloidmatrix1();

for (int k = 0; k < vertList.size(); k++) {

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

for (int j = 0; j < vertList.size(); j++) {

if (Floidmatrix[i][k] != INT\_MAX && Floidmatrix[k][j] != INT\_MAX &&

Floidmatrix[i][j] > Floidmatrix[i][k] + Floidmatrix[k][j]) {

Floidmatrix[i][j] = Floidmatrix[i][k] + Floidmatrix[k][j];

Floidmatrix1[i][j] = Floidmatrix1[i][k];

}

}

}

}

}

string PrintFloid(int start, int end) {

int startPos = GetVertPos(start);

int endPos = GetVertPos(end);

if (startPos == -1 || endPos == -1 || Floidmatrix[startPos][endPos] == INT\_MAX) {

return "No path exists";

}

string path = to\_string(start) + "->";

while (startPos != endPos) {

startPos = Floidmatrix1[startPos][endPos];

path += to\_string(vertList[startPos]) + "->";

}

path.pop\_back();

path.pop\_back();

return path;

}

};

class GraphVisualizer {

private:

Graph& graph;

int currentVisual;

RenderWindow& window;

Font font;

vector<CircleShape> nodes;

vector<Text> nodeTexts;

vector<VertexArray> lines;

vector<RectangleShape> buttons;

vector<Text> buttonTexts;

RectangleShape inputBox;

Text inputText;

Text infoText;

string inputString;

bool inputActive;

int traversalType;

Clock traversalClock;

int insertedgepend;

bool removeedgepend;

bool deikstripend;

bool floidpend;

int val1;

int val2;

vector<int> out;

vector<ConvexShape> arrowheads;

vector<Text> edgeWeights;

public:

public:

GraphVisualizer(Graph& gra, RenderWindow& w) :

graph(gra), window(w), inputActive(false), traversalType(0),

insertedgepend(0), removeedgepend(false), deikstripend(false), floidpend(false) {

if (!font.loadFromFile("C:\\Users\\Ярослав\\Downloads\\Telegram Desktop\\arialmt.ttf")) {

cerr << "Failed to load font\n";

}

createButton(20, 20, 150, 40, "Insert Vert", Color::Green);

createButton(190, 20, 150, 40, "Remove Vert", Color::Red);

createButton(360, 20, 150, 40, "Insert Edge", Color::Cyan);

createButton(530, 20, 150, 40, "Remove Edge", Color::Magenta);

createButton(700, 20, 150, 40, "Obhod v glubinu", Color::Yellow);

createButton(20, 80, 150, 40, "Obhod v shirinu", Color::Blue);

createButton(190, 80, 150, 40, "Dijkstra", Color(255, 165, 0));

createButton(360, 80, 150, 40, "Floyd", Color(200, 200, 255));

inputBox.setSize(Vector2f(200, 30));

inputBox.setPosition(530, 80);

inputBox.setFillColor(Color::White);

inputBox.setOutlineThickness(2);

inputBox.setOutlineColor(Color::Black);

inputText.setFont(font);

inputText.setCharacterSize(20);

inputText.setFillColor(Color::Black);

inputText.setPosition(535, 80);

infoText.setFont(font);

infoText.setCharacterSize(20);

infoText.setFillColor(Color::White);

infoText.setPosition(20, 550);

updateVisualization();

}

void createButton(float x, float y, float width, float height, const string& label, Color color) {

RectangleShape button(Vector2f(width, height));

button.setPosition(x, y);

button.setFillColor(color);

buttons.push\_back(button);

Text text;

text.setFont(font);

text.setString(label);

text.setCharacterSize(20);

text.setFillColor(Color::Black);

FloatRect textRect = text.getLocalBounds();

text.setOrigin(textRect.left + textRect.width / 2.0f, textRect.top + textRect.height / 2.0f);

text.setPosition(x + width / 2.0f, y + height / 2.0f);

buttonTexts.push\_back(text);

}

void handleEvent(Event& event) {

if (event.type == Event::MouseButtonPressed && event.mouseButton.button == Mouse::Left) {

Vector2f mousePos(event.mouseButton.x, event.mouseButton.y);

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

if (buttons[i].getGlobalBounds().contains(mousePos)) {

handleButtonClick(i);

break;

}

}

inputActive = inputBox.getGlobalBounds().contains(mousePos);

inputBox.setOutlineColor(inputActive ? Color::Blue : Color::Black);

}

if (event.type == Event::TextEntered && inputActive) {

if (event.text.unicode == '\b' && !inputString.empty()) {

inputString.pop\_back();

}

else if (event.text.unicode >= '0' && event.text.unicode <= '9') {

inputString += static\_cast<char>(event.text.unicode);

}

inputText.setString(inputString);

}

}

void handleButtonClick(size\_t buttonIndex) {

if (buttonIndex >= buttons.size()) return;

string buttonText = buttonTexts[buttonIndex].getString();

stringstream infoStream;

if (buttonText == "Insert Vert" && !inputString.empty()) {

int value = stoi(inputString);

graph.Insert(value);

infoStream << "Inserted Vertex: " << value;

}

else if (buttonText == "Remove Vert" && !inputString.empty()) {

int value = stoi(inputString);

graph.Remove(value);

infoStream << "Removed Vertex: " << value;

}

else if (buttonText == "Insert Edge") {

if (insertedgepend == 0 && !inputString.empty()) {

val1 = stoi(inputString);

insertedgepend++;

infoStream << "Insert Edge - Enter second vertex";

}

else if (insertedgepend == 1 && !inputString.empty()) {

val2 = stoi(inputString);

insertedgepend++;

infoStream << "Insert Edge " << val1 << "->" << val2 << " - Enter weight";

}

else if (insertedgepend == 2 && !inputString.empty()) {

int weight = stoi(inputString);

graph.InsertEdge(val1, val2, weight);

insertedgepend = 0;

infoStream << "Inserted Edge: " << val1 << "->" << val2 << " (weight: " << weight << ")";

}

}

else if (buttonText == "Remove Edge") {

if (!removeedgepend && !inputString.empty()) {

val1 = stoi(inputString);

removeedgepend = true;

infoStream << "Remove Edge - Enter destination vertex";

}

else if (removeedgepend && !inputString.empty()) {

val2 = stoi(inputString);

graph.RemoveEdge(val1, val2);

removeedgepend = false;

infoStream << "Removed Edge: " << val1 << "->" << val2;

}

}

else if (buttonText == "Dijkstra") {

if (!deikstripend && !inputString.empty()) {

val1 = stoi(inputString);

deikstripend = true;

infoStream << "Dijkstra - Enter destination vertex";

}

else if (deikstripend && !inputString.empty()) {

val2 = stoi(inputString);

vector<int> isVis = graph.initisVis();

vector<int> label = graph.initlabel();

int result = graph.Deikstri(val1, val2, label, isVis);

deikstripend = false;

infoStream << "Dijkstra: " << val1 << "->" << val2 << " = " << result;

}

}

else if (buttonText == "Floyd") {

if (!floidpend && !inputString.empty()) {

val1 = stoi(inputString);

floidpend = true;

infoStream << "Floyd - Enter destination vertex";

}

else if (floidpend && !inputString.empty()) {

val2 = stoi(inputString);

graph.Floid();

int result = graph.Floidmatrix[graph.GetVertPos(val1)][graph.GetVertPos(val2)];

string path = graph.PrintFloid(val1, val2);

floidpend = false;

infoStream << "Floyd: " << val1 << "->" << val2 << " = " << result << " Path: " << path;

}

}

else if (buttonText == "Obhod v glubinu") {

vector<int> isVis = graph.initisVis();

out.clear();

graph.glub(graph.vertList[0], isVis, out);

traversalType = 1;

traversalClock.restart();

infoStream << "Obhod v glubinu started";

}

else if (buttonText == "Obhod v shirinu") {

vector<int> isVis = graph.initisVis();

out.clear();

graph.shir(graph.vertList[0], isVis, out);

traversalType = 2;

traversalClock.restart();

infoStream << "Obhod v shirinu started";

}

if (buttonText != "Insert Edge" && buttonText != "Remove Edge" &&

buttonText != "Dijkstra" && buttonText != "Floyd") {

inputString.clear();

inputText.setString(inputString);

}

infoText.setString(infoStream.str());

updateVisualization();

}

void update(float deltaTime) {

if (traversalType > 0 && traversalClock.getElapsedTime().asSeconds() > 0.5f) {

if (!out.empty()) {

currentVisual = out[0];

out.erase(out.begin());

traversalClock.restart();

updateVisualization();

}

else {

traversalType = 0;

}

}

}

void updateVisualization() {

nodes.clear();

nodeTexts.clear();

lines.clear();

arrowheads.clear();

edgeWeights.clear();

if (graph.isEmpty()) return;

float centerX = window.getSize().x / 2;

float centerY = (window.getSize().y + 100) / 2;

float radius = 200;

int vertexCount = graph.vertList.size();

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

float angle = 2 \* 3.14159265f \* i / vertexCount;

float x = centerX + radius \* cos(angle);

float y = centerY + radius \* sin(angle);

CircleShape circle(25);

circle.setPosition(x - 25, y - 25);

circle.setFillColor(Color::White);

circle.setOutlineThickness(2);

circle.setOutlineColor(Color::Black);

if (traversalType > 0 && graph.vertList[i] == currentVisual) {

circle.setFillColor(Color::Green);

}

nodes.push\_back(circle);

Text text;

text.setFont(font);

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

text.setCharacterSize(20);

text.setFillColor(Color::Black);

FloatRect textRect = text.getLocalBounds();

text.setOrigin(textRect.left + textRect.width / 2.0f,

textRect.top + textRect.height / 2.0f);

text.setPosition(x, y);

nodeTexts.push\_back(text);

}

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

for (int j = 0; j < vertexCount; ++j) {

if (graph.matrix[i][j] > 0) {

Vector2f start = nodes[i].getPosition() + Vector2f(25, 25);

Vector2f end = nodes[j].getPosition() + Vector2f(25, 25);

Vector2f direction = end - start;

float length = sqrt(direction.x \* direction.x + direction.y \* direction.y);

Vector2f unitDir = direction / length;

start += unitDir \* 25.f;

end -= unitDir \* 25.f;

VertexArray line(Lines, 2);

line[0].position = start;

line[1].position = end;

line[0].color = Color::Black;

line[1].color = Color::Black;

lines.push\_back(line);

if (graph.matrix[i][j] != graph.matrix[j][i]) {

ConvexShape arrow(3);

arrow.setPoint(0, end);

arrow.setPoint(1, end - unitDir \* 10.f + Vector2f(-unitDir.y, unitDir.x) \* 5.f);

arrow.setPoint(2, end - unitDir \* 10.f + Vector2f(unitDir.y, -unitDir.x) \* 5.f);

arrow.setFillColor(Color::Black);

arrowheads.push\_back(arrow);

}

Text weightText;

weightText.setFont(font);

weightText.setString(to\_string(graph.matrix[i][j]));

weightText.setCharacterSize(16);

weightText.setFillColor(Color::Red);

weightText.setPosition((start + end) / 2.f);

edgeWeights.push\_back(weightText);

}

}

}

}

void draw() {

window.clear(Color(50, 50, 50));

for (const auto& line : lines) {

window.draw(line);

}

for (const auto& arrow : arrowheads) {

window.draw(arrow);

}

for (const auto& weight : edgeWeights) {

window.draw(weight);

}

for (const auto& node : nodes) {

window.draw(node);

}

for (const auto& text : nodeTexts) {

window.draw(text);

}

for (const auto& button : buttons) {

window.draw(button);

}

for (const auto& text : buttonTexts) {

window.draw(text);

}

window.draw(inputBox);

window.draw(inputText);

window.draw(infoText);

window.display();

}

};

int main() {

RenderWindow window(VideoMode(1000, 600), "Graph Visualizer");

Graph graph(6);

GraphVisualizer visualizer(graph, window);

Clock clock;

while (window.isOpen()) {

Event event;

while (window.pollEvent(event)) {

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

window.close();

}

visualizer.handleEvent(event);

}

float deltaTime = clock.restart().asSeconds();

visualizer.update(deltaTime);

visualizer.draw();

}

return 0;

}

float deltaTime = clock.restart().asSeconds();

visualizer.update(deltaTime);

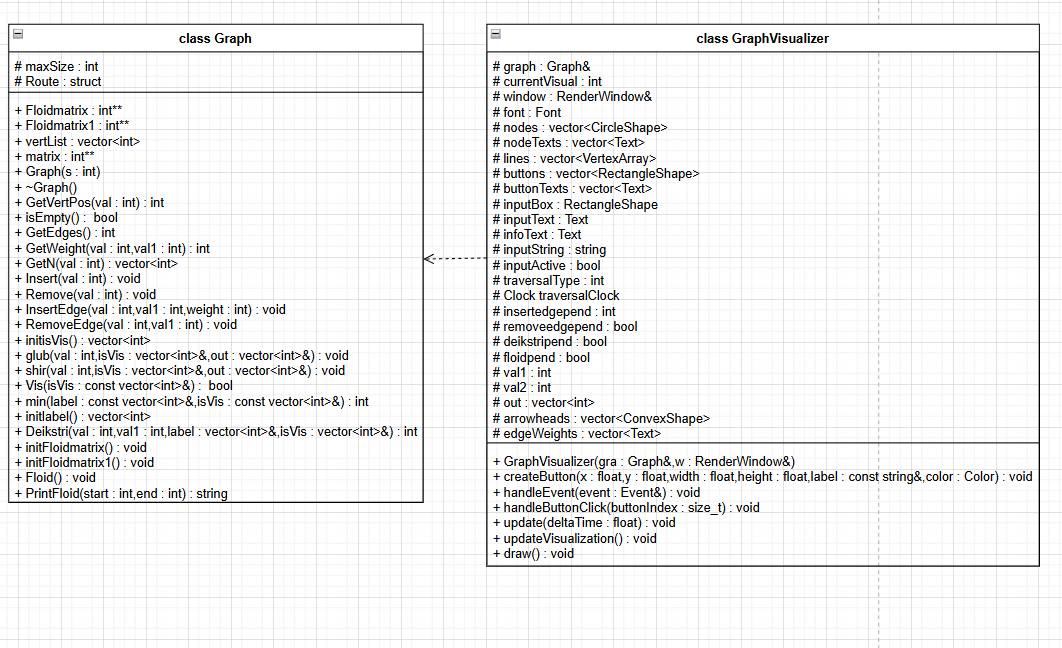
visualizer.draw();

}

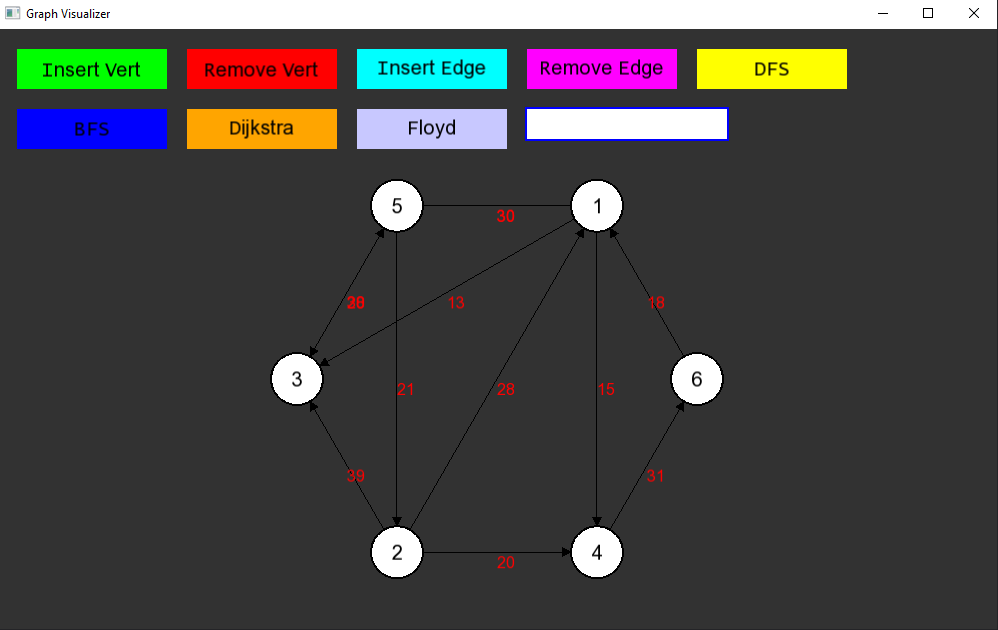
return 0;

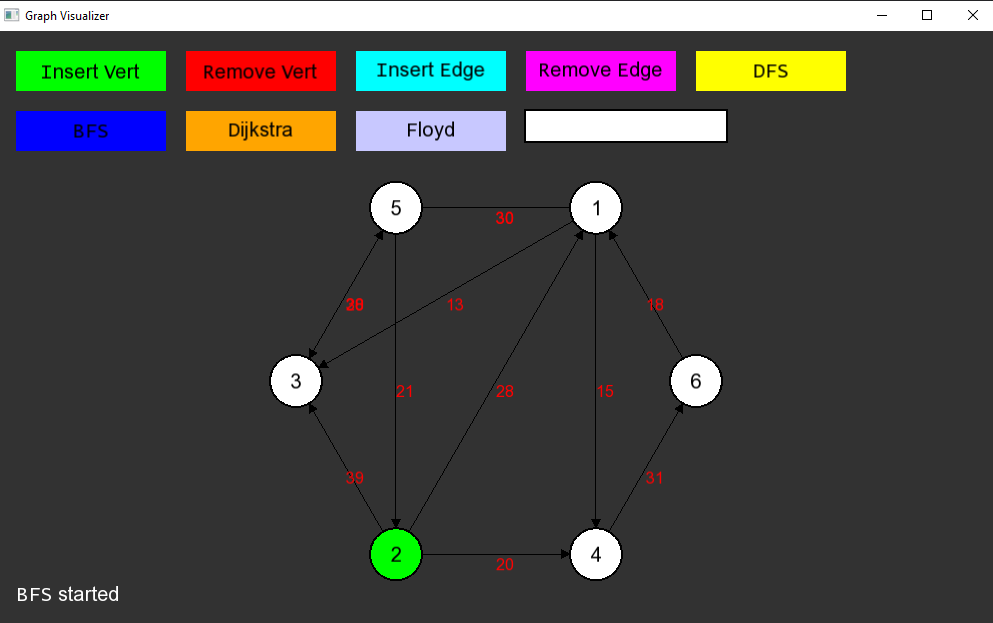
}

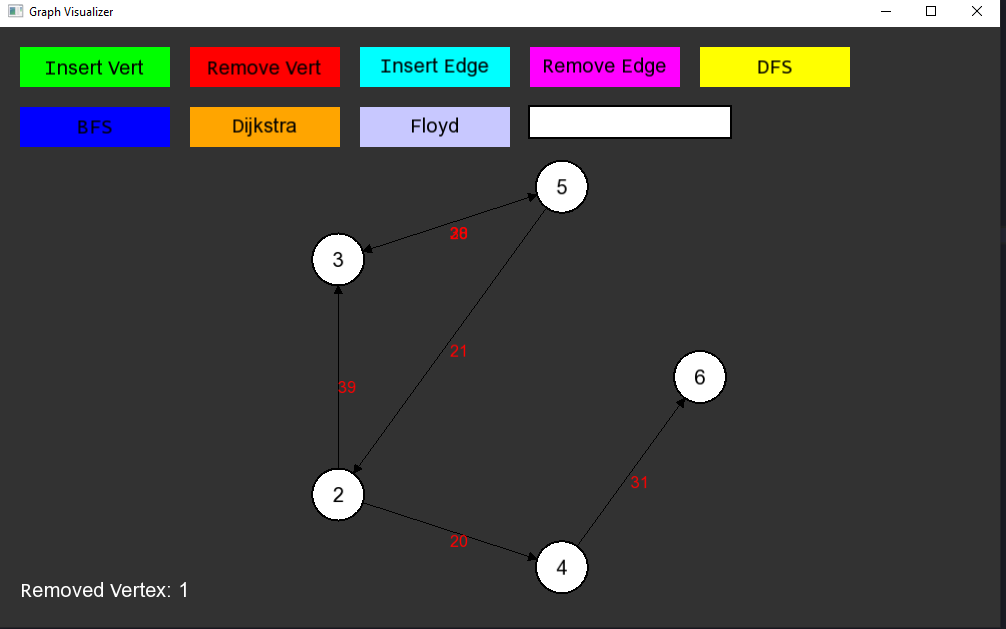
UML диаграмма:

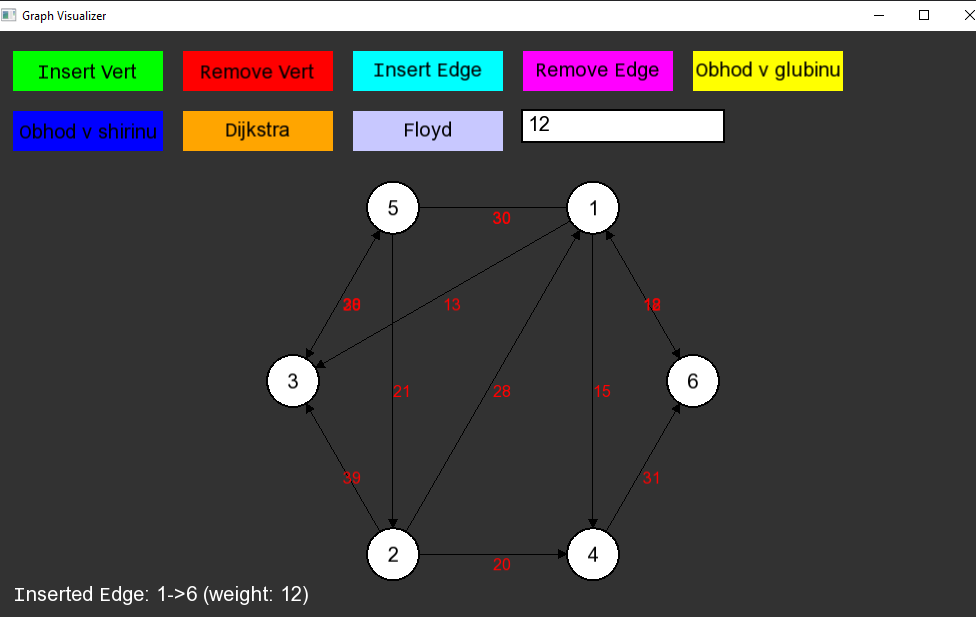


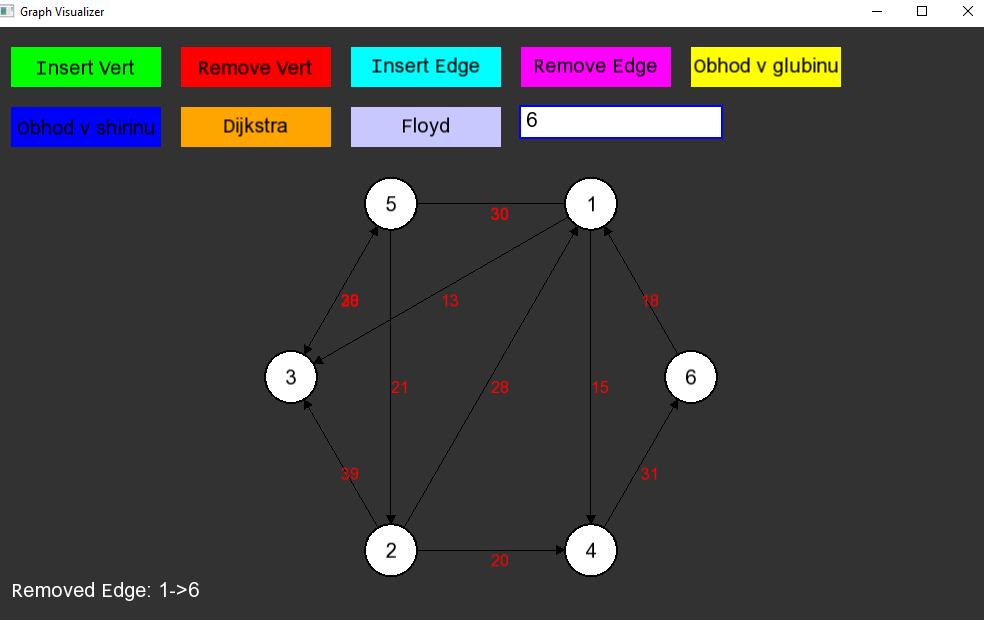
Работа программы:

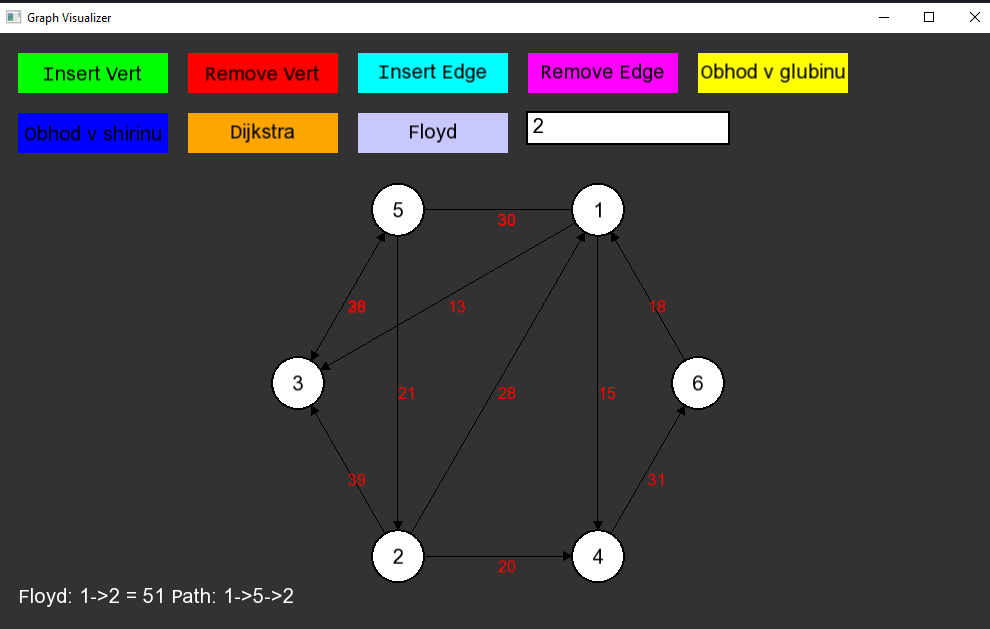


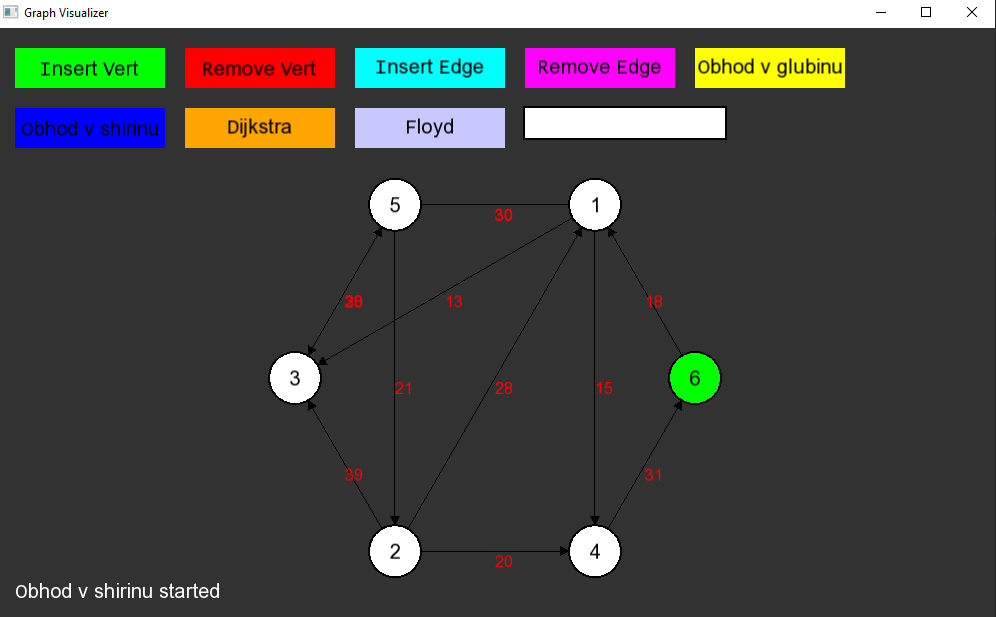


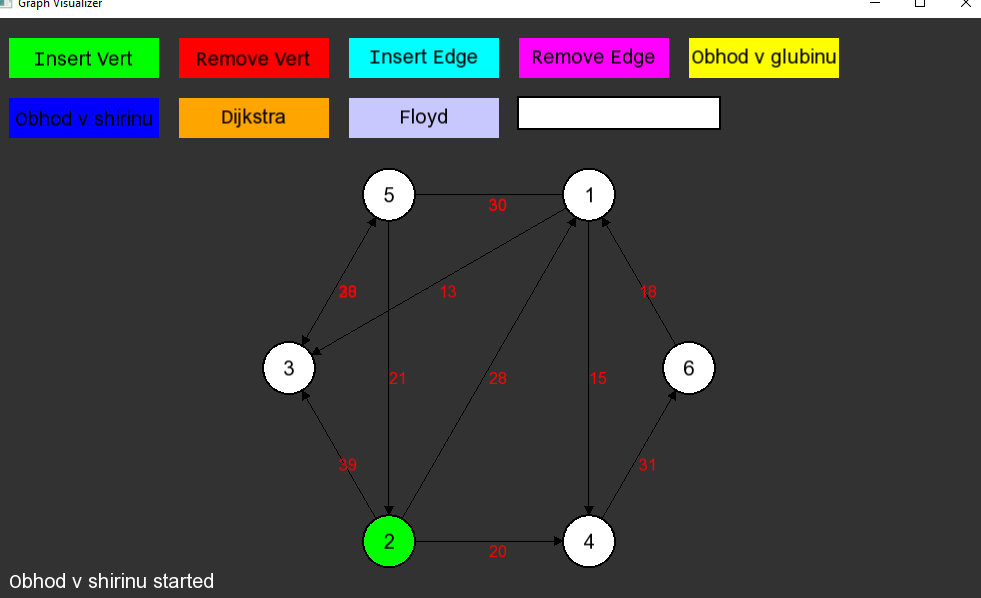


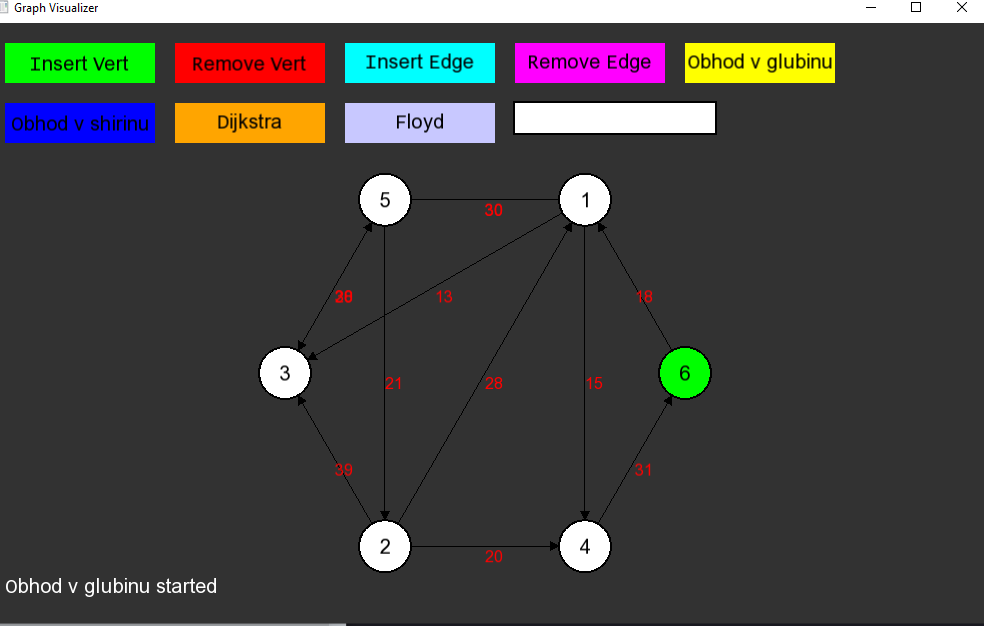


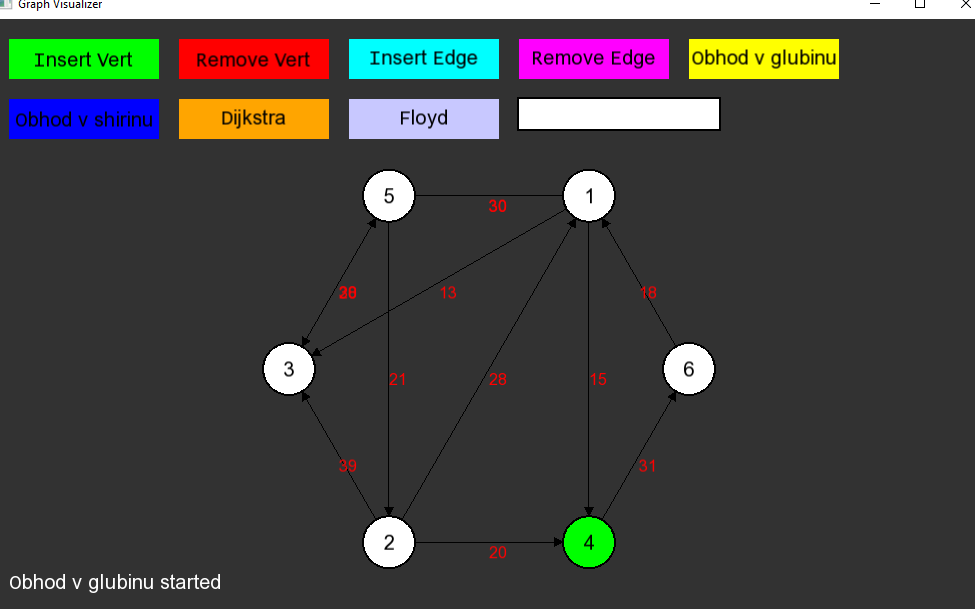












https://github.com/Prefix008/lab