Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ПНИПУ)

УТВЕРЖДАЮ  
Доцент кафедры ИТАС  
\_\_\_\_\_\_\_\_\_\_\_\_\_ О. А. Полякова  
"\_\_" \_\_\_\_\_\_\_\_\_\_ 2025 г.

ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Работа с графами

по теме:  
Задача коммивояжёра

(промежуточный)

Исполнитель:  
Студент РИС-24-3б \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Караваев

Пермь, 2025

**ПОСТАНОВКА ЗАДАЧИ**

Рисунок 1

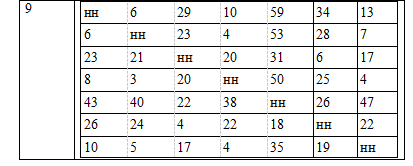


Рисунок 1 - матрица смежности

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

Реализовать задачу коммивояжёра

Создать интерактивный интерфейс (на выбор) : Windows Forms / Qt / OpenGL и т.д

Визуализация графа: добавление/удаление ребер, вершин.

Критерии графа:

Не менее 6 вершин

Двунаправленный

**ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

Реализации ключевых классов/функций

1. Алгоритм коммивояжера (salesman):
2. Редукция матрицы весов.
3. Поиск максимальных оценок нулей.
4. Проверка на образование циклов (createsSmallCycle).
5. Визуализация рёбер:

VertexArray line(Lines, 2);

line[0].position = start;

line[1].position = end;

lines.push\_back(line);

1. Обработка ввода:

if (event.text.unicode == '\b') { /\* Backspace \*/ }

else if (isdigit(event.text.unicode)) { /\* Цифры \*/ }

Использованные инструменты и технологии:

1. Язык и библиотеки C++17, SFML 2.5.1, STL
2. Среда разработки IDE: Visual Studio 2022, Сборка:встроенный компилятор VS
3. Вспомогательные программы: draw.io для диаграмм, Git для контроля версий

Достижения автора:

Изучил как работать в SFML.

Оптимизация алгоритма:

Реализация метода ветвей и границ для задачи коммивояжера.

Визуализация промежуточных шагов.

Интерактивность:

Динамическое обновление графа при добавлении/удалении элементов.

Введение

Актуальность работы

Современные информационные системы требуют эффективных методов обработки и визуализации графовых структур, которые широко применяются в логистике, сетевом анализе, биоинформатике и других областях. Разработка программного обеспечения для анализа графов с интерактивным интерфейсом позволяет не только решать теоретические задачи, но и наглядно демонстрировать работу алгоритмов, что особенно важно в образовательном процессе.

Цель работы

Создание программы на языке C++ для визуализации и анализа взвешенных ориентированных графов с реализацией алгоритма коммивояжера методом ветвей и границ, включая интерактивное управление структурой графа.

Задачи исследования

Разработать класс для хранения и обработки графа (матрица смежности, операции добавления/удаления вершин и рёбер).

Реализовать алгоритм поиска оптимального маршрута коммивояжёра.

Создать графический интерфейс

Визуализация графа (вершины, рёбра, веса)

Интерактивное управления (кнопки, поля ввода)

Отображение результатов работы алгоритмов

Провести тестирование корректности работы программы на различных входных данных.

**ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ**

Код

#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:

vector<int> vertList;

int\*\* matrix;

int\*\* matrix1;

int h1, h2;

vector<pair<int, int>> roads;

Graph(int s) :maxSize(s) {

matrix = new int\* [maxSize];

matrix1 = new int\* [maxSize];

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

matrix1[i] = new int[maxSize]();

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

}

vertList = { 0,1,2,3,4,5,6};

int tempmatrix[7][7] = {

{10000, 6, 29, 10, 59, 34,13},

{6, 10000, 23, 4, 53, 28,7},

{23, 21, 10000, 20, 31, 6,17},

{8, 3, 20, 10000, 50, 25,4},

{43, 40, 22, 38, 10000, 26,47},

{26, 24, 4, 22, 18, 10000,22},

{10, 5, 17, 4, 35, 19, 10000}

};

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[] matrix1[i];

}

delete[] matrix;

delete[] matrix1;

}

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;

}

}

bool g() {

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

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

if (matrix1[i][j] < 10000) return true;

}

}

return false;

}

bool createsSmallCycle(int from, int to, vector<pair<int,int>> path) {

map<int, int> edge\_map;

map<int, int> edge\_map1;

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

edge\_map[path[i].first] = path[i].second;

edge\_map1[path[i].second] = path[i].first;

}

int current = to;

int count = 0;

while (edge\_map.count(current)) {

current = edge\_map[current];

count++;

if (current == from) {

return count+1<maxSize;

}

if (count >= path.size()) break;

}

current = from;

count = 0;

while (edge\_map1.count(current)) {

current = edge\_map1[current];

count++;

if (current == from) {

return count+1<maxSize;

}

if (count >= path.size()) break;

}

return false;

}

void salesman() {

roads.clear();

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

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

matrix1[i][j] = matrix[i][j];

}

}

while (g()) {

vector<int> minListRow;

vector<int> minListColumn;

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

int RowM = 10000;

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

if (matrix1[i][j] < RowM) {

RowM = matrix1[i][j];

}

}

minListRow.push\_back(RowM);

}

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

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

if (minListRow[i] != 10000&& matrix1[i][j]!=10000)matrix1[i][j] =matrix1[i][j] - minListRow[i];

}

}

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

int ColumnM = 10000;

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

if (matrix1[j][i] < ColumnM) {

ColumnM = matrix1[j][i];

}

}

minListColumn.push\_back(ColumnM);

}

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

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

if(minListColumn[i] != 10000&&matrix1[j][i] != 10000) matrix1[j][i] =matrix1[j][i] - minListColumn[i];

}

}

int max = -1;

int maxi = -1;

int maxj = -1;

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

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

if (matrix1[i][j] == 0) {

int RowM = 10000;

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

if (g != j && matrix1[i][g] < RowM) {

RowM = matrix1[i][g];

}

}

int ColumnM = 10000;

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

if (g != i && matrix1[g][j] < ColumnM) {

ColumnM = matrix1[g][j];

}

}

int sum = RowM + ColumnM;

if (sum > max) {

max = sum;

maxi = i;

maxj = j;

}

}

}

}

if (!createsSmallCycle(maxi, maxj, roads)) {

cout << "go"<<endl;

roads.push\_back(make\_pair(maxi, maxj));

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

matrix1[i][maxj] = 10000;

matrix1[maxi][i] = 10000;

}

}

else {

cout << "goi" << endl;

matrix1[maxi][maxj] = 10000;

}

}

for (auto x : roads) {

cout << x.first << " " << x.second << endl;

}

}

string printSalesman(int ver) {

int sum = 0;

string str;

str += to\_string(ver) + "->";

int copyver=ver;

bool flag = true;

for (int i = 0; i < roads.size()&&flag; i++) {

if (roads[i].first == ver) {

if (roads[i].second == copyver) flag = false;

str += to\_string(roads[i].second) + "->";

sum += matrix[roads[i].first][roads[i].second];

ver = roads[i].second;

i = -1;

}

}

str += "sum=" + to\_string(sum);

return str;

}

};

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;

bool salesmanpend;

int val1;

int val2;

vector<int> out;

vector<ConvexShape> arrowheads;

vector<Text> edgeWeights;

public:

GraphVisualizer(Graph& gra, RenderWindow& w) :

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

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

if (!font.loadFromFile("C:\\Users\\Artem Loginov\\Desktop\\laboratorki\\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, "salesman", Color::Yellow);

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;

}

}

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

int value = stoi(inputString);

graph.salesman();

string a = graph.printSalesman(value);

infoStream << "Salesman: " << a;

}

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 mid = (nodes[j].getPosition() + Vector2f(25, 25) + nodes[i].getPosition() + Vector2f(25, 25)) \* 0.5f;

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);

ConvexShape arrow(3);

arrow.setPoint(0, mid);

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

arrow.setPoint(2, mid - 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(10);

weightText.setFillColor(Color::Red);

weightText.setPosition((start + mid) / 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(7);

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;

}

**РЕЗУЛЬТАТЫ**

Было реализованна задача в SFML. Был создан интерфейс и работающие кнопки (Приложение А), была сделана UML-диаграммы (Приложение Б).

**ПРИЛОЖЕНИЕ А**

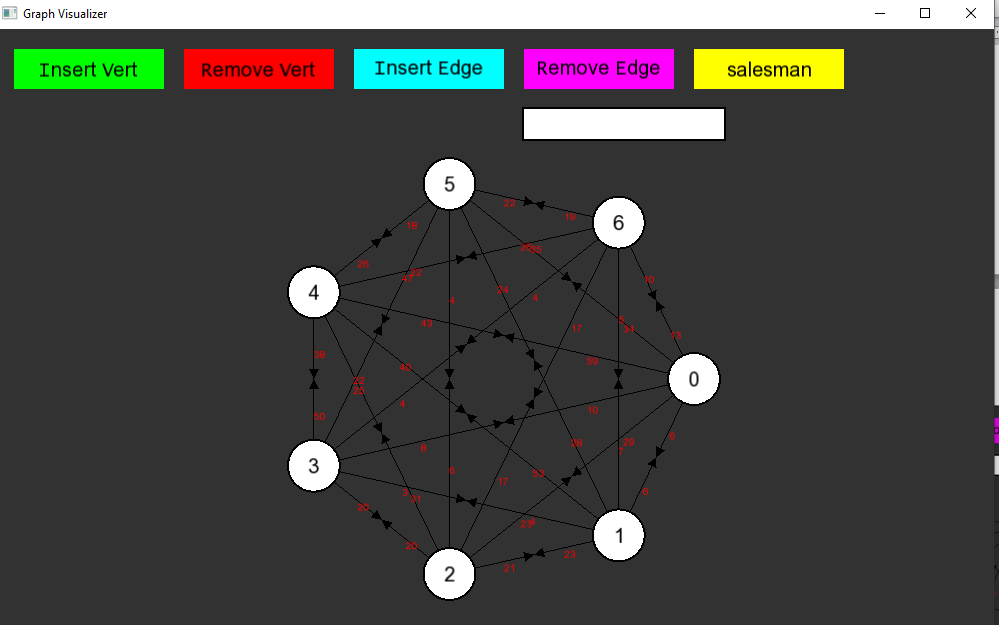


Рисунок 2 - общий вид

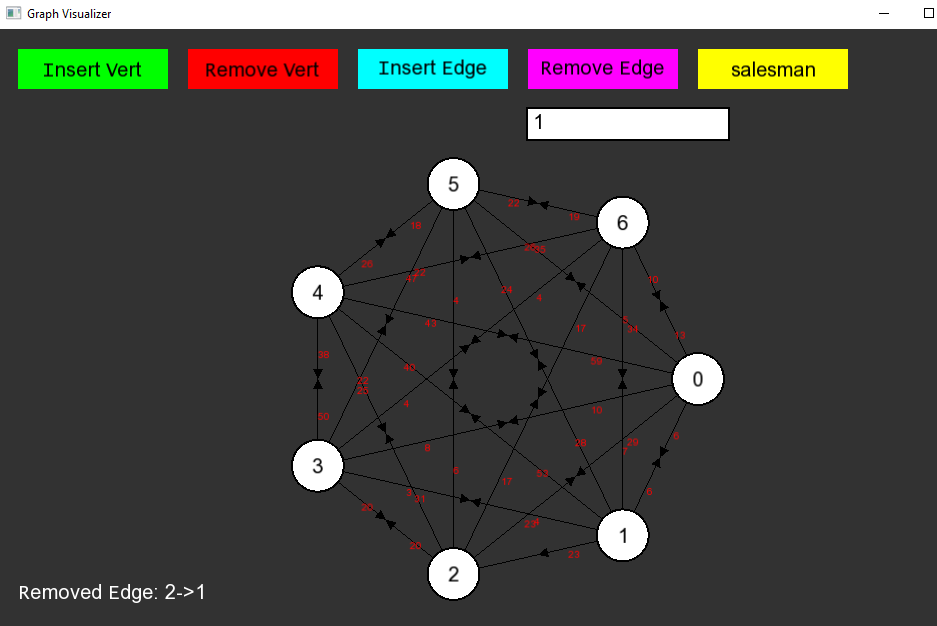


Рисунок 3 - удаление ребра

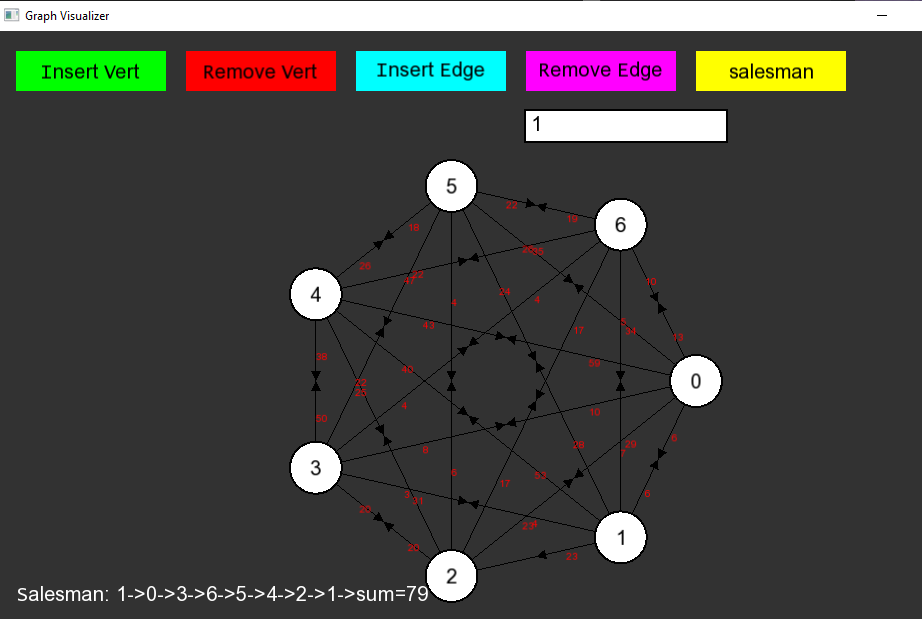


Рисунок 4 - задача коммивояжёра

**ПРИЛОЖЕНИЕ Б**

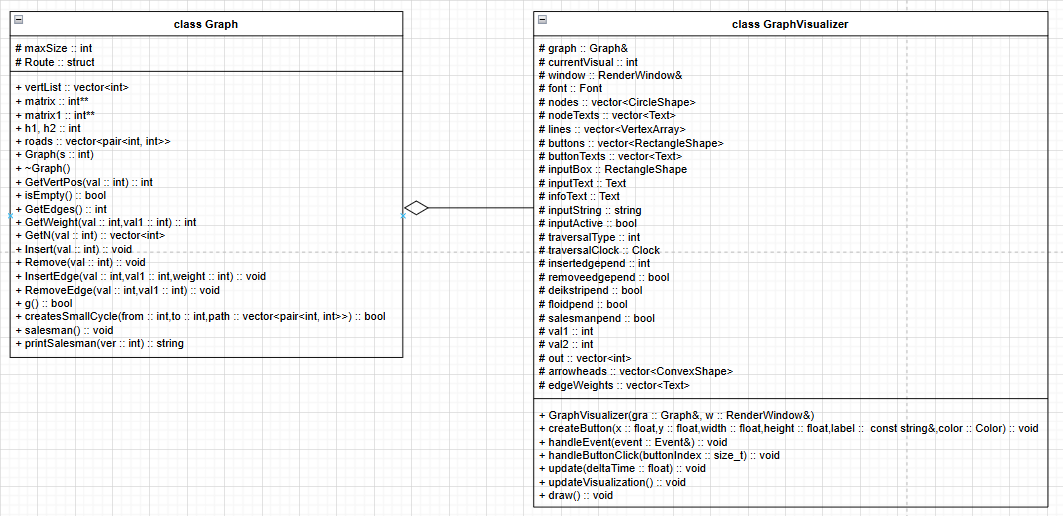


Рисунок 5 - UML-диаграмма