Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**Отчет по Творческому заданию**

Задача Коммивояжёра, разработка калькулятора

Выполнил работу

студент группы

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2022

**Задача Коммивояжёра**

Реализовать поиск кратчайшего пути графа

При работе были использованы

-для работы с визуализацией OpenGL

-метод ветвей и границ

**Код**

#include "open.h"

using namespace std;

int WinW = 1000;

int WinH = 600;

bool\* Mouse\_Vert = new bool[maxSize];

bool standView = false;

bool Vert\_Move = false;

vector<pair<int, int>> Path;

vector<int> NewPath;

int R;

int Moving\_Vertex;

int CursorX;

int CursorY;

vertCoord vertC[maxSize + 1];

Graph graph;

int Button\_Flag;

Graph::Graph()

{

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

{

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

{

adjMatrix[i][j] = 0;

}

Mouse\_Vert[i] = false;

}

}

Graph::~Graph()

{ }

int Graph::GetVertPos(const int& vertex)

{

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

{

if (vertList[i] == vertex)

return i;

}

return -1;

}

bool Graph::IsEmpty()

{

if (vertList.size() != 0)

return false;

else

return true;

}

bool Graph::IsFull()

{

return (vertList.size() == maxSize);

}

void Graph::InsertVertex(const int& vertex)

{

if (!IsFull())

vertList.push\_back(vertex);

else

{

cout << "ERROR! Graf is full! " << endl;

return;

}

}

void Graph::InsertEdge(const int& vertex1, const int& vertex2, int weight)

{

if (weight < 1)

{

cout << "\nIncorrectly entered value\n";

return;

}

if (GetVertPos(vertex1) != (-1) && GetVertPos(vertex2) != (-1))

{

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

if (adjMatrix[vertPos1][vertPos2] != 0 && adjMatrix[vertPos2][vertPos1] != 0)

{

cout << "ERROR! Path length alredy exists" << endl;

return;

}

else

{

adjMatrix[vertPos1][vertPos2] = weight;

}

}

else

{

cout << "ERROR! There are no such nodes " << endl;

return;

}

}

void Graph::Print()

{

if (!IsEmpty())

{

cout << "Matrix: " << endl;

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

{

cout << vertList[i] << " ";

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

cout << setw(4) << adjMatrix[i][j];

cout << endl;

}

}

else

cout << "Error! Graph is empty" << endl;

}

void Graph::eraseLastVertex()

{

if (IsEmpty())

{

cout << "\nGraph empty\n";

return;

}

int n = vertList.size();

for (int j = 0; j < n; j++)

{

adjMatrix[n - 1][j] = 0;

adjMatrix[j][n - 1] = 0;

}

vertList.pop\_back();

}

void Graph::eraseEdge(const int& vertex1, const int& vertex2)

{

if (GetVertPos(vertex1) != (-1) && GetVertPos(vertex2) != (-1))

{

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

if (adjMatrix[vertPos1][vertPos2] == 0 && adjMatrix[vertPos2][vertPos1] == 0)

{

cout << "ERROR!" << endl;

return;

}

else

{

adjMatrix[vertPos1][vertPos2] = 0;

}

}

else

{

cout << "ERROR! Graph does not have one or these vertices " << endl;

return;

}

}

int Graph::GetAmountEdges()

{

int edges = 0;

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

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

if (adjMatrix[i][j] > 0)

edges++;

return edges;

}

void Graph::drawGraph()

{

int n = graph.GetAmountVerts();

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

{

if (!standView)

setCoords(i, n);

}

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

{

for (int j = 0; j < n; j++)

{

int a = adjMatrix[i][j];

if (a != 0)

drawLine(a, vertC[i].x, vertC[i].y, vertC[j].x, vertC[j].y);

if (a == adjMatrix[j][i] && a != 0)

drawLine(a, vertC[j].x, vertC[j].y, vertC[i].x, vertC[i].y);

}

}

drawVertex(n);

glutPostRedisplay();

}

int\*\* Make\_Matrix()

{

int n = graph.GetAmountVerts();

int\*\* matrix = new int\* [n];

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

matrix[i] = new int[n];

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

{

for (int j = 0; j < n; j++)

{

int elem = graph.getAdjMatrixElem(i, j);

if (elem == 0 or i == j)

matrix[i][j] = -1;

else

matrix[i][j] = elem;

}

}

cout << "Initial matrix: \n";

Print\_Matrix(matrix);

return matrix;

}

int\* Find\_Min(int\* line, int n)

{

int min = 1000000;

for (int j = 0; j < n; j++)

if (line[j] >= 0 && line[j] < min)

min = line[j];

for (int j = 0; j < n; j++)

if (line[j] >= 0)

line[j] -= min;

return line;

}

void Print\_Matrix(int\*\* matrix)

{

int n = graph.GetAmountVerts();

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

{

for (int j = 0; j < n; j++)

cout << setw(4) << matrix[i][j];

cout << endl;

}

}

int\*\* Reduct\_Matrix(int\*\* oldmatrix)

{

int\*\* matrix = oldmatrix;

int n = graph.GetAmountVerts();

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

matrix[i] = Find\_Min(matrix[i], n);

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

{

int min = 1000000;

for (int j = 0; j < n; j++)

{

if (matrix[j][i] >= 0 && matrix[j][i] < min)

min = matrix[j][i];

}

for (int j = 0; j < n; j++)

{

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

matrix[j][i] -= min;

}

}

cout << "\nReduction matrix: \n";

Print\_Matrix(matrix);

return matrix;

}

int\*\* Rebuild\_Matrix(int\*\* oldmatrix)

{

int n = graph.GetAmountVerts();

int\*\* matrix = Reduct\_Matrix(oldmatrix);

int max = -1;

int line, column;

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

{

for (int j = 0; j < n; j++)

{

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

{

int minLine = 1000000;

int minColumn = 1000000;

for (int k = 0; k < n; k++)

{

if (matrix[i][k] != -1 && k != j && matrix[i][k] < minLine)

minLine = matrix[i][k];

}

for (int k = 0; k < n; k++)

{

if (matrix[k][j] != -1 && k != i && matrix[k][j] < minColumn)

minColumn = matrix[k][j];

}

if (max < minColumn + minLine)

{

max = minColumn + minLine;

line = i;

column = j;

}

}

}

}

pair<int, int> p;

p.first = line + 1;

p.second = column + 1;

Path.push\_back(p);

matrix[line][column] = -1;

matrix[column][line] = -1;

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

{

matrix[line][i] = -1;

matrix[i][column] = -1;

}

cout << endl;

cout << "Delete zero: \n";

Print\_Matrix(matrix);

cout << "\nSegments of the way: ";

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

cout << Path[i].first << " -> " << Path[i].second << " ";

cout << endl;

return matrix;

}

void Print\_Result()

{

int second = Path[0].second;

int i = 2;

NewPath.push\_back(Path[0].first);

NewPath.push\_back(Path[0].second);

while (i != graph.GetAmountVerts() + 1)

for (int j = 1; j < graph.GetAmountVerts(); j++)

if (Path[j].first == second)

{

second = Path[j].second;

NewPath.push\_back(second);

i++;

}

cout << "Result: ";

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

{

cout << NewPath[i];

if (i != NewPath.size() - 1)

cout << " -> ";

}

int sum = 0;

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

{

int line = Path[i].first - 1;

int column = Path[i].second - 1;

sum += graph.getAdjMatrixElem(line, column);

}

cout << "\nS = " << sum << endl;;

}

//Кнопка найти решение

void drawButton1()

{

if (Button\_Flag == 2)

glColor3f(0.3, 0.4, 0.5);

else

glColor3f(1.0, 0.6, 0.3);

glBegin(GL\_QUADS);

glVertex2i(50, WinH - WinH / 7 - 20);

glVertex2i(50, WinH - 2 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 2 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - WinH / 7 - 20);

glEnd();

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

glBegin(GL\_LINE\_LOOP);

glVertex2i(50, WinH - WinH / 7 - 20);

glVertex2i(50, WinH - 2 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 2 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - WinH / 7 - 20);

glEnd();

string name = "FIND SOLUTION";

glRasterPos2i(WinW / 16-5, 0.77 \* WinH);

for (int i = 0; i < name.length(); i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10, name[i]);

}

//вывод матрицы

void drawButton2()

{

if (Button\_Flag == 3)

glColor3f(0.3, 0.4, 0.5);

else

glColor3f(1.0, 0.6, 0.3);

glBegin(GL\_QUADS);

glVertex2i(50, WinH - 2 \* (WinH / 7) - 20);

glVertex2i(50, WinH - 3 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 3 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 2 \* (WinH / 7) - 20);

glEnd();

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

glBegin(GL\_LINE\_LOOP);

glVertex2i(50, WinH - 2 \* (WinH / 7) - 20);

glVertex2i(50, WinH - 3 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 3 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 2 \* (WinH / 7) - 20);

glEnd();

string name = "PRINT MATRIX";

glRasterPos2i(WinW / 16.5, 0.63 \* WinH);

for (int i = 0; i < name.length(); i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10, name[i]);

}

//кнопка добавить путь

void drawButton3()

{

if (Button\_Flag == 4)

glColor3f(0.3, 0.4, 0.5);

else

glColor3f(1.0, 0.6, 0.3);

glBegin(GL\_QUADS);

glVertex2i(50, WinH - 3 \* (WinH / 7) - 20);

glVertex2i(50, WinH - 4 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 4 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 3 \* (WinH / 7) - 20);

glEnd();

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

glBegin(GL\_LINE\_LOOP);

glVertex2i(50, WinH - 3 \* (WinH / 7) - 20);

glVertex2i(50, WinH - 4 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 4 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 3 \* (WinH / 7) - 20);

glEnd();

string name = "ADD";

glRasterPos2i(WinW / 16+25, 0.48 \* WinH);

for (int i = 0; i < name.length(); i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10, name[i]);

}

//кнопка удалить путь

void drawButton4()

{

if (Button\_Flag == 5)

glColor3f(0.3, 0.4, 0.5);

else

glColor3f(1.0, 0.6, 0.3);

glBegin(GL\_QUADS);

glVertex2i(50, WinH - 4 \* (WinH / 7) - 20);

glVertex2i(50, WinH - 5 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 5 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 4 \* (WinH / 7) - 20);

glEnd();

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

glBegin(GL\_LINE\_LOOP);

glVertex2i(50, WinH - 4 \* (WinH / 7) - 20);

glVertex2i(50, WinH - 5 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 5 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 4 \* (WinH / 7) - 20);

glEnd();

string name = " DELETE ";

glRasterPos2i(WinW / 17+6, 0.34 \* WinH);

for (int i = 0; i < name.length(); i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10, name[i]);

}

//отрисовка круга

void drawCircle(int x, int y, int R)

{

glColor3f(0.9f, 0.9f, 0.3f);

float x1, y1;

glBegin(GL\_POLYGON);

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

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y1 = R \* cos(theta) + y;

x1 = R \* sin(theta) + x;;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(0.2f, 0.2f, 0.3f);

float x2, y2;

glBegin(GL\_LINE\_LOOP);

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

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y2 = R \* cos(theta) + y;

x2 = R \* sin(theta) + x;

glVertex2f(x2, y2);

}

glEnd();

}

void drawBorderedCircle(int x, int y, int R)

{

glColor3f(0.3, 0.4, 0.5);

float x1, y1;

glBegin(GL\_POLYGON);

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

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y1 = R \* cos(theta) + y;

x1 = R \* sin(theta) + x;;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(0.0, 0.392, 0.0);

float x2, y2;

glBegin(GL\_LINE\_LOOP);

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

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y2 = R \* cos(theta) + y;

x2 = R \* sin(theta) + x;

glVertex2f(x2, y2);

}

glEnd();

}

//работа с текстом

void drawText(int text, int x1, int y1)

{

glColor3f(0.0, 0.0, 0.0);

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = to\_string(text);

glRasterPos2i(x1 - 5, y1 - 5);

for (size\_t j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

//отрисовка линии

void drawLine(int text, int x0, int y0, int x1, int y1)

{

glColor3i(0, 0, 0);

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

drawText(text, (x0 + x1) / 2 + 10, (y0 + y1) / 2 + 10);

float vx = x0 - x1;

float vy = y0 - y1;

float s = 1.5f / sqrt(vx \* vx + vy \* vy);

vx \*= s;

vy \*= s;

x1 = x1 + R \* vx;

y1 = y1 + R \* vy;

glColor3i(0, 0, 0);

glBegin(GL\_TRIANGLES);

glVertex2f(x1, y1);

glVertex2f(x1 + 10 \* (vx + vy), y1 + 10 \* (vy - vx));

glVertex2f(x1 + 10 \* (vx - vy), y1 + 10 \* (vy + vx));

glEnd();

}

//Текст с кружочками

void drawVertex(int n)

{

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

{

if (Mouse\_Vert[i])

drawBorderedCircle(vertC[i].x, vertC[i].y, R);

else

drawCircle(vertC[i].x, vertC[i].y, R);

drawText(i + 1, vertC[i].x, vertC[i].y);

}

}

//Работа с кругами

void setCoords(int i, int n)

{

int R\_;

int x0 = WinW / 2;

int y0 = WinH / 2;

if (WinW > WinH)

{

R = 3 \* (WinH / 13) / n;

R\_ = WinH / 2 - R - 10;

}

else

{

R = 3 \* (WinW / 13) / n;

R\_ = WinW / 2 - R - 10;

}

float theta = 2.0f \* 3.1415926f \* i / n;

int y1 = R\_ \* cos(theta) + y0;

int x1 = R\_ \* sin(theta) + x0;

vertC[i].x = x1;

vertC[i].y = y1;

}

//создание графа

void makeGraph()

{

standView = false;

int amountVerts, amountEdges, sourceVertex, targetVertex, edgeWeight;

cout << "How many nodes are in the graph: "; cin >> amountVerts;

cout << "How many paths: "; cin >> amountEdges;

for (int i = 1; i <= amountVerts; i++) {

graph.InsertVertex(i);

}

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

{

cout << "Start node: "; cin >> sourceVertex;

cout << "Finish node: "; cin >> targetVertex;

cout << "Way: "; cin >> edgeWeight;

graph.InsertEdge(sourceVertex, targetVertex, edgeWeight);

}

cout << endl;

graph.Print();

}

bool Salesman\_Check(int\*\* matrix)

{

if (graph.IsEmpty())

return false;

for (int i = 0; i < graph.GetAmountVerts(); i++)

{

int cnt = 0;

for (int j = 0; j < graph.GetAmountVerts(); j++)

{

if (matrix[i][j] > 0)

cnt++;

}

if (cnt < 1)

return false;

}

return true;

}

//перемещение кругов

int Circle\_Check(int x, int y)

{

for (int i = 0; i < graph.GetAmountVerts(); i++)

if (pow(x - vertC[i].x, 2) + pow(y - vertC[i].y, 2) <= pow(R, 2))

return i;

return -1;

}

//нажатие кнопок

void Button\_Check(int x, int y)

{

if (x > 50 && x < WinW / 7 && y < (WinH - 20) && y >(WinH - WinH / 7))

Button\_Flag = 1;

else if (x > 50 && x < WinW / 7 && y < (WinH - WinH / 7 - 20) && y > WinH - 2 \* (WinH / 7))

Button\_Flag = 2;

else if (x > 50 && x < WinW / 7 && y < WinH - 2 \* (WinH / 7) - 20 && y > WinH - 3 \* (WinH / 7))

Button\_Flag = 3;

else if (x > 50 && x < WinW / 7 && y > WinH - 4 \* (WinH / 7) && y < WinH - 3 \* (WinH / 7) - 20)

Button\_Flag = 4;

else if (x > 50 && x < WinW / 7 && y > WinH - 5 \* (WinH / 7) && y < WinH - 4 \* (WinH / 7) - 20)

Button\_Flag = 5;

else

Button\_Flag = 0;

}

void mouseMove(int x, int y)

{

y = WinH - y;

CursorX = x;

CursorY = y;

int i = Circle\_Check(x, y);

if (i != -1)

Mouse\_Vert[i] = true;

else

for (int j = 0; j < graph.GetAmountVerts(); j++)

Mouse\_Vert[j] = false;

if (Vert\_Move)

{

vertC[Moving\_Vertex].x = CursorX;

vertC[Moving\_Vertex].y = CursorY;

}

Button\_Check(x, y);

glutPostRedisplay();

}

void mouseClick(int button, int state, int x, int y)

{

int j = Circle\_Check(x, WinH - y);

if (Vert\_Move)

{

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

{

Vert\_Move = false;

return;

}

}

if (j != -1)

{

standView = true;

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

{

Vert\_Move = true;

Moving\_Vertex = j;

return;

}

}

if (x >= 50 and x <= (WinW / 7) and y >= ((WinH / 7) + 20) and y <= 2 \* (WinH / 7))

{

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

{

NewPath.clear();

Path.clear();

cout << "\nSolution:\n";

int\*\* matrix = Make\_Matrix();

bool checker = Salesman\_Check(matrix);

if (!checker)

{

cout << "\n Solutions don't exist\n\n";

return;

}

int n = graph.GetAmountVerts();

while (Path.size() < n)

matrix = Rebuild\_Matrix(matrix);

cout << endl;

Print\_Result();

return;

}

}

if (x >= 50 and x <= (WinW / 7) and y >= (2 \* (WinH / 7) + 20) and y <= 3 \* (WinH / 7))

{

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

{

graph.Print();

return;

}

}

if (x >= 50 && x <= WinW / 7 && y <= 4 \* (WinH / 7) && y >= 3 \* (WinH / 7) + 20)

{

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

{

int sourceVertex, targetVertex, edgeWeight, vertNum;

cout << "How much to add: "; cin >> vertNum;

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

{

cout << "Start node: "; cin >> sourceVertex;

cout << "Finish node: "; cin >> targetVertex;

cout << "Way: "; cin >> edgeWeight;

graph.InsertEdge(sourceVertex, targetVertex, edgeWeight);

}

return;

}

}

if (x > 50 && x < WinW / 7 && y <= 5 \* (WinH / 7) && y >= 4 \* (WinH / 7) + 20)

{

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

{

int sourceVertex, targetVertex;

cout << "Start node: "; cin >> sourceVertex;

cout << "Finish node: "; cin >> targetVertex;

graph.eraseEdge(sourceVertex, targetVertex);

return;

}

}

}

void reshape(int w, int h)

{

WinW = w;

WinH = h;

glViewport(0, 0, (GLsizei)WinW, (GLsizei)WinH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, (GLdouble)WinW, 0, (GLdouble)WinH);

glutPostRedisplay();

}

void display()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClearColor(0.3, 0.9, 0.9, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

drawButton1();

drawButton2();

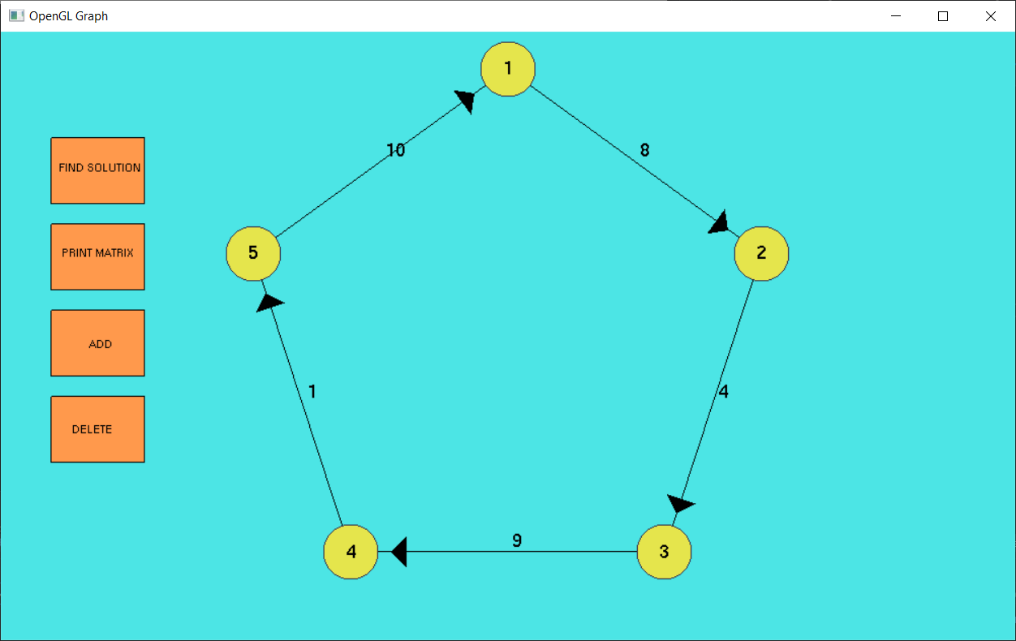
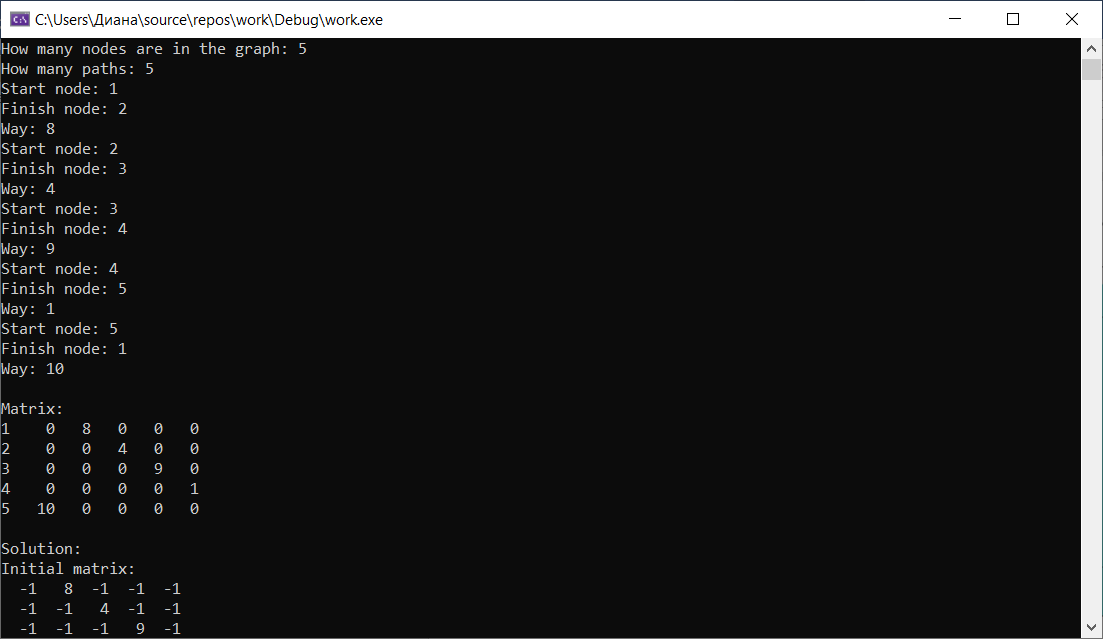
drawButton3();

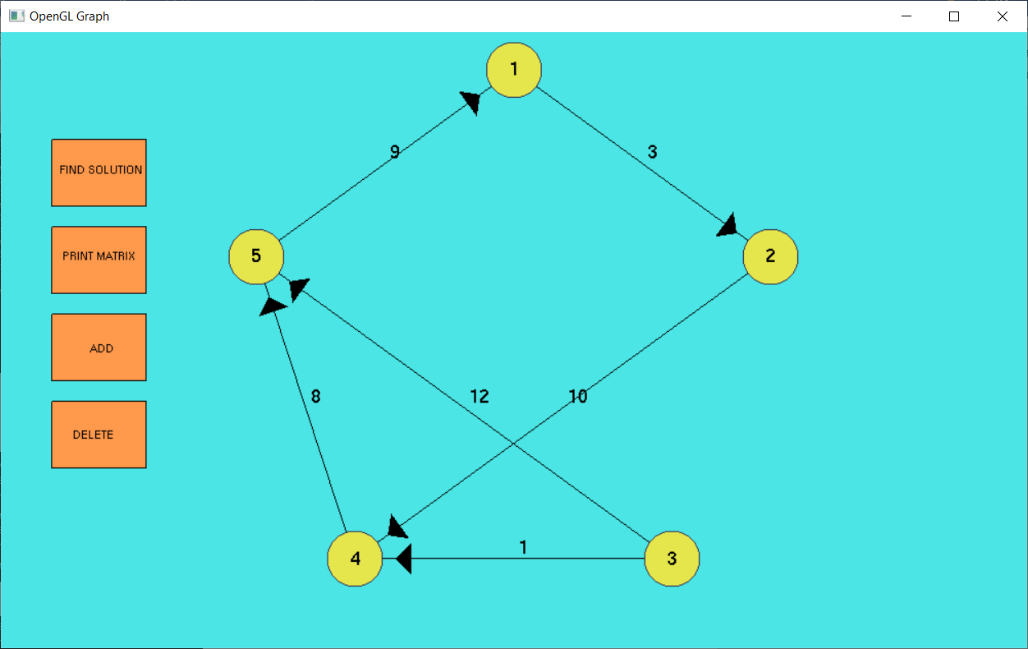
drawButton4();

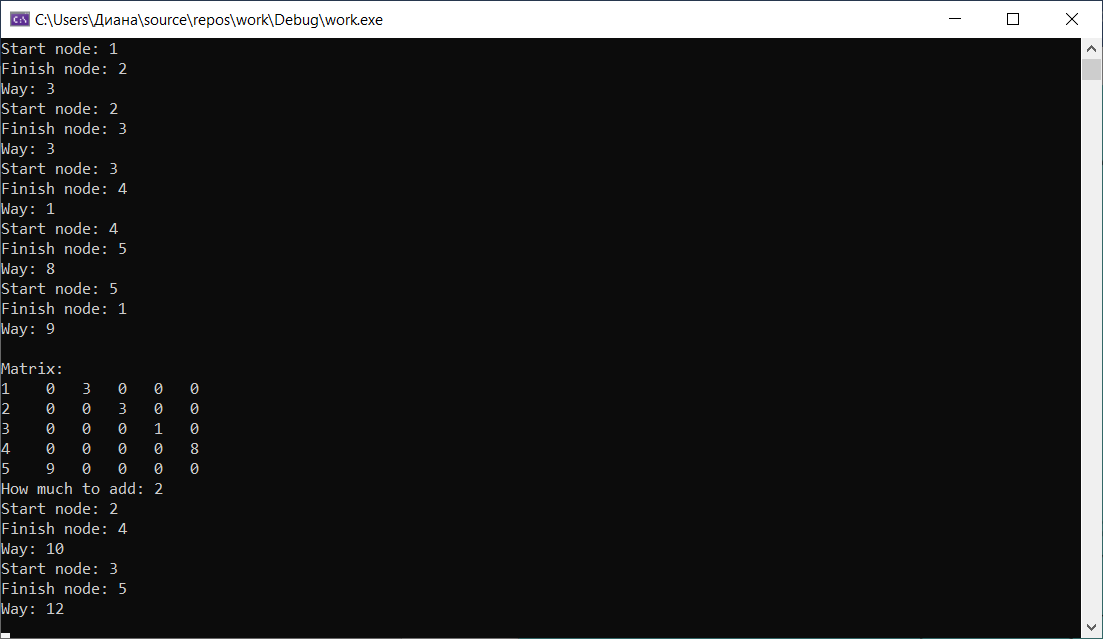
graph.drawGraph();

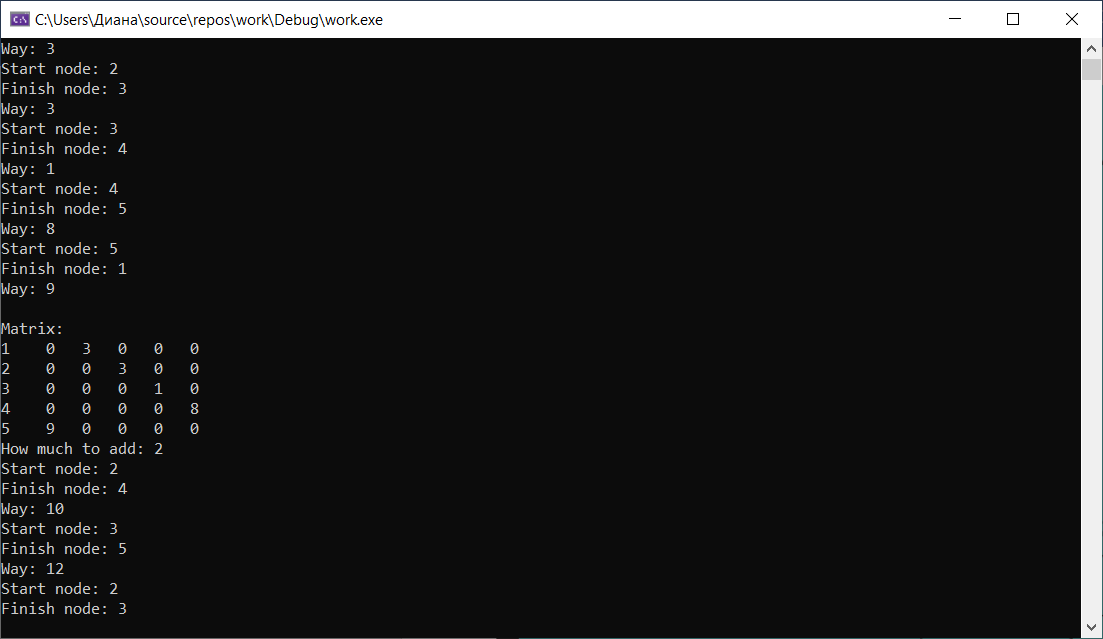
glutSwapBuffers();

}

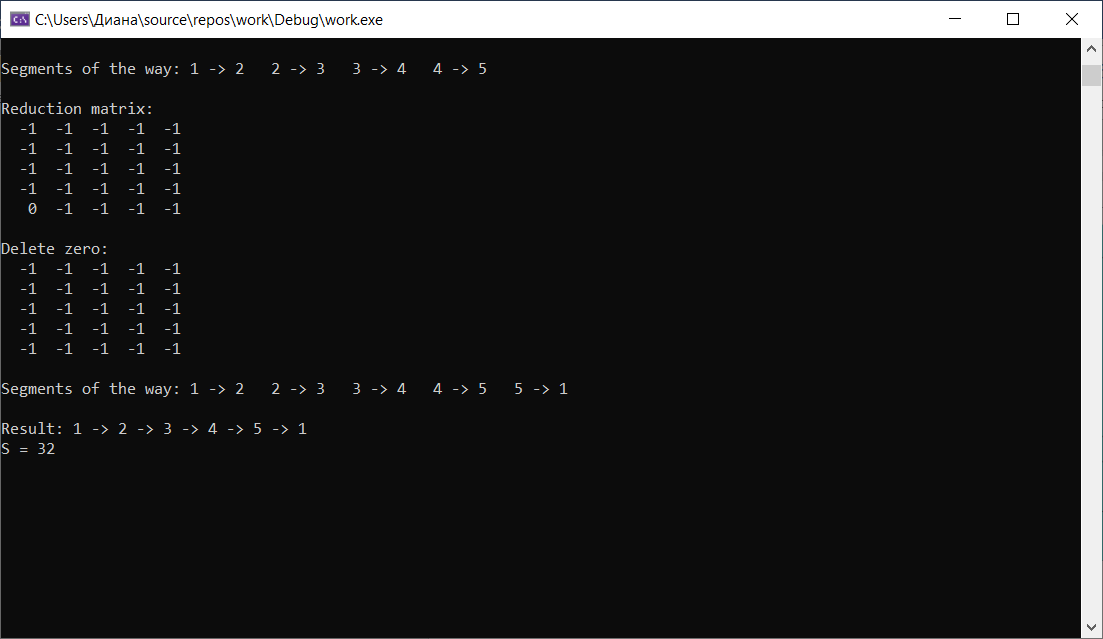




Добавление элементов:

Удаление элементов:

**Вывод ответа:**



**UML-диаграмма**

