Пермский Национальный Исследовательский Политехнический Университет

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Творческий проект

по дисциплине

«Основы алгоритмизации и программирования»

Тема: «Решение задачи Коммивояжера»

Вариант 6

Выполнил:

студент группы РИС-20-1б

Лысенок В.Р.

Проверила:

доцент кафедры ИТАС

Полякова О. А.

Пермь 2021.

# **Введение**

* 1. **Актуальность**

Задачи комбинаторной оптимизации встречаются на практике постоянно. Наиболее известной среди них является задача коммивояжера. Так, например, в функционировании современных предприятий самого разного рода существенное место занимают транспортные потоки: для каждой компании актуален вопрос о своевременной доставке товара потребителям в кратчайшие сроки. Решение задачи коммивояжера в данном вопросе помогает найти наиболее рациональный и выгодный путь, что в свою очередь играет огромную роль для экономической составляющей компании, а именно – в снижении затрат.

**1.2 Цель**

Разработать программу для решения задачи коммивояжера методом ветвей и границ и проработать визуализацию графа.

* 1. **Задачи**

1. Изучить теоретические материалы по платформе *OpenGL*.
2. Написать программу для решения задачи коммивояжера.
3. Дополнить программы дизайнерскими решениями и разработать собственный интерфейс.
4. Протестировать работу программ на различном наборе входных данных.

# **Задача коммивояжера. Метод ветвей и границ**

* 1. **Постановка задачи**

Решить задачу коммивояжера для графа, указанного в варианте 6. Разработать программу, которая будет универсальной на любом наборе исходных данных. Проработать визуализирующую часть в программе посредством *OpenGL*.

* 1. **Анализ задачи**

1. Создается класс *Graph.*  В классе определяется два *public* поля: вектор вершин и матрица смежности целочисленных типов. В *private* секции класса содержаться следующие селекторы и методы для работы с графом:

* GetVertText //получить значение вершины
* GetVertList //получить вектор вершин
* GetAdjMatrixElem //получить элемент матрицы смежности
* GetAmountVerts //получить количество вершин графа
* GetVertPos //получить позицию вершины в векторе вершин
* IsEmpty //проверка на пустоту
* IsFull //проверка на полноту
* InsertVertex //добавить вершину в конец вектора вершин
* InsertEdge //добавить ребро
* Print //вывод матрицы смежности в консоль
* EraseLastVert //удалить последнюю вершину в векторе вершин
* EraseEdge //удалить ребро
* DrawGraph //рисование графа посредством *OpenGL*

1. Дополнительно определяются функции необходимые для решения задачи Коммивояжера. Функция *Salesman\_Check* принимает матрицу смежности графа и определяет возможность решения задачи коммивояжера. Функция *Change\_Matrix* копирует матрицу смежности графа, заменяет в матрице 0 на -1 по причине того, что 0 используется в дальнейшем в решении задачи. В функции *Reduct\_Matrix* из элементов ряда матрицы вычитается минимальный элемент соответствующего ряда. Таким образом, происходит редукция строк и столбцов матрицы. В функции *High\_Zero* происходит поиск нулевого элемента с наибольшей оценкой, также сохраняется промежуточный отрезок пути. Элементы строки и столбца найденного элемента заполняются -1, элемент исключается из дальнейшего поиска. Функция *Print\_Result* выводит на экран найденный путь коммивояжера, поочередно все отрезки путей.
2. Рисование графа начинается с вызова метода *DrawGraph*, в котором вершинам устанавливаются координаты, вызываются функции рисования кругов, ребер и их веса, значений вершин.
3. В интерфейсе программы предусмотрены следующие семь кнопок: 1 – создание нового графа / перерисовка вершин текущего графа в круговом виде, 2 – решение задачи коммивояжера, 3 – вывод матрицы смежности графа на экран, 4 – добавление вершины, 5 – удаление последней вершины, 6 – добавление ребра, 7 – удаление ребра. Все кнопки рисуются по одинаковому принципу: прямоугольник с заданными координатами четырех вершин и замкнутая линия с такими же координатами – граница прямоугольника. Кнопки меняют цвет при наведении на них курсора мыши. Это реализуется с помощью переменных-флажков и функции *mouseMove*, которая отслеживает координаты курсора мыши. Если курсор в зоне кнопки, то есть удовлетворяет ее координатам, то флажку присваивается необходимое значение и кнопка рисуется одним цветом, иначе значение флажка остается неизменным и кнопка рисуется стандартным цветом. Такой же принцип работы реализован с вершинами графа, при наведении на них они также меняют цвет. В данном случае проверка на то, входит ли курсор в зону вершины, осуществляется с помощью использования уравнения окружности. С помощью функции *mouseMove* также реализовано перемещение вершин графа. При нажатии на вершину и последующего перемещения курсора координаты центра круга (вершины) инициализируются координатами курсора, таким образом, вершина движется. При втором нажатии на вершину движение прекращается, координаты круга фиксируются. Регистрация нажатий на кнопки реализована с помощью функции *mouseClick.* Если курсор в зоне кнопки и произведено нажатие, то вызываются необходимые методы в соответствии с функцией кнопки.
   1. **Код на языке С++**

1. Файл <Main.cpp>

#include "Graph.h"

#include <iostream>

using namespace std;

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

{

setlocale(LC\_ALL, "ru");

glutInit(&argc, argv);

makeGraph();

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(WinW, WinH);

glutCreateWindow("Graph");

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMouseFunc(mouseClick);

glutPassiveMotionFunc(mouseMove);

glutMainLoop();

}

2. Файл “Graph.h”

#pragma once

#include <glut.h>

#include <iostream>

#include <Windows.h>

#include <iomanip>

#include <vector>

#include <cmath>

#include <string>

using namespace std;

const int maxSize = 20; //максимальное число вершин

extern int WinW; //ширина окна (OpenGL)

extern int WinH; //высота окна (OpenGL)

struct vertCoord //структура координат вершин

{

int x, y;

};

class Graph

{

private:

vector<int> vertList; //вектор вершин

int adjMatrix[maxSize][maxSize]; //матрица смежности

public:

Graph(); //конструктор (заполняет матрицу смежности нулями)

~Graph(); //деструктор

bool IsEmpty(); //проверка на пустоту

bool IsFull(); //проверка на полноту

int GetVertText(int i) { return vertList[i]; } //получить значение вершины

vector<int> GetVertList() { return vertList; } //получить вектор вершин

int GetAdjMatrixElem(int i, int j) { return adjMatrix[i][j]; }//получить элемент матрицы смежности

void SetEdgeZero(int i, int j) { adjMatrix[i][j] = 0; adjMatrix[j][i] = 0; }//занулить элемент матрицы (для Коммивояжера)

int GetAmountVerts() { return vertList.size(); } //получить количество вершин графа

int GetVertPos(const int& vertex); //получить позицию вершины int GetAmountEdges(); //получить количество ребер графа

void InsertVertex(const int& vertex); //добавить вершину void InsertEdge(const int& vertex1, const int& vertex2, int weigth);//добавить ребро весом weight между vertex1 и vertex2

void Print(); //вывод матрицы смежности в консоль

void EraseLastVert(); //удалить последнюю вершину в векторе вершин

void EraseEdge(const int& vertex1, const int& vertex2);//удалить ребро между vertex1 и vertex2

void DrawGraph(); //рисование графа OpenGL

};

extern Graph graph; //создается объект класса *Graph*

//

// Функции для решения задачи Коммивояжера

//

void Print\_Matrix(int\*\* matrix); //вывод текущей матрицы

bool Salesman\_Check(int\*\* matrix); //проверка: можно ли решить задачу Коммивояжера

int\*\* Change\_Matrix(); //создается матрица, заменяя 0 на -1

int\* Search\_MinElem(int\* line, int n); //поиск минимального элемента в строке матрицы

int\*\* Reduct\_Matrix(int\*\* oldmatrix); //редукция матрицы

int\*\* High\_Zero(int\*\* oldmatrix);

void Print\_Result(); //вывод результата решения задачи Коммивояжера

//

// Рисование кнопок (OpenGL)

//

void drawButton1(); //кнопка "Make New Graph / Circular View"

void drawButton2(); //кнопка "Salesman Task"

void drawButton3(); //кнопка "Print Matrix In Console"

void drawButton4(); //кнопка "Add Vertex"

void drawButton5(); //кнопка "Delete Vertex"

void drawButton6(); //кнопка "Add Edge"

void drawButton7(); //кнопка "Delete Edge"

void drawRadius(); //значение радиуса в окне OpenGL

//

// Рисование элементов графа (OpenGL)

//

void drawCircle(int x, int y, int R); //рисование круга

void drawBorderedCircle(int x, int y, int R); //рисование круга, если курсор в зоне вершины

void drawText(int nom, int x1, int y1); //рисование веса ребра

void drawLine(int text, int x0, int y0, int x1, int y1); //рисование ребра

void drawVertex(int n); //рисование вершины

void makeGraph(); //создание графа

void setCoords(int i, int n); //установка координат для вершин графа

//

// Функции - проверки

//

int Circle\_Check(int x, int y); //проверка: курсор в зоне вершины или нет

void Button\_Check(int x, int y); //проверка: курсор в зоне кнопки или нет

//

// Функции отслеживания поведения мыши

//

void mouseMove(int x, int y); //отслеживание движения курсора мыши

void mouseClick(int button, int state, int x, int y); //функция регистрации нажатий мыши

//

// Изображение

//

void reshape(int w, int h); //отвечает за масштабирование

void display(); //отвечает за изображение

3. Файл <Graph.cpp>

#include "Graph.h"

using namespace std;

//

// Определяем стартовые размеры окна

//

int WinW = 1080;

int WinH = 780;

bool\* Mouse\_Vert = new bool[maxSize]; //массив меток (курсор в зоне вершины или нет)

bool standView = false; //отвечает за перестроение графа в стандартный вид (все вершины расположены по кругу)

bool Vert\_Move = false; //отвечает за перемещение вершин

int R; //радиус вершин графа

int Button\_Flag; //если курсор в зоне кнопки, принимает значение номера кнопки (подсветка кнопок при наведении)

int Moving\_Vertex; //номер перемещаемой вершины

int CursorX; //позиция курсора по Х

int CursorY; //позиция курсора по У

vertCoord vertC[maxSize + 2]; //массив структур (координаты вершин), +2 чтобы не вылетало искл., при заполнении до maxSize

Graph graph;

vector<pair<int, int>> Way;

vector<int> New\_Way;

Graph::Graph() //конструктор, заполнение матрицы смежности 0 и меток Mouse\_Vert

{

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

{

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

{

adjMatrix[i][j] = 0;

}

Mouse\_Vert[i] = false;

}

}

Graph::~Graph() //деструктор

{ }

int Graph::GetVertPos(const int& vertex) //получение номера вершины в векторе вершин

{

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

{

if (vertList[i] == vertex)

return i;

}

return -1;

}

bool Graph::IsEmpty() //проверка на пустоту

{

if (vertList.size() != 0)

return false;

else

return true;

}

bool Graph::IsFull() //проверка на полноту

{

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

}

void Graph::InsertVertex(const int& vertex) //добавление новой вершины

{

if (!IsFull())

vertList.push\_back(vertex);

else

{

cout << "Граф уже заполнен. Невозможно добавить новую вершину " << endl;

return;

}

}

void Graph::InsertEdge(const int& vertex1, const int& vertex2, int weight) //добавление нового ребра

{

if (weight < 1)

{

cout << "\nДанная величина веса некорректна\n";

return;

}

if (GetVertPos(vertex1) != (-1) && GetVertPos(vertex2) != (-1)) //если вершины есть в графе

{

int vertPos1 = GetVertPos(vertex1); //находим позиции вершин

int vertPos2 = GetVertPos(vertex2);

if (adjMatrix[vertPos1][vertPos2] != 0 && adjMatrix[vertPos2][vertPos1] != 0)//если между ними уже есть ребро

{

cout << "Ребро между вершинами уже есть" << endl;

return;

}

else //иначе добавляем ребро

{

adjMatrix[vertPos1][vertPos2] = weight;

}

}

else

{

cout << "Обеих вершин (или одной из них) нет в графе " << endl;

return;

}

}

void Graph::Print() //вывод матрицы смежности на экран

{

if (!IsEmpty())

{

cout << "Матрица смежности графа: " << 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 << "\nГраф пуст\n" << endl;

}

void Graph::EraseLastVert() //удаление последней вершины

{

if (IsEmpty())

{

cout << "\nГраф пуст\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) //удаление ребра

{

if (GetVertPos(vertex1) != (-1) && GetVertPos(vertex2) != (-1)) //если обе вершины есть в графе

{

int vertPos1 = GetVertPos(vertex1); //находим позиции вершин

int vertPos2 = GetVertPos(vertex2);

if (adjMatrix[vertPos1][vertPos2] == 0) //если между вершинами ребра нет

{

cout << "Ребра между данными вершинами нет\n" << endl;

return;

}

else //иначе удаляем ребро (зануляем соответствуюшую ячейку в матрице смежности)

{

adjMatrix[vertPos1][vertPos2] = 0;

}

}

else //если хотя бы одной вершины в графе нет

{

cout << "Обеих вершин (или одной из них) нет в графе " << endl;

return;

}

}

int Graph::GetAmountEdges() //получение количества ребер в графе

{

int numb = 0;

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

{

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

{

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

{

numb++;

}

}

}

return numb;

}

void Graph::DrawGraph() //рисование графа

{

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\*\* Change\_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 << "Начальная матрица: \n";

Print\_Matrix(matrix);

return matrix;

}

int\* Search\_MinElem(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] = Search\_MinElem(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 << "\nРедуцированная матрица: \n";

Print\_Matrix(matrix);

return matrix;

}

int\*\* High\_Zero(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;

Way.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 << "Матрица после удаления 0 с наибольшей оценкой: \n";

Print\_Matrix(matrix);

cout << "\nПромежуточные отрезки путей: ";

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

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

cout << endl;

return matrix;

}

void Print\_Result()

{

int second = Way[0].second;

int i = 2;

New\_Way.push\_back(Way[0].first);

New\_Way.push\_back(Way[0].second);

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

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

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

{

second = Way[j].second;

New\_Way.push\_back(second);

i++;

}

cout << "Ответ: ";

for (int i = 0; i < New\_Way.size(); i++)

{

cout << New\_Way[i];

if (i != New\_Way.size() - 1)

cout << " -> ";

}

int sum = 0;

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

{

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

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

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

}

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

}

//

// Реализация функций рисования кнопок

//

void drawButton1()

{

if (Button\_Flag == 1)

glColor3f(0.565, 0.933, 0.565);

else

glColor3f(0.9, 0.8, 0.9);

glBegin(GL\_QUADS);

glVertex2i(50, WinH - 20);

glVertex2i(50, WinH - WinH / 7);

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

glVertex2i(WinW / 7, WinH - 20);

glEnd();

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

glBegin(GL\_LINE\_LOOP);

glVertex2i(50, WinH - 20);

glVertex2i(50, WinH - WinH / 7);

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

glVertex2i(WinW / 7, WinH - 20);

glEnd();

string name = "MAKE NEW GRAPH/";

string name1 = "CIRCULAR VIEW";

glRasterPos2i(WinW / 21, 0.92 \* WinH);

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

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

glRasterPos2i(WinW / 18, (0.92 \* WinH) - 15);

for (int i = 0; i < name1.length(); i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10, name1[i]);

}

void drawButton2()

{

if (Button\_Flag == 2)

glColor3f(0.565, 0.933, 0.565);

else

glColor3f(0.9, 0.8, 0.9);

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 = "\"SALESMAN\" TASK";

glRasterPos2i(WinW / 19, 0.77 \* WinH);

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

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

}

void drawButton3()

{

if (Button\_Flag == 3)

glColor3f(0.565, 0.933, 0.565);

else

glColor3f(0.9, 0.8, 0.9);

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";

string name1 = "IN CONSOLE";

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

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

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

glRasterPos2i(WinW / 16, (0.63 \* WinH) - 15);

for (int i = 0; i < name1.length(); i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10, name1[i]);

}

void drawButton4()

{

if (Button\_Flag == 4)

glColor3f(0.565, 0.933, 0.565);

else

glColor3f(0.9, 0.8, 0.9);

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 VERTEX";

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

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

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

}

void drawButton5()

{

if (Button\_Flag == 5)

glColor3f(0.565, 0.933, 0.565);

else

glColor3f(0.9, 0.8, 0.9);

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 VERTEX";

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

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

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

}

void drawButton6()

{

if (Button\_Flag == 6)

glColor3f(0.565, 0.933, 0.565);

else

glColor3f(0.9, 0.8, 0.9);

glBegin(GL\_QUADS);

glVertex2i(50, WinH - 5 \* (WinH / 7) - 20);

glVertex2i(50, WinH - 6 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 6 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 5 \* (WinH / 7) - 20);

glEnd();

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

glBegin(GL\_LINE\_LOOP);

glVertex2i(50, WinH - 5 \* (WinH / 7) - 20);

glVertex2i(50, WinH - 6 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 6 \* (WinH / 7));

glVertex2i(WinW / 7, WinH - 5 \* (WinH / 7) - 20);

glEnd();

string name = "ADD EDGE";

glRasterPos2i(WinW / 14.8, 0.2 \* WinH);

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

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

}

void drawButton7()

{

if (Button\_Flag == 7)

glColor3f(0.565, 0.933, 0.565);

else

glColor3f(0.9, 0.8, 0.9);

glBegin(GL\_QUADS);

glVertex2i(50, WinH - 6 \* (WinH / 7) - 20);

glVertex2i(50, 10);

glVertex2i(WinW / 7, 10);

glVertex2i(WinW / 7, WinH - 6 \* (WinH / 7) - 20);

glEnd();

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

glBegin(GL\_LINE\_LOOP);

glVertex2i(50, WinH - 6 \* (WinH / 7) - 20);

glVertex2i(50, 10);

glVertex2i(WinW / 7, 10);

glVertex2i(WinW / 7, WinH - 6 \* (WinH / 7) - 20);

glEnd();

string name = "DELETE EDGE";

glRasterPos2i(WinW / 16, 0.061 \* WinH);

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

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

}

void drawCircle(int x, int y, int R) //рисование обычного круга (курсор не в зоне вершины)

{

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

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(0.0f, 0.0f, 0.0f);

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) //рисование круга другим цветом (курсор в зоне вершины)

{

glColor3f(0.565, 0.933, 0.565);

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();

}

void drawText(int text, int x1, int y1) //рисование веса ребра / значения вершины

{

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]);

}

void drawLine(int text, int x0, int y0, int x1, int y1) //рисование ребра ориентированного взвешенного графа

{

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 = 1.0f / 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();

}

void drawVertex(int n) //рисование вершины (круг + значение)

{

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);

}

}

void setCoords(int i, int n) //расстановка координат вершин по кругу радиуса R\_ и установка радиуса вершин R

{

int R\_;

int x0 = WinW / 2;

int y0 = WinH / 2;

if (WinW > WinH)

{

R = 5 \* (WinH / 13) / n;

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

}

else

{

R = 5 \* (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;

}

void makeGraph() //создание графа

{

standView = false;

int amountVerts, amountEdges, sourceVertex, targetVertex, edgeWeight;

cout << "Введите количество вершин в графе: "; cin >> amountVerts;

cout << "Введите количество ребер в графе: "; cin >> amountEdges;

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

graph.InsertVertex(i);

}

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

{

cout << "Исходная вершина: "; cin >> sourceVertex;

cout << "Конечная вершина: "; cin >> targetVertex;

cout << "Вес ребра: "; cin >> edgeWeight;

graph.InsertEdge(sourceVertex, targetVertex, edgeWeight);

}

cout << endl;

graph.Print();

}

bool Salesman\_Check(int\*\* matrix) //проверка: корректна ли задача Коммивояжера

{

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;

}

int Circle\_Check(int x, int y) //проверка: курсор в зоне вершины или нет

{

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;

}

void Button\_Check(int x, int y) //проверка: курсор в зоне кнопки или нет

{

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 if (x > 50 && x < WinW / 7 && y > WinH - 6 \* (WinH / 7) && y < WinH - 5 \* (WinH / 7) - 20)

Button\_Flag = 6;

else if (x > 50 && x < WinW / 7 && y > 10 && y < WinH - 6 \* (WinH / 7) - 20)

Button\_Flag = 7;

else

Button\_Flag = 0;

}

void mouseMove(int x, int y) //отслеживание движения курсора и возможность двигать вершину

{

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 drawRadius() //рисование значения радиуса вершин в окне

{

int Rad = 0;

if (graph.GetAmountVerts() != 0)

Rad = R;

int space = 24;

string r = "RADIUS = " + to\_string(Rad);

glRasterPos2i((WinW / 7) + 50, WinH - 50);

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

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, r[i]);

}

void mouseClick(int button, int state, int x, int y) //регистрация нажатий мыши

{

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 (button == 3 || button == 4)

{

standView = true;

if (button == 3)

{

if (R != 300)

R++;

}

else if (button == 4)

{

if (R != 15)

R--;

}

}

if (x >= 50 and x <= (WinW / 7) and y >= 20 and y <= (WinH / 7))

{

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

{

New\_Way.clear();

Way.clear();

Graph New;

graph = New;

makeGraph();

return;

}

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

{

standView = false;

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)

{

New\_Way.clear();

Way.clear();

cout << "\nЗадача Коммивояжера:\n";

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

bool checker = Salesman\_Check(matrix);

if (!checker)

{

cout << "\nЗадача Коммивояжера для данного графа некорректна\n\n";

return;

}

int n = graph.GetAmountVerts();

while (Way.size() < n)

matrix = High\_Zero(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 cur = graph.GetAmountVerts();

graph.InsertVertex(cur + 1);

vertC[cur].x = WinW / 2;

vertC[cur].y = WinH / 2;

return;

}

}

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

{

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

{

graph.EraseLastVert();

int cur = graph.GetAmountVerts();

return;

}

}

if (x >= 50 && x <= WinW / 7 && y <= 6 \* (WinH / 7) && y >= 5 \* (WinH / 7) + 20)

{

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

{

int sourceVertex, targetVertex, edgeWeight, vertNum;

cout << "Количество добавляемых ребер: "; cin >> vertNum;

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

{

cout << "Исходная вершина: "; cin >> sourceVertex;

cout << "Конечная вершина: "; cin >> targetVertex;

cout << "Вес ребра: "; cin >> edgeWeight;

graph.InsertEdge(sourceVertex, targetVertex, edgeWeight);

}

return;

}

}

if (x > 50 && x < WinW / 7 && y <= WinH - 10 && y >= 6 \* (WinH / 7) + 20)

{

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

{

int sourceVertex, targetVertex;

cout << "Исходная вершина: "; cin >> sourceVertex;

cout << "Конечная вершина: "; 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(1.0, 1.0, 1.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

drawButton1();

drawButton2();

drawButton3();

drawButton4();

drawButton5();

drawButton6();

drawButton7();

graph.DrawGraph();

drawRadius();

glutSwapBuffers();

}

* 1. **UML – диаграмма**

На рисунке 1, представленном ниже, изображена UML – диаграмма класса *Graph*. В диаграмме представлены все методы, необходимые для работы с классом *Graph* и два *public* поля: матрица смежности графа и вектор вершин.

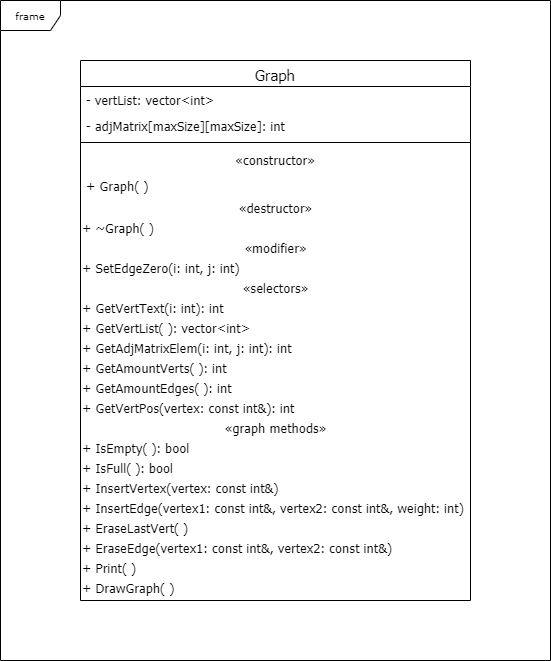


Рисунок 1 – UML-диаграмма класса *Graph*

* 1. **Результаты работы программы**

**2.6 Библиотеки**

В разработке программы были использованы следующие стандартные библиотеки *Visual Studio 2019* и *API OpenGL*:

1. **<vector>** - для хранения вершин графа, кратчайшего пути и его отрезков;
2. **<iostream>** - для ввода/вывода информации в консоль;
3. **<glut.h>** - для использования методов OpenGL;
4. **<string>** - для использования строкового типа данных и его методов;
5. **<iomanip>** - для табличного вывода матрицы смежности графа на экран.

# **2.7 Рефлексия**

В процессе разработки программы возникало немало трудностей. Было нелегко организовать перемещение вершин графа в окне. Необходимо было правильно переопределить координаты перемещаемой вершины, чтобы движение происходило именно вслед за позицией курсора мыши. На мой взгляд, получилось создать комфортный и приятный интерфейс и визуализировать граф таким образом, что у пользователя не возникало бы никаких вопросов в процессе использования программы.

# **4. Заключение**

В процессе разработки программ для решения задачи коммивояжера было изучено много нового теоретического материала по платформе *OpenGL*. Работа над творческим проектом позволила повысить уровень программирования на языке *С++*, показала большие возможности данного языка и его взаимодействие с другими платформами. В результате все поставленные задачи были выполнены. Особенно хочется отметить такие конструкторские решения в программах, как перемещение вершин графа. Разработанные приложения можно усовершенствовать в перспективе, добавить новые функции и методы для расширения возможностей работы и оптимизации алгоритмов. На данный момент требуемый результат получен и программы работают корректно.

Приобретенный опыт в результате выполнения творческой работы поможет разрабатывать более сложные алгоритмы не только на языке *С++*, но и на других языках программирования.