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 ТВОРЧЕСКОЙ РАБОТЕ №2**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Задача Коммивояжера»

Выполнила работу

Студентка группы РИС-23-3Б

Шуракова А.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2024

**Введение**

Задача коммивояжёра является классической задачей оптимизации, которая заключается в поиске кратчайшего замкнутого маршрута, проходящего через заданный набор городов и возвращающегося в исходный город. Метод ветвей и границ метод основан на разбиении исходной задачи на более мелкие подзадачи, а затем применении оценок верхней и нижней границ, чтобы исключить из рассмотрения часть возможных решений, которые не могут быть оптимальными. Эта задача имеет широкое применение в различных областях, таких как логистика, планирование маршрутов и многих других.

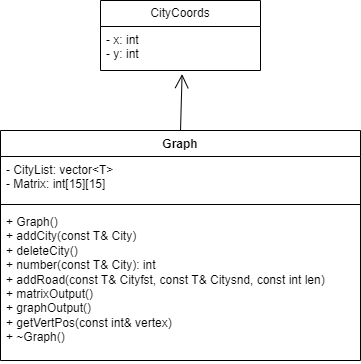
**Постановка задачи**

• Реализация приложения на основе OpenGL, для решения задачи Коммивояжёра.

• Изучить метод ветвей и границ.

• Визуализация графа и решения задачи.

**UML-диаграмма**

****

**Код программы**

**traveling\_salesman.h**

#pragma once

#include "freeglut.h"

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <vector>

#include <sstream>

#include <string>

using namespace std;

bool pr = false; // Объявление и инициализация логической переменной

vector<int> TSPRoute; // Объявление вектора целых чисел TSPRoute для хранения маршрута коммивояжера

struct CityCoords // Cтруктуры для хранения координат города

{

int x, y; // Поля структуры

};

CityCoords CityCoord[15]; // Объявление массива из 15 элементов

template<class T>

class Graph

{

vector<T> CityList; //контейнер вектор, хранит вершины графа

public:

int Matrix[15][15]; //матрица смежности

Graph() {

// Цикл для инициализации матрицы смежности

for (int i = 0; i < 15; i++) {

for (int j = 0; j < 15; j++) {

// Инициализация элементов матрицы нулями

this->Matrix[i][j] = 0;

}

}

}

//метод добавления вершин

void addCity(const T& City) {

// Если количество городов уже равно 15

if (this->CityList.size() == 15) {

cout << "\nОшибка! Вы ввели более 15 городов";

return;

}

else {

// Добавление нового города в список

this->CityList.push\_back(City);

}

}

//метод удаления вершин

void deleteCity() {

if (this->CityList.size() == 0) {

// Если список городов пуст

cout << "\nОшибка! Городов в списке больше нет: удалять нечего";

}

else {

// Удаление последнего города из списка

this->CityList.pop\_back();

}

}

// Метод для поиска номера вершины (города) в списке

int number(const T& City) {

for (int i = 0; i < this->CityList.size(); i++) {

// Если вершина найдена

if (this->CityList[i] == City) {

// Возвращаем ее номер

return i;

}

}

return -1;

}

// Метод для добавления ребра между двумя вершинами (городами)

void addRoad(const T& Cityfst, const T& Citysnd, const int len) {

int Number1, Number2;

// Если первый город не найден

if (this->number(Cityfst) == -1) {

cout << "\nОшибка! Города с номером " << Citysnd << " не существует\n";

}

else {

// Если второй город не найден

if (this->number(Citysnd) == -1) {

cout << "\nОшибка! Города с номером " << Citysnd << " не существует\n";

}

else {

Number1 = number(Cityfst); // Получаем номер первого города

Number2 = number(Citysnd); // Получаем номер второго города

// Если ребро между городами еще не существует

if (Matrix[Number1][Number2] == 0 && Matrix[Number2][Number1] == 0) {

// Добавляем ребро с указанной длиной

this->Matrix[Number1][Number2] = len;

// Добавляем ребро в обратном направлении

this->Matrix[Number2][Number1] = len;

}

else {

cout << "\nМежду этими городами уже проложен путь";

}

}

}

}

//вывод матрицы смежности

void matrixOutput() {

if ((this->CityList.size()) == 0) {

cout << "\nСписок городов пуст. Матрица смежности графа не может быть выведена";

}

else {

cout << "\nМатрица смежности городского графа:\n";

cout << "\t";

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

cout << CityList[i] << "\t|\t";

}

cout << "\n\n";

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

cout << CityList[i] << "\t";

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

cout << this->Matrix[i][j] << "\t|\t";

}

cout << "\n\n";

}

}

}

void graphOutput(); // Объявление прототипа функции

int getVertPos(const int& vertex); // Объявление прототипа функции getVertPos()

~Graph() {}; // Деструктор класса Graph

};

int countCity;

int\*\* s;

int\* road;

int\*\*\* Matrixnew;

int\* path;

int WinWeight;

int WinHeight;

int R;

int CityQuantity = 0;

int RoadQuantity = 0;

Graph<int> graph;

//метод для нахождения оптимального маршрута для задачи коммивояжера

void findAnswer(int\*\*\* Matrixnew, int countCity, int\*\* HelpMatrix, int\* path) {

//для строк

for (int l = 0; l < countCity; l++) {

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

int min = 1000000;

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

// Если есть путь между городами i и j, и его длина меньше текущего минимума

if (Matrixnew[i][j] && min > \*Matrixnew[i][j])

min = \*Matrixnew[i][j];

}

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

// Если есть путь между городами i и j

if (Matrixnew[i][j])

\*Matrixnew[i][j] -= min;

}

}

//для столбцов

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

int min = 1000000;

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

if (Matrixnew[i][j] && min > \*Matrixnew[i][j])

min = \*Matrixnew[i][j];

}

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

if (Matrixnew[i][j])

\*Matrixnew[i][j] -= min;

}

}

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

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

HelpMatrix[i][j] = 0;

}

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

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

if (Matrixnew[i][j] && !\*Matrixnew[i][j]) {

int colmin = 1000000; // Инициализация минимального значения в столбце

int stringmin = 1000000; // Инициализация минимального значения в строке

for (int l = 0; l < countCity; l++)

// Находим минимальное значение в столбце j, исключая строку i

if (l != i && Matrixnew[l][j] && colmin > \*Matrixnew[l][j])

colmin = \*Matrixnew[l][j];

for (int l = 0; l < countCity; l++)

// Находим минимальное значение в строке i, исключая столбец j

if (l != j && Matrixnew[i][l] && stringmin > \*Matrixnew[i][l])

stringmin = \*Matrixnew[i][l];

// Записываем в вспомогательную матрицу сумму минимальных значений в строке и столбце

HelpMatrix[i][j] = colmin + stringmin;

}

}

int mcost = 0, mi = 0, mj = 0;

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

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

if (Matrixnew[i][j] && mcost < HelpMatrix[i][j])

{

mcost = HelpMatrix[i][j];

mi = i;

mj = j;

}

path[mi] = mj;

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

Matrixnew[i][mj] = nullptr;

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

Matrixnew[mi][i] = nullptr;

Matrixnew[mj][mi] = nullptr;

}

}

//метод для создания матрицы

void createMatrix(int\*\*\*& Matrixnew, int& countCity, int\*\*& s, int\*& road) {

countCity = CityQuantity;

s = new int\* [countCity]; // Выделение памяти для массива указателей

road = new int[countCity]; // Выделение памяти для массива int

Matrixnew = new int\*\* [countCity]; // Выделение памяти для массива указателей на указатели

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

s[i] = new int[countCity]; // Выделение памяти для каждого массива int в s

}

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

Matrixnew[i] = new int\* [countCity]; // Выделение памяти для каждого массива указателей на int в Matrixnew

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

// Если путь между городами i и j отсутствует

if (graph.Matrix[i][j] == 0) {

Matrixnew[i][j] = nullptr;

continue;

}

// Выделение памяти и присвоение значения матрицы

Matrixnew[i][j] = new int(graph.Matrix[i][j]);

}

}

}

//метод для решения задачи Коммивояжера

void TSP(int\*\*\* Matrixnew, int countCity, int\*\* s, int\* road)

{

TSPRoute.clear(); // Очистка вектора CommiRoads

createMatrix(Matrixnew, countCity, s, road);

findAnswer(Matrixnew, countCity, s, road);

int sumlength = 0;

for (int i = 0, j = 0; i < countCity; i++) {

j = road[i];

sumlength += graph.Matrix[i][j]; // Вычисление общей длины пути

}

cout << "\n\nКратчайший путь, пройденный Коммивояжером ";

int selLenght = 0;

for (int l = 0; l < countCity;) {

for (int i = 0, j = 0; i < countCity; i++) {

if ((selLenght == 0) || (selLenght == i + 1)) {

if (selLenght == 0) {

TSPRoute.push\_back(i + 1);

cout << i + 1;

}

j = road[i];

selLenght = j + 1;

if (selLenght > 0) {

cout << " -> " << selLenght;

TSPRoute.push\_back(selLenght);

}

l++;

}

}

}

cout << "\n\nОбщая длина пути, который пройдет Коммивояжер: " << sumlength << "\n\n";

if (pr)

pr = false;

else

pr = true;

}

//метод для вывода всех путей

void ouputRoads(int\*\*\* Matrixnew, int countCity, int\*\* s, int\* road) {

createMatrix(Matrixnew, countCity, s, road);

findAnswer(Matrixnew, countCity, s, road);

int sumlength = 0;

cout << "\nВсе пути между городами, по которым пройдет коммивояжер: \n";

for (int i = 0, j = 0; i < countCity; i++) {

j = road[i];

cout << i + 1 << " -> " << j + 1 << '\n';

}

}

//метод для создания графа

Graph <int> createGraph() {

Graph<int> Graph;

int first, second, length;

int\* firstCity;

int\* secondCity;

int\* NewCity;

cout << "\nВведите количество городов: ";

cout << "\n";

cin >> CityQuantity;

cout << "\nВведите количество маршрутов между этими городами: ";

cout << "\n";

cin >> RoadQuantity;

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

NewCity = &i;

Graph.addCity(\*NewCity);

}

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

cout << "\nВведите номер города, из которого начинается маршрут " << i + 1 << " : ";

cin >> first;

firstCity = &first;

cout << "\nВведите номер города, к которому ведет маршрут " << i + 1 << " : ";

cin >> second;

secondCity = &second;

cout << "\nВведите расстояние между городами " << first << " и " << second << " : ";

cin >> length;

Graph.addRoad(\*firstCity, \*secondCity, length);

}

return Graph;

}

void saveCity(int i, int countCity)

{

int NewR;

int x0 = WinWeight / 2; // Вычисление x-координаты центра окна

int y0 = WinHeight / 2; // Вычисление y-координаты центра окна

if (WinWeight > WinHeight) // Проверка соотношения сторон окна

{

// Вычисление радиуса города в зависимости от размеров окна и количества городов

R = 5 \* (WinHeight / 13) / countCity;

// Вычисление радиуса окружности, на которой будут располагаться города

NewR = WinHeight / 2 - R - 10;

}

else {

// Вычисление радиуса города в зависимости от размеров окна и количества городов

R = 5 \* (WinWeight / 13) / countCity;

// Вычисление радиуса окружности, на которой будут располагаться города

NewR = WinWeight / 2 - R - 10;

}

// Вычисление угла для позиционирования города

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

float y1 = NewR \* cos(theta) + y0; // Вычисление y-координаты города

float x1 = NewR \* sin(theta) + x0; // Вычисление x-координаты города

// Сохранение x-координаты города

CityCoord[i].x = x1;

// Сохранение y-координаты города

CityCoord[i].y = y1;

}

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

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

float x1, y1;

glBegin(GL\_POLYGON); // Начинаем рисование многоугольника (окружности).

for (int i = 0; i < 360; i++) { // Итерируемся по углам от 0 до 360 градусов.

float theta = 2.0f \* 3.1415926f \* float(i) / float(360); // Вычисляем угол в радианах.

x1 = R \* sin(theta) + x; // Вычисляем координату x текущей точки окружности.

y1 = R \* cos(theta) + y; // Вычисляем координату y текущей точки окружности.

glVertex2f(x1, y1); // Устанавливаем точку в текущих координатах.

}

glEnd(); // Завершаем рисование многоугольника (окружности).

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

float x2, y2;

glBegin(GL\_LINE\_LOOP); // Начинаем рисование линии контура окружности.

for (int i = 0; i < 360; i++) { // Итерируемся по углам от 0 до 360 градусов.

float theta = 2.0f \* 3.1415926f \* float(i) / float(360); // Вычисляем угол в радианах.

x2 = R \* sin(theta) + x; // Вычисляем координату x текущей точки контура окружности.

y2 = R \* cos(theta) + y; // Вычисляем координату y текущей точки контура окружности.

glVertex2f(x2, y2); // Устанавливаем точку в текущих координатах.

}

glEnd(); // Завершаем рисование линии контура окружности.

}

void newTSPText(int nom, int x1, int y1)

{

// Установка шрифта для рендеринга текста

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

// Преобразование числа nom в строку

string s = to\_string(nom);

// Установка позиции для рендеринга текста

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void newCityText(int nom, int x1, int y1)

{

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

string s = to\_string(nom);

// Установка цвета для рендеринга текста (бирюзовый)

glColor3f(0.0f, 0.5f, 0.5f);

// Установка позиции

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); j++) {

glutBitmapCharacter(font, s[j]);

}

}

void newText(int nom, int x1, int y1)

{

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

string s = to\_string(nom);

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

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); j++) {

glutBitmapCharacter(font, s[j]);

}

}

//отрисовка города

void drawCity(int countCity)

{

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

// Рисование круга для каждого города

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

// Рендеринг текста для каждого города

newCityText(i + 1, CityCoord[i].x, CityCoord[i].y);

}

}

//отрисовка дорог

void drawRoad(int text, int x0, int y0, int x1, int y1, bool b)

{

// Если b равно true, установить цвет линии на синий

if (b)

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

else

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

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

if (b) {

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

}

else {

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

}

}

//отрисовка задачи Коммивояжера

void drawTSPRoad(int text, int x0, int y0, int x1, int y1, bool b)

{

// Если b равно true, установить цвет линии на синий

if (b)

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

else

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

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

if (b) {

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

}

else {

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

}

}

template<class T>

void Graph<T>::graphOutput()

{

// Сохранение координат всех городов

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

saveCity(i, CityList.size());

}

// Отрисовка дорог между городами

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

for (int j = i + 1; j < CityList.size(); j++) {

if (Matrix[i][j] != 0) {

drawRoad(Matrix[i][j], CityCoord[i].x, CityCoord[i].y, CityCoord[j].x, CityCoord[j].y, false);

}

}

}

// Отрисовка самих городов

drawCity(CityList.size());

}

//метод для изменения размеров окна

void reshape(int w, int sumlength) {

WinWeight = w; // Установка ширины окна

WinHeight = sumlength; // Установка высоты окна

glViewport(0, 0, (GLsizei)WinWeight, (GLsizei)WinHeight); // Установка области просмотра

glMatrixMode(GL\_PROJECTION); // Выбор матрицы проекции

glLoadIdentity(); // Сброс матрицы проекции

gluOrtho2D(0, (GLdouble)WinWeight, 0, (GLdouble)WinHeight); // Установка ортогональной проекции

glutPostRedisplay(); // Запрос на перерисовку содержимого окна

}

void drawMenuText(string text, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18; // Установка шрифта для рендеринга текста

string s = text;

glRasterPos2i(x1 + 5, y1 - 20); // Установка позиции для рендеринга текста

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void drawMenu() {

int width = 60;

int lgth = 730;

glColor3d(0.0f, 0.0f, 1.0f); // Установка цвета кнопки

glBegin(GL\_QUADS); // Начало рисования четырехугольника

glVertex2i(width, lgth - width - 30);// Координаты верхней левой вершины

glVertex2i(width + 255, lgth - width - 30); // Координаты верхней правой вершины

glVertex2i(width + 255, lgth - width); // Координаты нижней правой вершины

glVertex2i(width, lgth - width); // Координаты нижней левой вершины

glEnd(); // Конец рисования четырехугольника

glColor3d(1.0, 1.0, 1.0); // Установка белого цвета для рисования текста

drawMenuText("Add a city", width + 100, lgth - width - 2); // Вывод текста

glColor3d(0.0f, 0.0f, 1.0f);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 70);

glVertex2i(width + 255, lgth - width - 70);

glVertex2i(width + 255, lgth - width - 40);

glVertex2i(width, lgth - width - 40);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Delete a city", width + 100, lgth - width - 42);

glColor3d(0.0f, 0.0f, 1.0f);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 110);

glVertex2i(width + 255, lgth - width - 110);

glVertex2i(width + 255, lgth - width - 80);

glVertex2i(width, lgth - width - 80);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText(" Output the matrix of roads", width, lgth - width - 82);

glColor3d(0.0f, 0.0f, 1.0f);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 180);

glVertex2i(width + 255, lgth - width - 180);

glVertex2i(width + 255, lgth - width - 120);

glVertex2i(width, lgth - width - 120);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText(" Calculate the path traveled ", width, lgth - width - 122);

glColor3d(1.0, 1.0, 1.0);

drawMenuText("a traveling salesman by", width + 50, lgth - width - 150);

glColor3d(0.0f, 0.0f, 1.0f);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 220);

glVertex2i(width + 255, lgth - width - 220);

glVertex2i(width + 255, lgth - width - 190);

glVertex2i(width, lgth - width - 190);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText(" Create new roads graph", width, lgth - width - 190);

glColor3d(0.0f, 0.0f, 1.0f);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 230);

glVertex2i(width + 255, lgth - width - 230);

glVertex2i(width + 255, lgth - width - 260);

glVertex2i(width, lgth - width - 260);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText(" List of possible roads", width, lgth - width - 230);

}

void mouseClick(int button, int stat, int x, int y) {

int width = 60;

int lgth = 730;

if (stat == GLUT\_DOWN) {

if (x > width && x < width + 135 && y > width && y < width + 30) {

// Обработка добавления города

int first;

int second;

int length;

int\* firstCity;

int\* secondCity;

cout << "\nНомер добавляемого города: ";

cin >> first;

firstCity = &first;

cout << "\nНомер города, с которым будет связан: ";

cin >> second;

cout << "\n\n";

secondCity = &second;

if (first > CityQuantity || second > CityQuantity) {

CityQuantity++;

int\* NewCity = &CityQuantity;

graph.addCity(\*NewCity);

}

cout << "Длина пути между городами: "; cin >> length; cout << endl;

graph.addRoad(\*firstCity, \*secondCity, length);

}

if (x > width && x < width + 135 && y > width + 40 && y < width + 70) {

int first;

int second;

int length;

int\* firstCity;

cout << "\nНомер удаляемого города: ";

cin >> first;

cout << "\n";

firstCity = &first;

if (first == CityQuantity) {

CityQuantity--;

graph.deleteCity();

}

else cout << "\nУдалить это город невозможно \n";

}

if (x > width && x < width + 240 && y > width + 80 && y < width + 100) {

graph.matrixOutput();

}

if (x > width && x < width + 240 && y > width + 120 && y < width + 140) {

TSP(Matrixnew, countCity, s, road);

}

if (x > width && x < width + 240 && y > width + 160 && y < width + 200) {

graph = createGraph();

}

if (x > width && x < width + 240 && y > width + 230 && y < width + 260) {

ouputRoads(Matrixnew, countCity, s, road);

}

}

glutPostRedisplay();

}

template<class T>

int Graph<T>::getVertPos(const int& vertex)

{

// Цикл, проходящий по всем городам в списке

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

if (CityList[i] == vertex)

return i;

return -1;

}

void display() {

glShadeModel(GL\_SMOOTH); // Установка сглаживания для отрисовки

glMatrixMode(GL\_PROJECTION); // Выбор матрицы проекции

glLoadIdentity(); // Сброс матрицы проекции

gluOrtho2D(0, WinWeight, 0, WinHeight); // Установка ортогональной проекции

glViewport(0, 0, WinWeight, WinHeight); // Установка области просмотра

glClearColor(0.0f, 1.0f, 1.0f, 1.0f); // Установка цвета фона

glClear(GL\_COLOR\_BUFFER\_BIT); // Очистка буфера цвета

graph.graphOutput(); // Отрисовка карты

if (pr)

{

for (int i = 0; i < TSPRoute.size() - 1; i++)

{

int vertPos1 = graph.getVertPos(TSPRoute[i]);

int vertPos2 = graph.getVertPos(TSPRoute[i + 1]);

drawTSPRoad(i + 1, CityCoord[vertPos1].x, CityCoord[vertPos1].y, CityCoord[vertPos2].x, CityCoord[vertPos2].y, true);

}

}

drawMenu(); // Отрисовка меню

glutSwapBuffers(); // Обмен буферами для отображения результата

}

**Traveling\_salesman.cpp**

#include <iostream>

#include "traveling\_salesman.h"

#include "freeglut.h"

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

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

{

system("chcp 1251");

glutInit(&argc, argv); // Инициализация GLUT с аргументами командной строки

graph = createGraph(); // Создание карты для задачи коммивояжера

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGB); // Установка режима отображения для OpenGL

glutInitWindowSize(1350, 730); // Задание размеров окна

glutCreateWindow("Коммивояжер"); // Создание окна с названием "Коммивояжер"

WinWeight = glutGet(GLUT\_WINDOW\_WIDTH); // Получение ширины окна

WinHeight = glutGet(GLUT\_WINDOW\_HEIGHT); // Получение высоты окна

//loadTexture();

glutDisplayFunc(display); // Регистрация функции отображения

glutReshapeFunc(reshape); // Регистрация функции изменения размеров окна

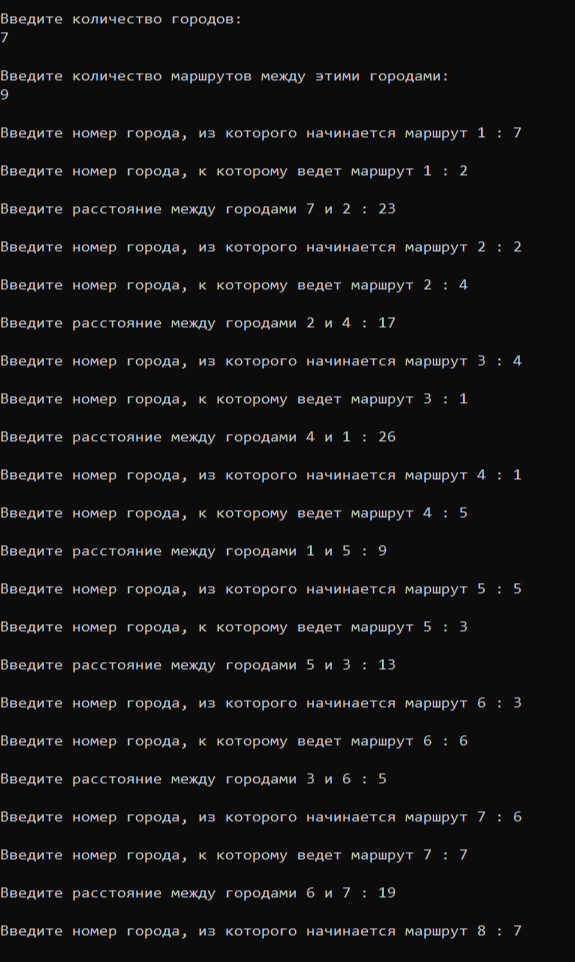
glutMouseFunc(mouseClick); // Регистрация функции обработки событий мыши

glutMainLoop(); // Запуск основного цикла GLUT

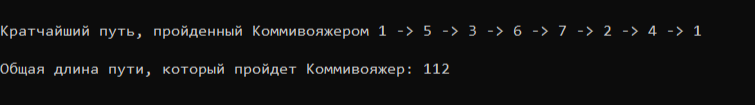
return 0;

}

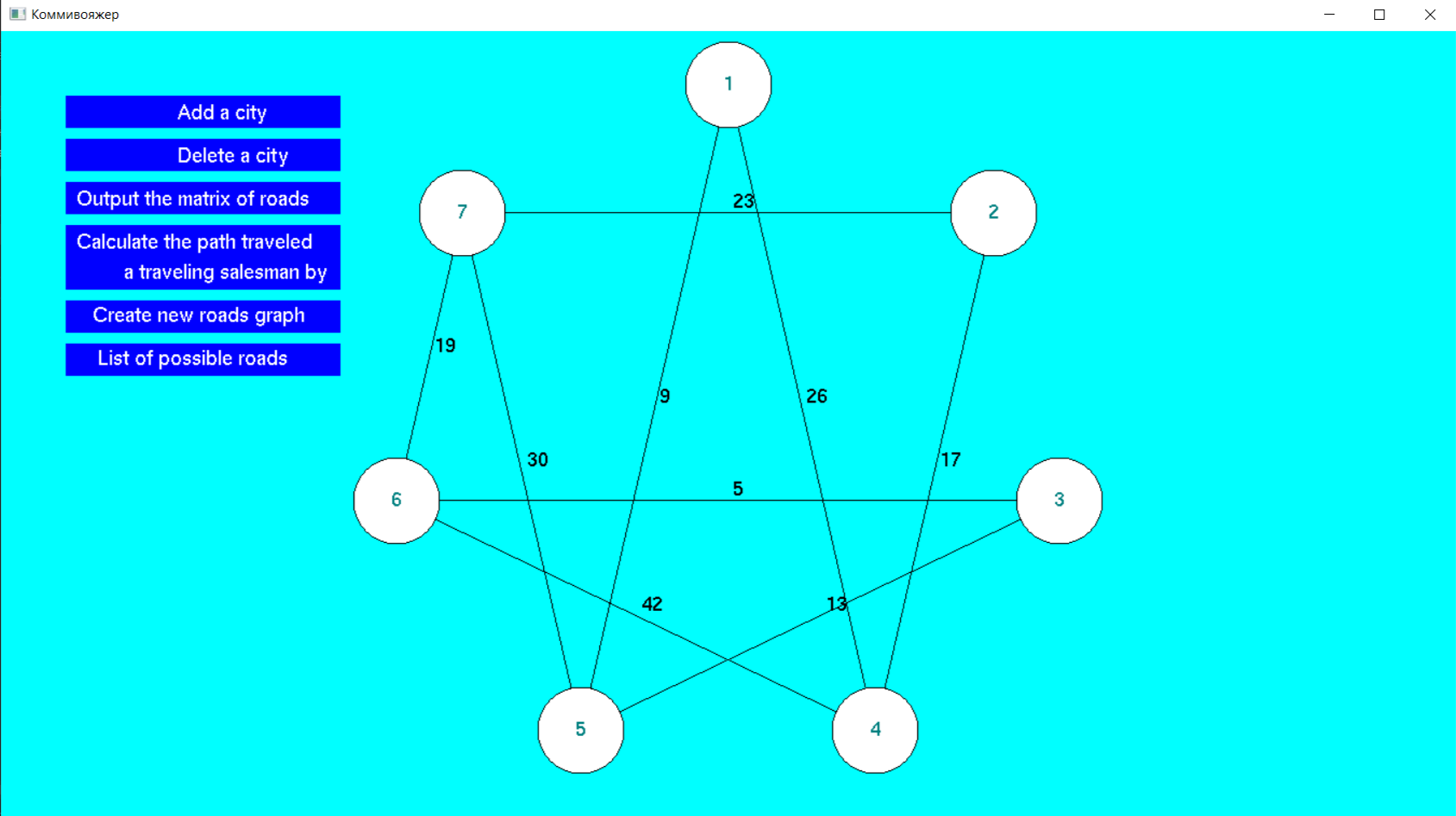
**Результат работы программы**

****

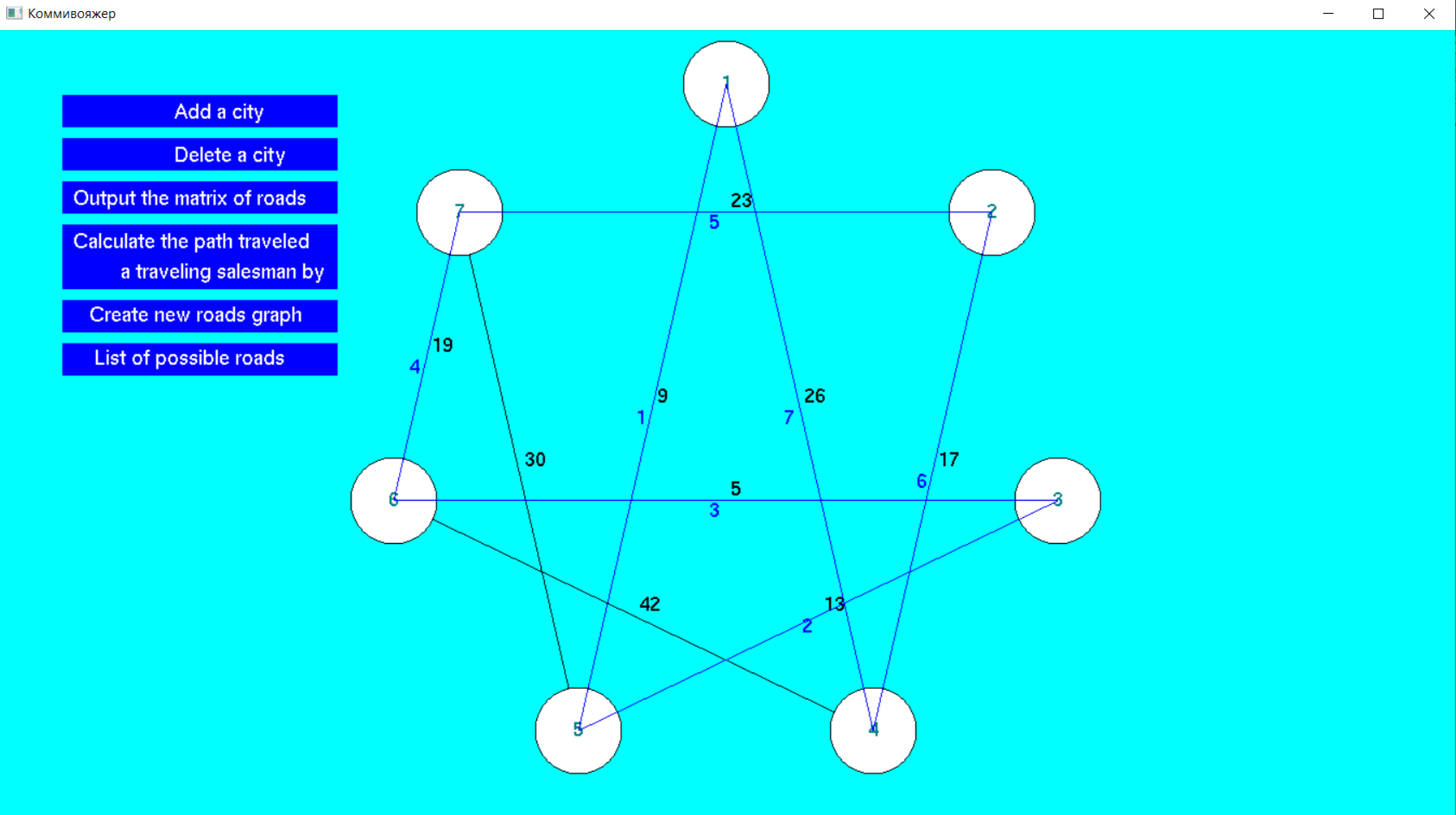
**Вывод решения задачи Коммивояжера**

****

**Результат работы программы в OpenGL**

****

**Результат решения задачи Коммивояжера в OpenGL**

****

**Вывод**

Программа выполняет свою задачу.

**Заключение**

Для решения задачи потребовались знания языка программирования, а конкретнее C++. Программа выполняет те условия, что были указаны в постановке задачи и работает без проблем.

**Ссылка на Git:**

https://github.com/Ananasic07/ARM-TSP/tree/main/Traveling%20salesman

**Ссылка на YouTube:**

https://youtu.be/v\_AsSs31x5M?si=d7vvq1emzgaoPZk2