Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

УДК 004

**Отчет по лабораторной работе**

**«Решение задачи коммивояжёра и его визуализация»**

Выполнили:

студенты группы РИС-23-1б

Пичайкин М. В., Гордеев В. А., Агзамов А. А., Комягин Д. А.

Проверил:

доцент кафедры ИТАС

Полякова О. А.

г. Пермь, 2024 г.

# **СОДЕРЖАНИЕ**

Введение 3

1 Алгоритм решения задачи коммивояжёра5

2 Разработка кода на языке программирования С++ 10

Заключение11

Приложение А12

# **ВВЕДЕНИЕ**

Задача коммивояжёра (Traveling Salesman Problem, TSP) является одной из классических задач комбинаторной оптимизации. В данной задаче необходимо найти кратчайший путь, проходящий через все города из заданного списка ровно один раз и возвращающийся в исходный город. TSP имеет огромное практическое значение в логистике, транспорте, проектировании микросхем, а также в других областях, где требуется оптимизация пути или маршрута.

Решение задачи коммивояжёра также является сложной задачей, требующей применения различных оптимизационных и алгоритмических подходов. Существует множество методов для решения TSP, включая методы полного перебора, приближенные алгоритмы, генетические алгоритмы, муравьиные алгоритмы и другие.

Исследование задачи коммивояжёра позволяет не только находить оптимальные или приближенные решения для конкретных ситуаций, но и изучать общие принципы оптимизации пути и маршрутов. Решение TSP также имеет важное значение в разработке алгоритмов для других комбинаторных задач и служит основой для изучения сложных оптимизационных проблем в области искусственного интеллекта и логистики.

Данная лабораторная работа заключается в анализе методов решения задачи коммивояжёра, исследовании их эффективности, изучении возможных оптимизаций, а также она включает в себя поиск наиболее эффективной визуализации решения. Исследование эффективности алгоритмов коммивояжёра требует изучения их скорости работы, точности результата, способности к решению больших задач и т.д.

Таким образом, выполнение данной лабораторной работы способствует развитию не только теоретических знаний, но и практических навыков в области разработки и оптимизации алгоритмов.

Объект исследования: вычислительный процесс нахождения оптимального пути методом ветвей и границ.

Предмет исследования: программа-приложение, определяющая кратчайший путь с помощью метода ветвей и границ для решения задачи коммивояжера.

Цель: реализация программы на языке программирования С++ для решения задачи коммивояжёра, выполнение визуализации решения.

Задачи:

1. Провести анализ алгоритмов решения;
2. Разработать кода на языке программирования С++;
3. Разработать визуализацию решения задачи коммивояжёра.

# **1. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА**

Перед тем как решать задачу коммивояжёра, мы должны проанализировать все возможные методы решения, выявить их достоинства и недостатки, быстроту, доступность (простоту). Для этой задачи были разработаны следующие алгоритмы:

**1.Полный перебор:**

Полный перебор (или метод «грубой силы») — метод решения задачи путем перебора всех возможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение нескольких лет или даже столетий.

**Анализ:**

**Cильный стороны:**

1. Гарантированное нахождение оптимального решения: Полный перебор рассматривает все возможные варианты, поэтому гарантированно найдет наилучшее решение.
2. Простота реализации: Алгоритм полного перебора обычно прост в реализации, поскольку он прямолинеен и не требует сложных эвристик.

**Слабые стороны:**

1. Экспоненциальная сложность: С увеличением числа вариантов решения время работы алгоритма возрастает экспоненциально.
2. Неэффективность для больших задач: Для задач с большим пространством решений полный перебор может быть непрактичным из-за его огромных вычислительных требований.

**2. Случайный перебор:**

Обычно выбор решения можно представить последовательностью выборов. Если делать эти выборы с помощью какого-либо случайного механизма, то решение находится очень быстро, так что можно находить решение многократно и запоминать «рекорд», т. е. наилучшее из встретившихся решений. Этот наивный подход существенно улучшается, когда удается учесть в случайном механизме перспективность тех или иных выборов, т. е. комбинировать случайный поиск с эвристическим методом и методом локального поиска. Такие методы применяются, например, при составлении расписаний для Аэрофлота.

**Анализ:**

**Сильные стороны:**

1. Быстрота: Случайный перебор может быстро находить приемлемые решения, особенно если пространство решений велико.
2. Простота: Алгоритм случайного перебора прост в реализации.
3. Возможность улучшения: Случайный перебор можно улучшить, добавив эвристики и методы локального поиска, что делает его более эффективным.

**Слабые стороны:**

1. Не гарантирует оптимальное решение: Случайный перебор не гарантирует нахождения наилучшего решения, он может найти только «хорошее» решение.
2. Зависимость от случайности: Качество решения зависит от случайного выбора, что может привести к непостоянству результатов.
3. Трудность выбора подходящей эвристики: Для эффективного случайного перебора необходимы хорошо подобранные эвристики, которые могут быть сложными в разработке.[2]

3.**Метод ветвей и границ**:

Метод ветвей и границ предложен в 1963 году группой авторов Дж. Литлом, К. Мурти, Д. Суини, К. Кэролом. Широко используемый вариант поиска с возвращением, фактически является лишь специальным частным случаем метода поиска с ограничениями. Ограничения в данном случае основываются на предположении, что на множестве возможных и частичных решений задана некоторая функция цены и что нужно найти оптимальное решение, т.е. решение с наименьшей ценой. Для применения метода ветвей и границ функция цены должна обладать тем свойством, что цена любого частичного решения не превышает цены любого расширения этого частичного решения (Заметим, что в большинстве случаев функция цены неотрицательна и даже удовлетворяет более сильному требованию).

Столь большой успех применения данного метода объясняется тем, что авторы первыми обратили внимание на широту возможностей метода, отметили важность использования специфики задачи и сами воспользовались спецификой задачи коммивояжера.

В основе метода ветвей и границ лежит идея последовательного разбиения множества допустимых решений на подмножества. На каждом шаге метода элементы разбиения подвергаются проверке для выяснения, содержит данное подмножество оптимальное решение или нет. Проверка осуществляется посредством вычисления оценки снизу для целевой функции на данном подмножестве. Если оценка снизу не меньше рекорда — наилучшего из найденных решений, то подмножество может быть отброшено. Проверяемое подмножество может быть отброшено еще и в том случае, когда в нем удается найти наилучшее решение. Если значение целевой функции на найденном решении меньше рекорда, то происходит смена рекорда. По окончании работы алгоритма рекорд является результатом его работы.

Если удается отбросить все элементы разбиения, то рекорд — оптимальное решение задачи. В противном случае, из неотброшенных подмножеств выбирается наиболее перспективное (например, с наименьшим значением нижней оценки), и оно подвергается разбиению. Новые подмножества вновь подвергаются проверке и т.д.[3]

**Анализ:**

**Сильные стороны:**

1. Гарантированное нахождение оптимального решения (при правильной реализации): Метод гарантирует нахождение оптимального решения, если оно существует, при условии правильного определения нижней оценки.
2. Эффективность: Метод отсекает целые ветви пространства поиска, которые не могут содержать оптимальное решение, что существенно сокращает количество рассматриваемых вариантов.
3. Гибкость: Метод можно адаптировать к различным задачам оптимизации, используя соответствующие функции оценки.

**Слабые стороны:**

1. Сложность реализации: Метод ветвей и границ может быть сложным в реализации, так как требует тщательного выбора функции оценки и стратегии ветвления.
2. Зависимость от функции оценки: Эффективность метода сильно зависит от качества функции оценки. Неточная или слишком слабая оценка может привести к рассмотрению большого количества лишних вариантов.
3. Потенциально большой объем памяти: Метод может требовать значительного объема памяти для хранения информации о ветвях и оценках.

**Вывод:**

Таким образом, для задач с небольшим пространством поиска, где требуется гарантированно найти наилучшее решение, полный перебор может быть вполне приемлемым. Если же пространство поиска обширно и допустимо довольствоваться "хорошим" решением, то случайный перебор, особенно в сочетании с эффективными эвристиками, станет предпочтительным выбором. Для сложных задач оптимизации, требующих оптимального решения и допускающих хорошую функцию оценки, метод ветвей и границ представляет собой наиболее эффективный подход, несмотря на сложность его реализации.

# **2. РАЗРАБОТКА КОДА НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ С++**

**Реализация:**

1) GraphClass: Класс, представляющий граф для задачи о коммивояжёре. Он содержит методы для вставки вершин, рёбер, удаления вершин и рёбер, а также поиска кратчайшего пути.

2) MyForm: Форма, содержащая графический интерфейс программы. В ней есть кнопки для добавления и удаления вершин и рёбер, а также для поиска кратчайшего пути.

3) draw\_circle, draw\_line, draw\_text: Функции для рисования графических элементов (кругов, линий, текста) с использованием библиотек OpenGL и GLUT.

**Описание работы программы:**

Пользователь запускает программу и видит графический интерфейс с кнопками и текстовыми полями, для ввода данных, которые передаются определённому методу в качестве аргументов.

Пользователь может добавлять вершины с помощью кнопки "Добавить вершину", которая вызывает метод insert\_vertex класса GraphClass. При этом каждая вершина имеет свой уникальный номер. Вершины равномерно распределяются по окружности.

Пользователь также может удалять вершины с помощью кнопки "Удалить вершину", которая вызывает метод delete\_vertex класса GraphClass. При этом удалится последняя добавленная вершина.

Для добавления ребра пользователь вводит номера начальной и конечной вершин, а также их вес (длину пути), и нажимает кнопку "Добавить ребро", которая вызывает метод insert\_edge класса GraphClass.

Для удаления ребра пользователь вводит номера начальной и конечной вершин и нажимает кнопку "Удалить ребро", которая вызывает метод delete\_edge класса GraphClass.

Для нахождения кратчайшего пути пользователь вводит номера начальной и конечной вершин и нажимает кнопку "Найти кратчайшее расстояние", которая вызывает метод shortest path класса GraphClass. Данный метод ищет кратчайшее расстояние между двумя вершинами методом ветвей и границ.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения лабораторной работы был разработан программный код, реализующий алгоритм решения задачи коммивояжера. Код успешно прошел тестирование и продемонстрировал корректную работу, находя оптимальный маршрут минимальной длины. Для наглядного представления решения была реализована визуализация с использованием графической библиотеки OpenGL, что позволило детально изучить найденный маршрут, проанализировать его эффективность и выявить возможные оптимизации.

Разработанное программное обеспечение с успехом решает поставленную задачу и может быть использовано для анализа и оптимизации логистических цепочек и транспортных сетей.

**ПРИЛОЖЕНИЕ А**

**(Полный код программы)**

*Graph.h*

#include <vector>

#include <iostream>

#include <queue>

#include <stack>

#include <limits>

#include <cmath>

#include <GL\glew.h>

#include <GL\freeglut.h>

#include <string>

using namespace std;

void draw\_circle(double x, double y, double radius) {

const int num\_segments = 100;

glBegin(GL\_TRIANGLE\_FAN);

glColor3f(0.7f, 0.7f, 0.7f);

glVertex2d(x, y);

for (int i = 0; i <= num\_segments; ++i) {

double angle = 2.0 \* 3.1415 \* i / num\_segments;

double dx = radius \* cos(angle);

double dy = radius \* sin(angle);

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

glVertex2d(x + dx, y + dy);

}

glEnd();

}

void draw\_line(double x1, double y1, double x2, double y2) {

glBegin(GL\_LINES);

glVertex2d(x1, y1);

glVertex2d(x2, y2);

glEnd();

}

void draw\_text(double x, double y, const string& text, int size) {

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

glRasterPos2d(x, y);

for (char c : text) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, c);

}

}

class GraphClass {

private:

vector<int> vertex\_list;

vector<vector<int>> adjacency\_matrix;

int max\_size;

int vertex\_count;

public:

GraphClass(int size) : max\_size(size), vertex\_count(0) {

adjacency\_matrix = vector<vector<int>>(size, vector<int>(size, 0));

}

bool is\_full() {

return vertex\_count == max\_size;

}

bool is\_empty() {

return vertex\_count == 0;

}

void insert\_vertex(int vertex) {

if (!is\_full()) {

cout << "\"+\" Вершина " << vertex << " добавлена!\n";

vertex\_list.push\_back(vertex);

vertex\_count++;

}

else {

cout << "Невозможно добавить вершину: достигнуто максимальное количество вершин\n";

}

}

void delete\_vertex(int vertex) {

if (!is\_empty()) {

cout << "\"-\" Вершина " << vertex << " удалена!\n";

vertex\_list.pop\_back();

vertex\_count--;

}

else {

cout << "Невозможно удалить вершину: граф пуст\n";

}

}

int get\_vertex\_position(int vertex) {

for (int i = 0; i < vertex\_count; i++) {

if (vertex\_list[i] == vertex) {

return i;

}

}

return -1;

}

int get\_amount\_vertex() {

return vertex\_count;

}

int get\_weight(int v1, int v2) {

int v1\_pos = get\_vertex\_position(v1);

int v2\_pos = get\_vertex\_position(v2);

if (v1\_pos == -1 || v2\_pos == -1) {

cout << "Одной из узлов в графе не существует\n";

return 0;

}

return adjacency\_matrix[v1\_pos][v2\_pos];

}

vector<int> get\_neighbours(int vertex) {

vector<int> neighbours\_list;

int pos = get\_vertex\_position(vertex);

if (pos != -1) {

for (int i = 0; i < vertex\_count; ++i) {

if (adjacency\_matrix[pos][i] != 0) {

neighbours\_list.push\_back(vertex\_list[i]);

}

}

}

return neighbours\_list;

}

void insert\_edge(int vertex\_1, int vertex\_2, int weight = 1) {

int vertex\_pos\_1 = get\_vertex\_position(vertex\_1);

int vertex\_pos\_2 = get\_vertex\_position(vertex\_2);

if (vertex\_pos\_1 == -1 || vertex\_pos\_2 == -1) {

cout << "Одной из вершин в графе не существует\n";

return;

}

if (adjacency\_matrix[vertex\_pos\_1][vertex\_pos\_2] == 0 && adjacency\_matrix[vertex\_pos\_2][vertex\_pos\_1] == 0) {

adjacency\_matrix[vertex\_pos\_1][vertex\_pos\_2] = weight;

adjacency\_matrix[vertex\_pos\_2][vertex\_pos\_1] = weight;

std::cout << "\"+\" Ребро от вершины " << vertex\_1 << " к вершине " << vertex\_2 << " добавлено!\n";

}

else {

cout << "Ребро между вершинами уже есть\n";

}

}

void delete\_edge(int vertex\_1, int vertex\_2) {

int pos\_1 = get\_vertex\_position(vertex\_1);

int pos\_2 = get\_vertex\_position(vertex\_2);

if (pos\_1 == -1 || pos\_2 == -1) {

cout << "Одной из вершин в графе не существует\n";

return;

}

if (adjacency\_matrix[pos\_1][pos\_2] == 0 || adjacency\_matrix[pos\_2][pos\_1] == 0) {

cout << "Ребро между вершинами отсутствует\n";

}

else {

adjacency\_matrix[pos\_1][pos\_2] = 0;

adjacency\_matrix[pos\_2][pos\_1] = 0;

cout << "Ребро между вершинами " << vertex\_1 << " и " << vertex\_2 << " удалено\n";

}

}

void print() {

if (!is\_empty()) {

cout << "Матрица смежности графа:\n";

cout << " ";

for (int i = 0; i < vertex\_count; i++) {

cout << " " << vertex\_list[i] << " ";

}

cout << endl;

for (int i = 0; i < vertex\_count; i++) {

cout << " " << vertex\_list[i] << " ";

for (int j = 0; j < vertex\_count; j++) {

cout << " " << adjacency\_matrix[i][j] << " ";

}

cout << endl;

}

}

else {

cout << "Граф пуст\n";

}

}

int get\_amount\_edges() {

int amount = 0;

for (int i = 0; i < vertex\_count; i++) {

for (int j = i; j < vertex\_count; j++) {

if (adjacency\_matrix[i][j] != 0) {

amount++;

}

}

}

return amount;

}

void DFS(int start\_vertex, bool\* visited\_vertex) {

cout << "Вершина " << start\_vertex << " пройдена\n";

visited\_vertex[get\_vertex\_position(start\_vertex)] = true;

vector<int> neighbours = get\_neighbours(start\_vertex);

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

if (!visited\_vertex[get\_vertex\_position(neighbours[i])]) {

DFS(neighbours[i], visited\_vertex);

}

}

}

void BFS(int start\_vertex, bool\* visited\_vertex) {

queue<int> queue;

visited\_vertex[get\_vertex\_position(start\_vertex)] = true;

queue.push(start\_vertex);

while (!queue.empty()) {

int current\_vertex = queue.front();

queue.pop();

cout << "Вершина " << current\_vertex << " пройдена\n";

vector<int> neighbours = get\_neighbours(current\_vertex);

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

int neighbour\_vertex = neighbours[i];

int neighbour\_pos = get\_vertex\_position(neighbour\_vertex);

if (!visited\_vertex[neighbour\_pos]) {

visited\_vertex[neighbour\_pos] = true;

queue.push(neighbour\_vertex);

}

}

}

}

int shortest\_path(int start\_vertex, int end\_vertex) {

vector<int> distance(vertex\_count, numeric\_limits<int>::max());

distance[get\_vertex\_position(start\_vertex)] = 0;

vector<int> previous(vertex\_count, -1);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

pq.push({ 0, start\_vertex });

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

vector<int> neighbours = get\_neighbours(u);

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

int v = neighbours[i];

int weight = get\_weight(u, v);

if (distance[get\_vertex\_position(v)] > distance[get\_vertex\_position(u)] + weight) {

distance[get\_vertex\_position(v)] = distance[get\_vertex\_position(u)] + weight;

previous[get\_vertex\_position(v)] = u;

pq.push({ distance[get\_vertex\_position(v)], v });

}

}

}

cout << "Кратчайший путь от " << start\_vertex << " к " << end\_vertex << " проходит через следующие вершины ";

stack<int> path;

int current\_vertex = end\_vertex;

while (current\_vertex != -1) {

path.push(current\_vertex);

current\_vertex = previous[get\_vertex\_position(current\_vertex)];

}

while (!path.empty()) {

if (path.size() == 1) {

cout << path.top();

}

else {

cout << path.top() << "->";

}

path.pop();

}

cout << " и равен " << distance[get\_vertex\_position(end\_vertex)];

cout << endl;

return distance[get\_vertex\_position(end\_vertex)];

}

void draw\_graph() const {

double angle = 2 \* 3.1415 / vertex\_count;

// Draw vertices

double min\_dimension = min(800.0, 600.0);

double radius = min\_dimension / 3;

for (int i = 0; i < vertex\_count; ++i) {

double x = 400 + radius \* cos(i \* angle);

double y = 300 + radius \* sin(i \* angle);

draw\_circle(x, y, 30);

draw\_text(x - 5, y + 5, to\_string(vertex\_list[i]), 16);

}

// Draw edges

for (int i = 0; i < vertex\_count; ++i) {

for (int j = i + 1; j < vertex\_count; ++j) {

if (adjacency\_matrix[i][j] != 0) {

double mid\_x = (400 + radius \* cos(i \* angle) + 400 + radius \* cos(j \* angle)) / 2;

double mid\_y = (300 + radius \* sin(i \* angle) + 300 + radius \* sin(j \* angle)) / 2;

draw\_line(400 + radius \* cos(i \* angle), 300 + radius \* sin(i \* angle),

400 + radius \* cos(j \* angle), 300 + radius \* sin(j \* angle));

// Draw text showing edge weight

draw\_text(mid\_x - 5, mid\_y + 5, to\_string(adjacency\_matrix[i][j]), 12);

}

}

}

}

};

*MyForm.h*

#pragma once

GraphClass graph\_1(15);

int serial\_number = 1;

void display() {

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

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

graph\_1.draw\_graph();

glFlush();

glutSwapBuffers();

}

namespace Graph {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void) {

InitializeComponent();

}

protected:

~MyForm() {

if (components) {

delete components;

}

}

private: System::Windows::Forms::Label^ label\_weight;

protected:

protected:

private: System::Windows::Forms::Button^ add\_vertex;

private: System::Windows::Forms::Button^ add\_edge;

private: System::Windows::Forms::TextBox^ text\_box\_from;

private: System::Windows::Forms::Button^ find\_way;

private: System::Windows::Forms::Button^ delete\_vertex;

private: System::Windows::Forms::Button^ delete\_edge;

private: System::Windows::Forms::Label^ label\_from;

private: System::Windows::Forms::TextBox^ text\_box\_to;

private: System::Windows::Forms::Label^ label\_to;

private: System::Windows::Forms::TextBox^ text\_box\_weight;

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

void InitializeComponent(void) {

this->label\_weight = (gcnew System::Windows::Forms::Label());

this->add\_vertex = (gcnew System::Windows::Forms::Button());

this->add\_edge = (gcnew System::Windows::Forms::Button());

this->text\_box\_from = (gcnew System::Windows::Forms::TextBox());

this->find\_way = (gcnew System::Windows::Forms::Button());

this->delete\_vertex = (gcnew System::Windows::Forms::Button());

this->delete\_edge = (gcnew System::Windows::Forms::Button());

this->label\_from = (gcnew System::Windows::Forms::Label());

this->text\_box\_to = (gcnew System::Windows::Forms::TextBox());

this->label\_to = (gcnew System::Windows::Forms::Label());

this->text\_box\_weight = (gcnew System::Windows::Forms::TextBox());

this->SuspendLayout();

//

// label\_weight

//

this->label\_weight->AutoSize = true;

this->label\_weight->Cursor = System::Windows::Forms::Cursors::Default;

this->label\_weight->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->label\_weight->Font = (gcnew System::Drawing::Font(L"Calibri", 19, System::Drawing::FontStyle::Italic));

this->label\_weight->Location = System::Drawing::Point(171, 23);

this->label\_weight->Name = L"label\_weight";

this->label\_weight->Size = System::Drawing::Size(122, 32);

this->label\_weight->TabIndex = 5;

this->label\_weight->Text = L"Вес ребра";

this->label\_weight->TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

//

// add\_vertex

//

this->add\_vertex->BackColor = System::Drawing::Color::LightGray;

this->add\_vertex->Cursor = System::Windows::Forms::Cursors::Hand;

this->add\_vertex->FlatAppearance->BorderColor = System::Drawing::Color::Black;

this->add\_vertex->FlatAppearance->BorderSize = 2;

this->add\_vertex->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->add\_vertex->Font = (gcnew System::Drawing::Font(L"Calibri", 19, static\_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold | System::Drawing::FontStyle::Italic)),

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

this->add\_vertex->Location = System::Drawing::Point(76, 131);

this->add\_vertex->Name = L"add\_vertex";

this->add\_vertex->Size = System::Drawing::Size(300, 100);

this->add\_vertex->TabIndex = 6;

this->add\_vertex->Text = L"Добавить вершину";

this->add\_vertex->UseVisualStyleBackColor = false;

this->add\_vertex->Click += gcnew System::EventHandler(this, &MyForm::add\_vertex\_Click);

//

// add\_edge

//

this->add\_edge->BackColor = System::Drawing::Color::LightGray;

this->add\_edge->Cursor = System::Windows::Forms::Cursors::Hand;

this->add\_edge->FlatAppearance->BorderColor = System::Drawing::Color::Black;

this->add\_edge->FlatAppearance->BorderSize = 2;

this->add\_edge->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->add\_edge->Font = (gcnew System::Drawing::Font(L"Calibri", 19, static\_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold | System::Drawing::FontStyle::Italic)),

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

this->add\_edge->Location = System::Drawing::Point(76, 364);

this->add\_edge->Name = L"add\_edge";

this->add\_edge->Size = System::Drawing::Size(300, 100);

this->add\_edge->TabIndex = 7;

this->add\_edge->Text = L"Добавить ребро";

this->add\_edge->UseVisualStyleBackColor = false;

this->add\_edge->Click += gcnew System::EventHandler(this, &MyForm::add\_edge\_Click);

//

// text\_box\_from

//

this->text\_box\_from->BackColor = System::Drawing::Color::LightGray;

this->text\_box\_from->BorderStyle = System::Windows::Forms::BorderStyle::FixedSingle;

this->text\_box\_from->Font = (gcnew System::Drawing::Font(L"Calibri", 19, System::Drawing::FontStyle::Italic, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->text\_box\_from->Location = System::Drawing::Point(12, 58);

this->text\_box\_from->Name = L"text\_box\_from";

this->text\_box\_from->Size = System::Drawing::Size(102, 38);

this->text\_box\_from->TabIndex = 8;

this->text\_box\_from->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// find\_way

//

this->find\_way->BackColor = System::Drawing::Color::LightGray;

this->find\_way->Cursor = System::Windows::Forms::Cursors::Hand;

this->find\_way->FlatAppearance->BorderColor = System::Drawing::Color::Black;

this->find\_way->FlatAppearance->BorderSize = 2;

this->find\_way->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->find\_way->Font = (gcnew System::Drawing::Font(L"Calibri", 19, static\_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold | System::Drawing::FontStyle::Italic)),

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

this->find\_way->Location = System::Drawing::Point(76, 598);

this->find\_way->Name = L"find\_way";

this->find\_way->Size = System::Drawing::Size(300, 100);

this->find\_way->TabIndex = 9;

this->find\_way->Text = L"Найти кратчайшее растояние";

this->find\_way->UseVisualStyleBackColor = false;

this->find\_way->Click += gcnew System::EventHandler(this, &MyForm::find\_way\_Click);

//

// delete\_vertex

//

this->delete\_vertex->BackColor = System::Drawing::Color::LightGray;

this->delete\_vertex->Cursor = System::Windows::Forms::Cursors::Hand;

this->delete\_vertex->FlatAppearance->BorderColor = System::Drawing::Color::Black;

this->delete\_vertex->FlatAppearance->BorderSize = 2;

this->delete\_vertex->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->delete\_vertex->Font = (gcnew System::Drawing::Font(L"Calibri", 19, static\_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold | System::Drawing::FontStyle::Italic)),

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

this->delete\_vertex->Location = System::Drawing::Point(76, 249);

this->delete\_vertex->Name = L"delete\_vertex";

this->delete\_vertex->Size = System::Drawing::Size(300, 100);

this->delete\_vertex->TabIndex = 10;

this->delete\_vertex->Text = L"Удалить вершину";

this->delete\_vertex->UseVisualStyleBackColor = false;

this->delete\_vertex->Click += gcnew System::EventHandler(this, &MyForm::delete\_vertex\_Click);

//

// delete\_edge

//

this->delete\_edge->BackColor = System::Drawing::Color::LightGray;

this->delete\_edge->Cursor = System::Windows::Forms::Cursors::Hand;

this->delete\_edge->FlatAppearance->BorderColor = System::Drawing::Color::Black;

this->delete\_edge->FlatAppearance->BorderSize = 2;

this->delete\_edge->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->delete\_edge->Font = (gcnew System::Drawing::Font(L"Calibri", 19, static\_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold | System::Drawing::FontStyle::Italic)),

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

this->delete\_edge->Location = System::Drawing::Point(76, 481);

this->delete\_edge->Name = L"delete\_edge";

this->delete\_edge->Size = System::Drawing::Size(300, 100);

this->delete\_edge->TabIndex = 11;

this->delete\_edge->Text = L"Удалить ребро";

this->delete\_edge->UseVisualStyleBackColor = false;

this->delete\_edge->Click += gcnew System::EventHandler(this, &MyForm::delete\_edge\_Click);

//

// label\_from

//

this->label\_from->AutoSize = true;

this->label\_from->Cursor = System::Windows::Forms::Cursors::Default;

this->label\_from->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->label\_from->Font = (gcnew System::Drawing::Font(L"Calibri", 19, System::Drawing::FontStyle::Italic));

this->label\_from->Location = System::Drawing::Point(35, 23);

this->label\_from->Name = L"label\_from";

this->label\_from->Size = System::Drawing::Size(52, 32);

this->label\_from->TabIndex = 12;

this->label\_from->Text = L"От";

this->label\_from->TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

//

// text\_box\_to

//

this->text\_box\_to->BackColor = System::Drawing::Color::LightGray;

this->text\_box\_to->BorderStyle = System::Windows::Forms::BorderStyle::FixedSingle;

this->text\_box\_to->Font = (gcnew System::Drawing::Font(L"Calibri", 19, System::Drawing::FontStyle::Italic, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->text\_box\_to->Location = System::Drawing::Point(345, 58);

this->text\_box\_to->Name = L"text\_box\_to";

this->text\_box\_to->Size = System::Drawing::Size(102, 38);

this->text\_box\_to->TabIndex = 13;

this->text\_box\_to->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// label\_to

//

this->label\_to->AutoSize = true;

this->label\_to->Cursor = System::Windows::Forms::Cursors::Default;

this->label\_to->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->label\_to->Font = (gcnew System::Drawing::Font(L"Calibri", 19, System::Drawing::FontStyle::Italic));

this->label\_to->Location = System::Drawing::Point(379, 23);

this->label\_to->Name = L"label\_to";

this->label\_to->Size = System::Drawing::Size(28, 32);

this->label\_to->TabIndex = 14;

this->label\_to->Text = L"К";

this->label\_to->TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

//

// text\_box\_weight

//

this->text\_box\_weight->BackColor = System::Drawing::Color::LightGray;

this->text\_box\_weight->BorderStyle = System::Windows::Forms::BorderStyle::FixedSingle;

this->text\_box\_weight->Font = (gcnew System::Drawing::Font(L"Calibri", 19, System::Drawing::FontStyle::Italic, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->text\_box\_weight->Location = System::Drawing::Point(177, 58);

this->text\_box\_weight->Name = L"text\_box\_weight";

this->text\_box\_weight->Size = System::Drawing::Size(102, 38);

this->text\_box\_weight->TabIndex = 15;

this->text\_box\_weight->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::Color::MediumPurple;

this->ClientSize = System::Drawing::Size(459, 722);

this->Controls->Add(this->text\_box\_weight);

this->Controls->Add(this->label\_to);

this->Controls->Add(this->text\_box\_to);

this->Controls->Add(this->label\_from);

this->Controls->Add(this->delete\_edge);

this->Controls->Add(this->delete\_vertex);

this->Controls->Add(this->find\_way);

this->Controls->Add(this->text\_box\_from);

this->Controls->Add(this->add\_edge);

this->Controls->Add(this->add\_vertex);

this->Controls->Add(this->label\_weight);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedDialog;

this->Name = L"MyForm";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"Graph";

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private:

System::Void add\_vertex\_Click(System::Object^ sender, System::EventArgs^ e) {

graph\_1.insert\_vertex(serial\_number);

serial\_number++;

}

private:

System::Void delete\_vertex\_Click(System::Object^ sender, System::EventArgs^ e) {

graph\_1.delete\_vertex(serial\_number - 1);

serial\_number--;

}

private:

System::Void add\_edge\_Click(System::Object^ sender, System::EventArgs^ e) {

if (this->text\_box\_from->Text != "" && this->text\_box\_to->Text != "" && this->text\_box\_weight->Text != "") {

try {

unsigned int value\_1 = Convert::ToUInt32(this->text\_box\_from->Text);

unsigned int value\_2 = Convert::ToUInt32(this->text\_box\_to->Text);

unsigned int value\_3 = Convert::ToUInt32(this->text\_box\_weight->Text);

graph\_1.insert\_edge(value\_1, value\_2, value\_3);

}

catch (FormatException^) {

MessageBox::Show(this, "Введенное значение не является натуральным числом!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (OverflowException^) {

MessageBox::Show(this, "Введенное значение не является натуральным числом!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

this->text\_box\_from->Text = "";

this->text\_box\_to->Text = "";

this->text\_box\_weight->Text = "";

}

else {

cout << "Введите значения!\n";

}

}

private:

System::Void delete\_edge\_Click(System::Object^ sender, System::EventArgs^ e) {

if (this->text\_box\_from->Text != "" && this->text\_box\_to->Text != "") {

try {

unsigned int value\_1 = Convert::ToUInt32(this->text\_box\_from->Text);

unsigned int value\_2 = Convert::ToUInt32(this->text\_box\_to->Text);

graph\_1.delete\_edge(value\_1, value\_2);

}

catch (FormatException^) {

MessageBox::Show(this, "Введенное значение не является натуральным числом!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (OverflowException^) {

MessageBox::Show(this, "Введенное значение не является натуральным числом!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

this->text\_box\_from->Text = "";

this->text\_box\_to->Text = "";

this->text\_box\_weight->Text = "";

}

else {

cout << "Введите значения!\n";

}

}

private:

System::Void find\_way\_Click(System::Object^ sender, System::EventArgs^ e) {

if (this->text\_box\_from->Text != "" && this->text\_box\_to->Text != "") {

try {

unsigned int value\_1 = Convert::ToUInt32(this->text\_box\_from->Text);

unsigned int value\_2 = Convert::ToUInt32(this->text\_box\_to->Text);

graph\_1.shortest\_path(value\_1, value\_2);

}

catch (FormatException^) {

MessageBox::Show(this, "Введенное значение не является натуральным числом!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (OverflowException^) {

MessageBox::Show(this, "Введенное значение не является натуральным числом!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

this->text\_box\_from->Text = "";

this->text\_box\_to->Text = "";

this->text\_box\_weight->Text = "";

}

else {

cout << "Введите значения!\n";

}

}

};

}

*MyForm.cpp*

#include "Graph.h"

#include "MyForm.h"

#include <thread>

using namespace System;

using namespace Graph;

using namespace System::Windows::Forms;

void runOpenGL() {

int argc = 0;

char\* argv = nullptr;

glutInit(&argc, &argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);

glutInitWindowSize(800, 600);

glutCreateWindow("Graph Visualization");

glutDisplayFunc(display);

glOrtho(0.0, 800.0, 600.0, 0.0, -1.0, 1.0);

glutMainLoop();

}

void runWindowsForms() {

Application::Run(gcnew MyForm());

}

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

setlocale(LC\_ALL, "RU");

// Запускаем GLUT в отдельном потоке

std::thread glut\_thread(runOpenGL);

glut\_thread.detach(); // Отсоединяем поток GLUT

// Запускаем Windows Forms в основном потоке

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

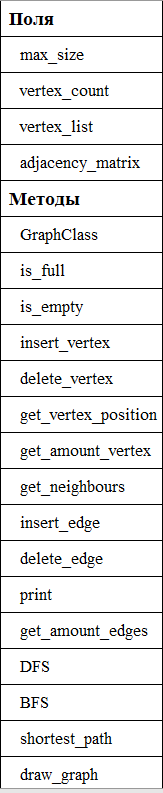
runWindowsForms();

return 0;

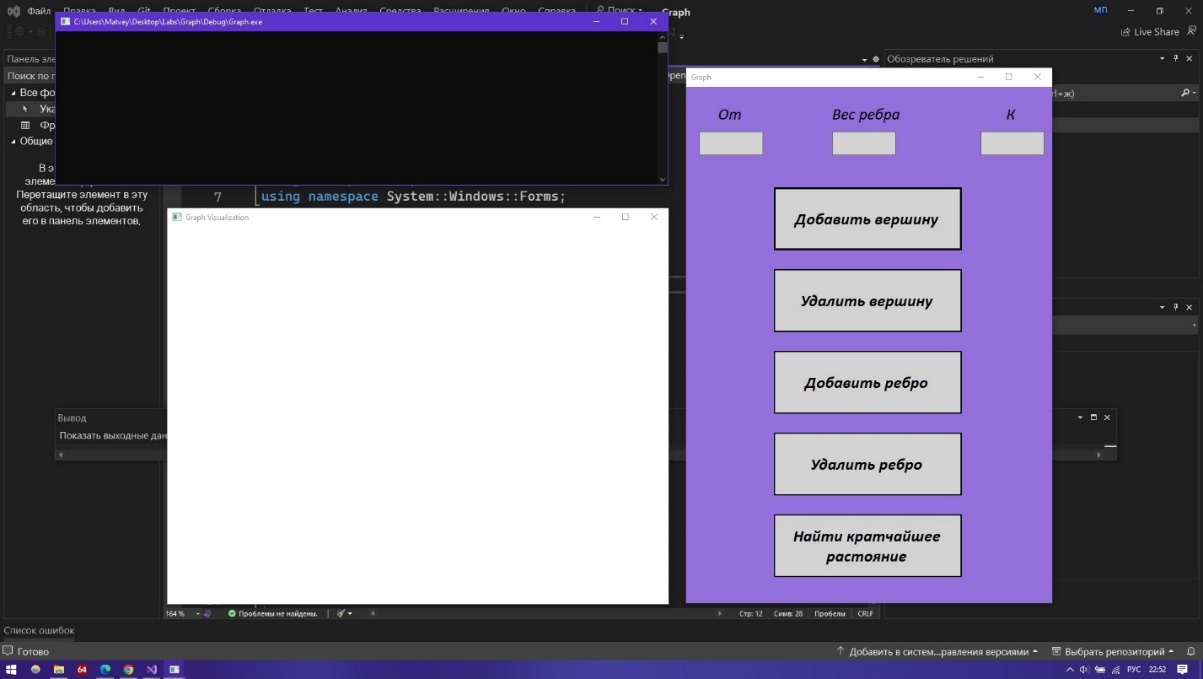
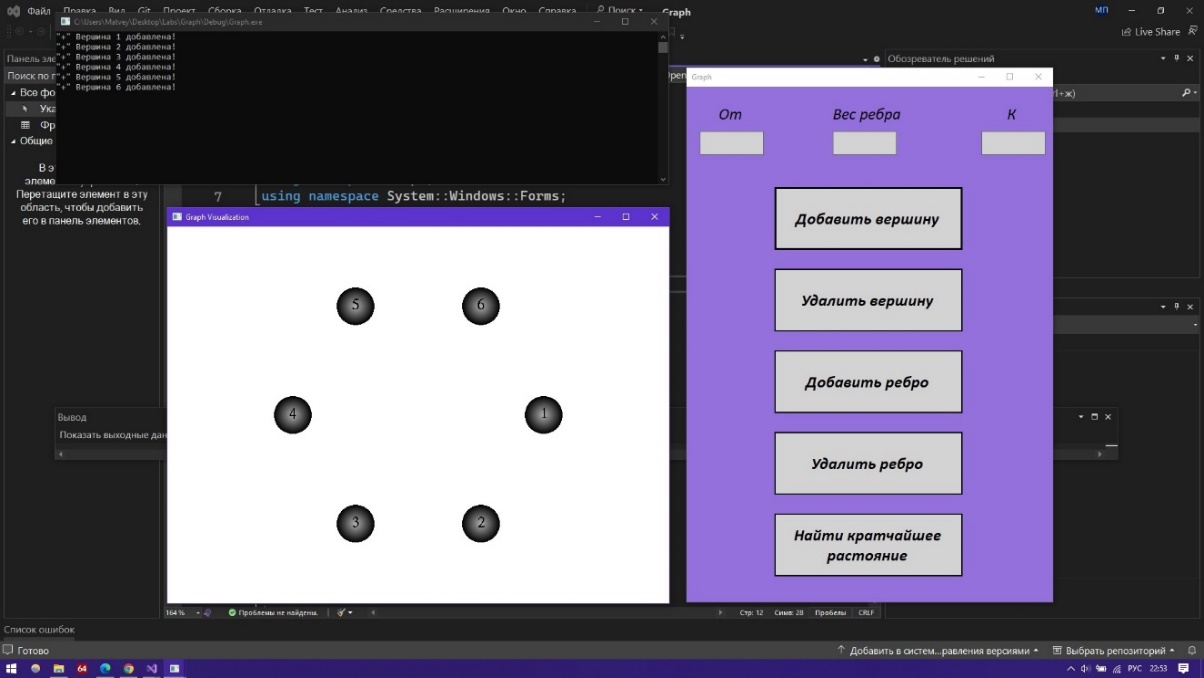
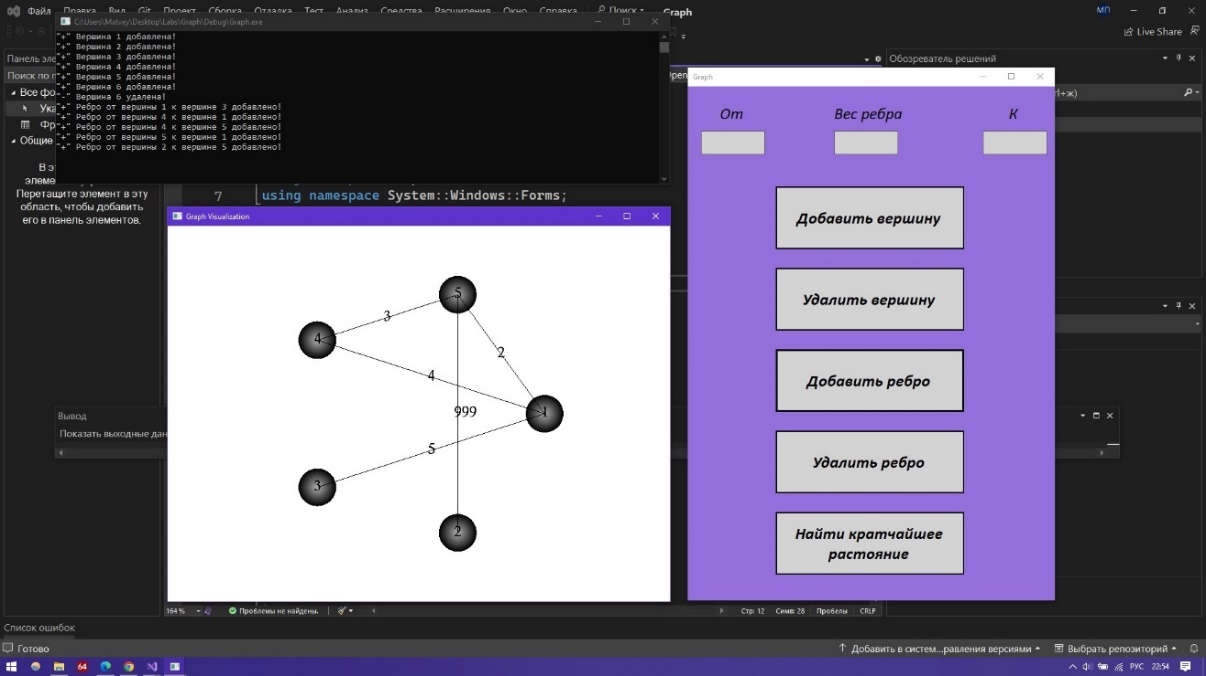
}

**ПРИЛОЖЕНИЕ Б**

**(UML)**



**ПРИЛОЖЕНИЕ В**

**(Визуализация)**

