Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(ФГАОУ ВО ПНИПУ)

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ

По дисциплине «Теория алгоритмов и структуры данных» на тему:

РЕШЕНИЕ ЗАДАЧИ КОММИВОЯЖЕРА МЕТОДОМ ВЕТВЕЙ И ГРАНИЦ

Выполнили

студентки группы РИС-23-2б

Жилина А.А.

Куклина Ю.В.

Проверила доцент кафедры ИТАС

Полякова О.А.

Пермь 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc166839825)

[1 Анализ алгоритма решения задачи Коммивояжера 4](#_Toc166839826)

[2 Разработка программы 7](#_Toc166839827)

[2.1 Постановка задачи 7](#_Toc166839828)

[2.2 Анализ задачи 7](#_Toc166839829)

[2.3 Разработка программы 9](#_Toc166839830)

[Заключение 10](#_Toc166839831)

[Список использованной литературы 11](#_Toc166839832)

[Приложение А 12](#_Toc166839833)

[Приложение Б 13](#_Toc166839834)

[Приложение В 28](#_Toc166839835)

# ВВЕДЕНИЕ

Задача коммивояжера – одна из классических задач комбинаторной оптимизации, которая имеет огромное практическое значение в различных областях, связанных с логистикой, транспортировкой и планированием маршрутов. Поставленная задача заключается в нахождении оптимального маршрута, проходящего через заданный набор городов (точек) таким образом, чтобы общая длина маршрута была минимальной.

В данном отчете будут рассмотрены основные аспекты задачи коммивояжера, такие как математическая постановка задачи, метод ветвей и границ для ее решения, применяемые алгоритмы и инструменты оптимизации.

Целью лабораторной работы является разработка программы для решения задачи Коммивояжера.

Задачи:

1. Проанализировать алгоритм решения задачи Коммивояжера методом ветвей и границ;
2. Разработать программу, решающую задачу Коммивояжера для данного графа.

# 1 Анализ алгоритма решения задачи Коммивояжера

Для решения задачи коммивояжера ее надо представить как математическую модель. При этом исходные условия можно записать в формате матрицы – таблицы, где строкам соответствуют города отправления, столбцам – города прибытия, а в ячейках указываются расстояния (время, стоимость) между ними; или в виде графа – схемы, состоящей из вершин (точек, кружков), которые символизируют города, и соединяющих их ребер (линий), длина которых соответствует расстоянию между городами [1].

Для решения задачи коммивояжера методом ветвей и границ необходимо выполнить следующий алгоритм (последовательность действий):

* Построение матрицы с исходными данными – в таблицу заносятся расстояния (Cij) между городами (в ячейки типа A-A, B-B и т. д. ставится символ M – условно бесконечно большое число); при этом строкам соответствуют города отбытия, а столбцам города прибытия;
* Нахождение минимумов по строкам – в каждой строке определяется минимальное число (di) и выписывается в отдельный столбец; Редукция строк – из значений ячеек каждой строки вычитаем соответствующий минимум (Cij = Cij – di), не затрагивая при этом клетки с M [2];
* Нахождение минимумов по столбцам – в каждом столбце определяется минимальное число (dj) и выписывается в отдельную строку; Редукция столбцов – из значений ячеек каждого столбца вычитаем соответствующий минимум (Cij = Cij – dj), не затрагивая при этом клетки с M;
* Нахождение корневой нижней границы (делаем это только один раз, в следующие разы пункт 6 пропускаем) – вычисляем нижнюю границу (минимально возможную на текущем этапе длину маршрута) в стартовой (корневой) точке решения, как сумму найденных ранее минимумов (H0 = ∑di + ∑dj) и начинаем построение графа (схемы) решения с внесения в него корневой вершины; Вычисление оценок нулевых клеток – считаем оценки (pij) для каждой ячейки с нулями, как сумму минимумов по строке и столбцу, в которых располагается нулевая клетка, не учитывая при этом саму нулевую клетку;
* Выбор нулевой клетки с максимальной оценкой – ищем среди нулевых клеток обладающую наибольшей оценкой (если таких ячеек несколько, выбираем любую), и получаем пару ветвей (вариантов) решения задачи: с включением в маршрут отрезка пути относящегося к выбранной ячейке и без включения [3];
* Редукция матрицы – вычеркиваем относящиеся к выбранной клетке строку и столбец, а также заменяем значение ячейки соответствующей обратному пути на M;
* Вычисление нижней границы первой ветви (включающей отрезок пути) – вновь находим минимумы по строкам, проводим редукцию строк, находим минимумы по столбцам, проводим редукцию столбцов, после чего вычисляем локальную нижнюю границу, как сумму предыдущей локальной нижней границы и минимумов (Hk = Hk-1 + ∑di + ∑dj), и добавляем вершину в граф;
* Вычисление нижней границы второй ветви (не включающей отрезок пути) – считаем локальную нижнюю границу, как сумму предыдущей локальной нижней границы и оценки выбранной ранее нулевой клетки (Hk\* = Hk-1 + pij), и добавляем вершину в граф; Выбор ветви с минимальным значением нижней границы – среди еще не ветвившихся вершин выбираем обладающую минимальным значением локальной нижней границы (вне зависимости от того, какую ветвь рассматриваем в данный момент);
* Если полный маршрут еще не найден, продолжаем решение, если найден – переходим к пункту 10 – если маршрут еще не найден, то ход дальнейшего решения зависит от выбранной ветви: (а) первая ветвь – переходим к пункту 7, (б) вторая ветвь – в клетку с максимальной оценкой ставим M и переходим к пункту 2, (в) другая ветвь – возвращаемся к соответствующим ей этапу решения и таблице данных;
* Построение полного маршрута и определение его длины – соединяем все найденные ранее отрезки пути в полный маршрут и считаем его общую длину (данные берем из исходной таблицы).

Таким образом, решение задачи Коммивояжера методом ветвей и границ является одним из эффективных решений.

# 2 Разработка программы

## 2.1 Постановка задачи

Дан набор городов и расстояния между ними. Требуется найти такой путь, который проходит через все города и возвращается в начальный город, при этом суммарное расстояние пути минимально.

1. В качестве варианта для демонстрации работы программы придумать свой граф, состоящий из 6 вершин.

2. Реализовать решение задачи Коммивояжера методом ветвей и границ, функции по добавлении новых узлов, удалению узлов, установки связей между узлами, разрыву связей. Представлено в ПРИЛОЖЕНИИ Б.

3. Разработать программу, которая будет универсальной на любом наборе исходных данных.

4. Проработать визуализирующую часть в программе средствами Windows Forms в части построения графа. Представлено в ПРИЛОЖЕНИИ В.

5. Исходные данные должны приниматься с консоли, либо через графический интерфейс с помощью Qt, Windows Forms или других фреймворков и библиотек в экосистеме языка C++.

6. Задокументировать программу диаграммой классов UML.

UML диаграммы представлены в ПРИЛОЖЕНИИ А.

## 2.2 Анализ задачи

Создание класса:

1. В классе *Graph* определены следующие поля данных:

* *UzelList* - вектор узлов графа,
* *RebroList* - вектор ребер графа,
* *matrixSmezhosty* - матрица смежности графа,
* *maxSize* - размер вектора вершин и матрицы смежности,
* *UzelsQueue* - очередь узлов,
* *labelList* - вектор расстояний.

1. В конструкторе *Graph* инициализируются переменные *maxSize* и *matrixSmezhosty*. Матрица смежности заполняется значениями -1.

Удаление узлов и ребер класса:

1. DeleteUzel(const int NameUzel): Этот метод удаляет вершину с заданным именем NameUzel из графа. Если граф не пустой, метод находит индекс удаляемой вершины в списке вершин UzelList. Затем метод проходит по всем строкам и столбцам матрицы смежности matrixSmezhosty и устанавливает значение -1 в ячейках, соответствующих удаляемой вершине. Наконец, метод также обновляет информацию о вершине в списке вершин, устанавливая для нее все значения равными -1.

2. DeleteRebro(const int& NameUzel1, const int& NameUzel2): Этот метод удаляет ребро между двумя вершинами с именами NameUzel1 и NameUzel2. Сначала метод находит индекс удаляемого ребра в списке ребер RebroList. Затем он обновляет информацию об удаляемом ребре, устанавливая все его значения равными -1. Далее метод проходит по всем элементам матрицы смежности и заменяет индексы удаляемого ребра на -1.

Добавление узлов и ребер класса:

1. InsertUzel(const T& nameUzel, const int x, const int y, TextBox^ textBox\_console): Этот метод добавляет новую вершину в граф. Сначала проверяется, не является ли граф уже полным с помощью метода IsFull(). Затем ищется первое пустое место в списке вершин UzelList и добавляется новая вершина в это место, сохраняя ее имя, координаты x и y. Если пустого места нет, добавляется новая вершина в конец списка.

2. lnsertRebro(const T& NameUzel1, const T& NameUzel2, const int& vecRebra, const int X, const int Y, TextBox^ textBox\_console): Этот метод добавляет новое ребро между двумя вершинами в графе. Сначала проверяется, существуют ли обе вершины в графе, и они либо обе указаны в графе, либо одна из них указана. Затем проверяется, не существует ли уже ребра между этими вершинами. Если ребра нет, метод ищет первое пустое место в списке ребер RebroList и добавляет новое ребро с данными о ребре и его координатах. После этого метод обновляет матрицу смежности для отражения нового ребра. Наконец, метод очищает временный вектор Temp.

Решение задачи Коммивояжера:

1. Инициализируется флаг *flag* и создается пустой вектор *ways* для хранения путей.
2. В цикле *do-while* выполняет ряд операций:
3. Находит минимальные значения для каждой строки матрицы *matrix.*
4. Производит редукцию строк матрицы *matrix* на основе минимальных значений.
5. Находит минимальные значения для каждого столбца редуцированной матрицы.
6. Производит редукцию столбцов на основе минимальных значений.
7. Находит индексы нулевой клетки с наибольшей оценкой в редуцированной матрице.
8. Производит редукцию матрицы на основе найденной нулевой клетки.
9. Если были найдены индексы нулевой клетки не равные 0, добавляет их в вектор *ways*.
10. Иначе, завершает работу алгоритма, формирует оптимальный путь *way*, вычисляет его длину и возвращает этот путь.
11. Цикл продолжается, пока флаг *flag* равен *true*.

## 2.3 Разработка программы

Код программы представлен в ПРИЛОЖЕНИИ Б. С работой программы можно ознакомиться в ПРИЛОЖЕНИИ В.

# ЗАКЛЮЧЕНИЕ

Задача коммивояжера является одной из знаменитых задач теории комбинаторики и пользуется популярностью благодаря тому, что к ней сводится большое количество практических задач.

Для достижения поставленной цели были решены следующие задачи:

1. Проанализирован алгоритм решения задачи Коммивояжера методом ветвей и границ;
2. Разработана программа, решающая задачу Коммивояжера для данного графа.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Галяутдинов Р.Р. Задача коммивояжера – метод ветвей и границ // Сайт преподавателя экономики. [2023]. URL: https://galyautdinov.ru/post/zadacha-kommivoyazhera (дата обращения: 16.05.2024).

2. Задачи коммивояжера [Электронный ресурс] // URL: http://window.edu.ru/window\_catalog/pdf2txt?p\_id=26518&p\_page=7 (дата обращения 16.05.2024).

3. Задача о коммивояжере [Электронный ресурс] // URL: http://mirslovarei.com/content\_biz/Zadacha-O-Kommivojazhere-4474.html (дата обращения 16.05.2024).

# ПРИЛОЖЕНИЕ А

**UML диаграммы**

Диаграммы классов и прецедентов представлены на рис. А1 и А2.

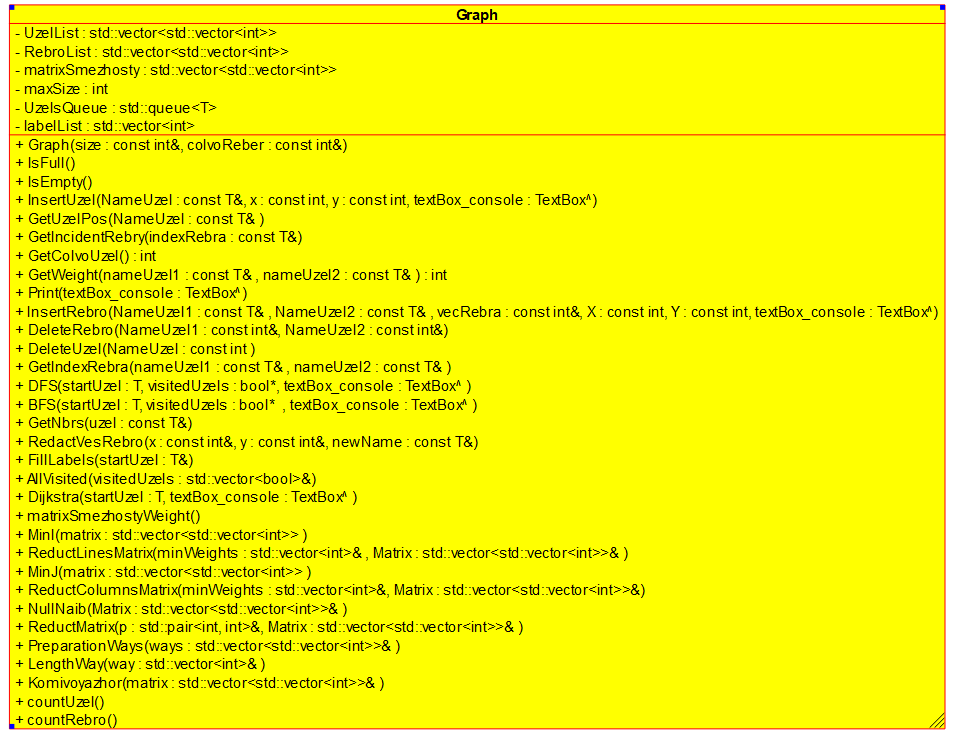


Рисунок А 1 - Диаграмма классов

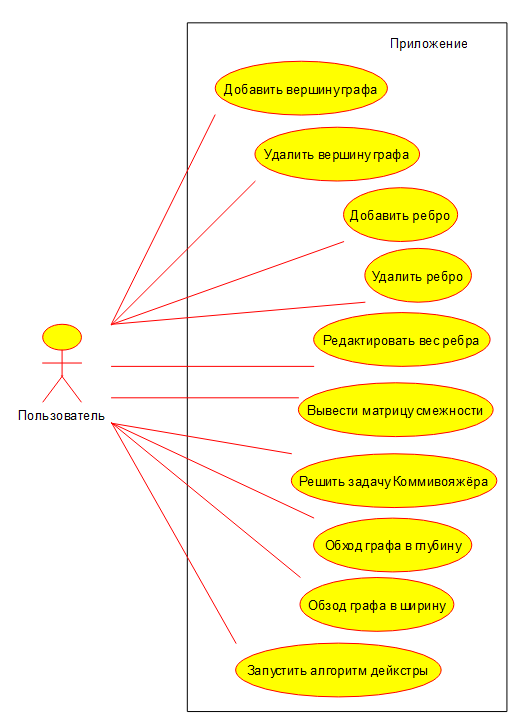


Рисунок А 2 - Диаграмма прецедентов

# ПРИЛОЖЕНИЕ Б

**Листинг программы**

#include "MainForm.h"

#include <Windows.h>

#include <vector>

#include <queue>

#include <string>

#include <stdlib.h>

#include <algorithm>

using namespace Komivoyazhor;

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MainForm);

return 0;

}

template <class T>

class Graph

{

public:

//вектор узлов

std::vector<std::vector<int>> UzelList;

//вектор ребер

std::vector<std::vector<int>> RebroList;

//матрица смежности

std::vector<std::vector<int>> matrixSmezhosty;

//размер вектора вершин и матрицы смежности

int maxSize;

//очередь узлов

std::queue<T> UzelsQueue;

//вектор расстояний

std::vector<int> labelList;

public:

Graph(const int& size, const int& colvoReber)

{

this->maxSize = size;

this->matrixSmezhosty = std::vector<std::vector<T>>(size, std::vector<T>(size));

for (int i = 0; i < this->maxSize; i++)

{

for (int j = 0; j < this->maxSize; j++)

{

this->matrixSmezhosty[i][j] = -1;

}

}

}

bool IsFull();//проверка графа на переполненность

bool IsEmpty();//проверка графа на пустоту

void InsertUzel(const T& NameUzel, const int x, const int y, TextBox^ textBox\_console);//вставить узел

int GetUzelPos(const T& NameUzel);//получить позицию (индекс) узла

std::pair<int, int> GetIncidentRebry(const T& indexRebra);//получить инциденты ребра

int GetColvoUzel();//получить количество узлов

int GetWeight(const T& nameUzel1, const T& nameUzel2);//получение веса ребра

void Print(TextBox^ textBox\_console);//печать графа

void lnsertRebro(const T& NameUzel1, const T& NameUzel2, const int& vecRebra, const int X, const int Y, TextBox^ textBox\_console);//вставка ребра

void DeleteRebro(const int& NameUzel1, const int& NameUzel2);//удаление ребра

void DeleteUzel(const int NameUzel);//удаление узла

int GetIndexRebra(const T& nameUzel1, const T& nameUzel2);

void DFS(T startUzel, bool\* visitedUzels, TextBox^ textBox\_console);//обход в глубину

void BFS(T startUzel, bool\* visitedUzels, TextBox^ textBox\_console);//обход в ширину

std::vector<T> GetNbrs(const T& uzel);//получение вектора соседей

void RedactVesRebro(const int& x, const int& y, const T& newName);

void FillLabels(T& startUzel);//Заполнение меток расстояния

bool AllVisited(std::vector<bool>& visitedUzels);//Проверяет, все ли узлы были посещены

void Dijkstra(T startUzel, TextBox^ textBox\_console);//алгоритм дейкстры

std::vector<std::vector<int>> matrixSmezhostyWeight();//матрица смежности с весом (а не индексами ребер)

std::vector<int> MinI(std::vector<std::vector<int>> matrix);//минимум по строке

std::vector<std::vector<int>> ReductLinesMatrix(std::vector<int>& minWeights, std::vector<std::vector<int>>& Matrix);//редукция строк

std::vector<int> MinJ(std::vector<std::vector<int>> ReductMatrix);//минимум по столбцу

std::vector<std::vector<int>> ReductColumnsMatrix(std::vector<int>& minWeights, std::vector<std::vector<int>>& Matrix);//редукция столбцов

std::pair<int, int> NullNaib(std::vector<std::vector<int>> Matrix);//выбор нулевого элемента с наибольшей оценкой

std::vector<std::vector<int>> ReductMatrix(std::pair<int, int>& p, std::vector<std::vector<int>>& Matrix);//редукция матрицы

std::vector<int> PreparationWays(std::vector<std::vector<int>>& ways);

int LengthWay(std::vector<int>& way);

std::vector<int> Komivoyazhor(std::vector<std::vector<int>>& matrix);//алгоритм комивояжёра

};

template <class T>

int Graph<T>::LengthWay(std::vector<int>& way)

{

std::vector<std::vector<int>> matrix = this->matrixSmezhostyWeight();

int sumLength = 0;

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

{

sumLength += matrix[way[i]][way[i + 1]];

}

return sumLength;

}

template <class T>

std::vector<std::vector<int>> Graph<T>::ReductMatrix(std::pair<int, int>& p, std::vector<std::vector<int>>& Matrix)

{

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

{

Matrix[p.first][i] = -1;

}

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

{

Matrix[i][p.second] = -1;

}

Matrix[p.second][p.first] = -1;

return Matrix;

}

template <class T>

std::vector<int> Graph<T>::PreparationWays(std::vector<std::vector<int>>& ways)

{

std::vector<int> way;

way.push\_back(0);

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

{

if (ways[i][0] == way.back())

{

way.push\_back(ways[i][1]);

ways.erase(ways.begin() + i);

i = -1;

}

}

return way;

}

template <class T>

std::vector<int> Graph<T>::Komivoyazhor(std::vector<std::vector<int>>& matrix)

{

bool flag = true;

std::vector<std::vector<int>> ways;

do

{

std::vector<int> di = MinI(matrix);//минимум по строкам

std::vector<std::vector<int>> ReductLinesMatrix = this->ReductLinesMatrix(di, matrix);//редукция строк

std::vector<int> dj = MinJ(ReductLinesMatrix);//минимум по столбцам

std::vector<std::vector<int>> ReductColumnsMatrix = this->ReductColumnsMatrix(dj, ReductLinesMatrix);//редукция столбцов

std::pair<int, int> p = NullNaib(ReductColumnsMatrix);//индексы нулевой клетки с наибольшей оценкой

matrix = this->ReductMatrix(p, ReductColumnsMatrix);//редукция матрицы

if (p.first != p.second != 0)

{

std::vector<int> Temp; Temp.push\_back(p.first); Temp.push\_back(p.second);

ways.push\_back(Temp);

Temp.clear();

}

else

{

flag = false;

std::vector<int> way;

way = this->PreparationWays(ways);

way.push\_back(this->LengthWay(way));

return way;

}

}

while (flag);

}

template <class T>

std::pair<int, int> Graph<T>::NullNaib(std::vector<std::vector<int>> Matrix)

{

std::vector<std::vector<int>> assessment;

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

{

for (int j = 0; j < this->UzelList.size(); j++)

{

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

{

int minElementLine = INT\_MAX;

for (int k = 0; k < this->UzelList.size(); k++)

{

if (Matrix[i][k] < minElementLine && Matrix[i][k] != -1 && k != j)

minElementLine = Matrix[i][k];

}

int minElementColumn = INT\_MAX;

for (int k = 0; k < this->UzelList.size(); k++)

{

if (Matrix[k][j] < minElementColumn && Matrix[k][j] != -1 && k != i)

minElementColumn = Matrix[k][j];

}

std::vector<int> Temp;

if (minElementLine == INT\_MAX) minElementLine = 0;

if (minElementColumn == INT\_MAX) minElementColumn = 0;

Temp.push\_back(minElementLine + minElementColumn);

Temp.push\_back(i);

Temp.push\_back(j);

assessment.push\_back(Temp);

Temp.clear();

}

}

}

int maxAssessment = -1;

std::pair<int, int> p;

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

{

if (assessment[i][0] > maxAssessment) {

p.first = assessment[i][1];

p.second = assessment[i][2];

maxAssessment = assessment[i][0];

}

}

return std::pair<int, int>(p);

}

template <class T>

std::vector<std::vector<int>> Graph<T>::matrixSmezhostyWeight()

{

std::vector<std::vector<int>> matrixSmezhostyWeight = std::vector<std::vector<T>>(this->UzelList.size(), std::vector<T>(this->UzelList.size()));

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

{

for (int j = 0; j < this->UzelList.size(); j++)

{

if (this->matrixSmezhosty[i][j] != -1)

matrixSmezhostyWeight[i][j] = this->RebroList[this->matrixSmezhosty[i][j]][0];

else matrixSmezhostyWeight[i][j] = -1;

}

}

return matrixSmezhostyWeight;

}

template <class T>

std::vector<std::vector<int>> Graph<T>::ReductColumnsMatrix(std::vector<int>& minWeights, std::vector<std::vector<int>>& Matrix)

{

std::vector<std::vector<int>> matrixReduct = std::vector<std::vector<T>>(this->UzelList.size(), std::vector<T>(this->UzelList.size()));

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

{

for (int j = 0; j < this->UzelList.size(); j++)

{

int weight = Matrix[j][i];

if (weight != -1)

matrixReduct[j][i] = weight - minWeights[i];

else matrixReduct[j][i] = -1;

}

}

return matrixReduct;

}

template <class T>

std::vector<int> Graph<T>::MinJ(std::vector<std::vector<int>> ReductMatrix)

{

std::vector<int> minWeights;

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

{

int minWeight = INT\_MAX;

for (int j = 0; j < this->UzelList.size(); j++)

{

int weight = ReductMatrix[j][i];

if (weight < minWeight && weight != -1)

minWeight = weight;

}

if (minWeight != INT\_MAX)

minWeights.push\_back(minWeight);

else minWeights.push\_back(-1);

}

return minWeights;

}

template <class T>

std::vector<std::vector<int>> Graph<T>::ReductLinesMatrix(std::vector<int>& minWeights, std::vector<std::vector<int>>& Matrix)

{

std::vector<std::vector<int>> matrixReduct = std::vector<std::vector<T>>(this->UzelList.size(), std::vector<T>(this->UzelList.size()));

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

{

for (int j = 0; j < this->UzelList.size(); j++)

{

int weight = Matrix[i][j];

if (weight != -1)

matrixReduct[i][j] = weight - minWeights[i];

else matrixReduct[i][j] = -1;

}

}

return matrixReduct;

}

template <class T>

std::vector<int> Graph<T>::MinI(std::vector<std::vector<int>> matrix)

{

std::vector<int> minWeights;

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

{

int minWeight = INT\_MAX;

for (int j = 0; j < this->UzelList.size(); j++)

{

int weight = matrix[i][j];

if (weight < minWeight && weight != -1)

minWeight = weight;

}

if (minWeight != INT\_MAX)

minWeights.push\_back(minWeight);

else minWeights.push\_back(-1);

}

return minWeights;

}

template <class T>

void Graph<T>::FillLabels(T& startUzel)

{

labelList = std::vector<T>(UzelList.size());

//Заполнение меток расстояния

for (int i = 0, size = this->UzelList.size(); i < size; ++i)

this->labelList[i] = 1000000;

int pos = GetUzelPos(startUzel);

this->labelList[pos] = 0;

}

template <class T>

bool Graph<T>::AllVisited(std::vector<bool>& visitedUzels)

{

//Проверяет, все ли узлы были посещены

bool flag = true;

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

if (visitedUzels[i] != true) flag = false;

return flag;

}

template <class T>

void Graph<T>::Dijkstra(T startUzel, TextBox^ textBox\_console)

{

if (GetUzelPos(startUzel) == -1)

{

return;

}

std::vector<bool> visitedUzels(this->UzelList.size());

fill(visitedUzels.begin(), visitedUzels.end(), false);

this->FillLabels(startUzel);

T curScr = startUzel;

std::vector<T> neighbors;

while (!this->AllVisited(visitedUzels))

{

neighbors = this->GetNbrs(curScr);

int startLabel = labelList[GetUzelPos(curScr)];

//Самый близкий сосед

T\* minNeighbor\_ptr = nullptr;

int minW = 1000000;

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

{

int weight = this->GetWeight(curScr, neighbors[i]);

int nIndex = this->GetUzelPos(neighbors[i]);

int nextLabel = labelList[nIndex];

if (startLabel + weight < nextLabel)

{

labelList[nIndex] = startLabel + weight;

}

if (!visitedUzels[nIndex] && minW > labelList[nIndex])

{

minW = labelList[nIndex];

minNeighbor\_ptr = &neighbors[i];

}

}

visitedUzels[GetUzelPos(curScr)] = true;

if (minNeighbor\_ptr != nullptr)

{

curScr = \*minNeighbor\_ptr;

}

if (!this->AllVisited(visitedUzels) && visitedUzels[GetUzelPos(curScr)] == true)

{

break;

}

}

/\* Вывод результата работы алгоритма на экран \*/

for (int i = 0; i < GetUzelPos(startUzel); ++i)

{

textBox\_console->Text += "Кратчайшее расстояние от вершины " + startUzel + " до вершины " + UzelList[i][0].ToString() + " равно "+ labelList[GetUzelPos(UzelList[i][0])].ToString() + "\r\n";

}

for (int i = GetUzelPos(startUzel) + 1; i < UzelList.size(); ++i)

{

textBox\_console->Text += "Кратчайшее расстояние от вершины "+ startUzel + " до вершины " + UzelList[i][0].ToString() + " равно " + labelList[GetUzelPos(UzelList[i][0])].ToString() + "\r\n";

}

}

template <class T>

std::vector<T> Graph<T>::GetNbrs(const T& uzel)

{

std::vector<T> nbrsList;//создание списка соседей

int pos = this->GetUzelPos(uzel); //вычисление позиции uzel в матрице смежности

if (pos != -1)

{

//проверка, что uzel есть в матрице смежности

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

{

if (this->matrixSmezhosty[pos][i] != -1)

{

nbrsList.push\_back(this->UzelList[i][0]);

}

}

}

return nbrsList; // возврат списка соседей

}

template<class T>

void Graph<T>::RedactVesRebro(const int& x, const int& y, const T& newName)

{

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

{

if (this->RebroList[i][1] == x && this->RebroList[i][2] == y)

{

this->RebroList[i][0] = newName;

}

}

}

template <class T >

void Graph<T>::DFS(T startUzel, bool\* visitedUzels, TextBox^ textBox\_console)

{

textBox\_console->Text += "Вершина " + startUzel + " посещена \r\n";

visitedUzels[this->GetUzelPos(startUzel)] = true;

std::vector<T> neighbors = this->GetNbrs(startUzel);

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

{

if (!visitedUzels[this->GetUzelPos(neighbors[i])])

{

this->DFS(neighbors[i], visitedUzels, textBox\_console);

}

}

}

template <class T>

void Graph<T>::BFS(T startUzel, bool\* visitedUzels, TextBox^ textBox\_console)

{

//Условие истинно только при первом вызове функции

if (visitedUzels[this->GetUzelPos(startUzel)] == false)

{

this->UzelsQueue.push(startUzel);

visitedUzels[this->GetUzelPos(startUzel)] = true;

textBox\_console->Text += "Вершина " + startUzel + " обработана\r\n";

}

std::vector<T> neighbors = this->GetNbrs(startUzel);

this->UzelsQueue.pop();

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

{

if (!visitedUzels[this->GetUzelPos(neighbors[i])])

{

this->UzelsQueue.push(neighbors[i]);

visitedUzels[this->GetUzelPos(neighbors[i])] = true;

textBox\_console->Text += "Вершина " + neighbors[i] + " обработана\r\n";

}

}

if (this->UzelsQueue.empty())

{

return;

}

BFS(UzelsQueue.front(), visitedUzels, textBox\_console);

}

template <class T>

inline bool Graph<T>::IsFull()

{

return this->UzelList.size() == this->maxSize;

}

template <class T>

inline bool Graph<T>::IsEmpty()

{

return this->UzelList.size() == 0;

}

template <class T>

inline int Graph<T>::GetUzelPos(const T& uzel)

{

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

{

if (this->UzelList[i][0] == uzel)

{

return i;

}

}

return -1;

}

template <class T>

inline std::pair<int, int> Graph<T>::GetIncidentRebry(const T& indexRebra)

{

for (int i = 0; i < this->maxSize; i++)

{

for (int j = 0; j < this->maxSize; j++)

{

if (this->matrixSmezhosty[i][j] == indexRebra)

{

return std::pair<int, int>(i, j);

}

}

}

}

template <class T>

inline int Graph<T>::GetIndexRebra(const T& NameUzel1, const T& NameUzel2)

{

int indexU1 = this->GetUzelPos(NameUzel1);

int indexU2 = this->GetUzelPos(NameUzel2);

return matrixSmezhosty[indexU1][indexU2];

}

template <class T>

inline int Graph<T>::GetColvoUzel()

{

return this->UzelList.size();

}

template <class T>

inline int Graph<T>::GetWeight(const T& nameUzel1, const T& nameUzel2)

{

if (this->IsEmpty())

{

return 0;

}

int uzel1\_pos = this->GetUzelPos(nameUzel1);

int uzel2\_pos = this->GetUzelPos(nameUzel2);

if (uzel1\_pos == -1 || uzel2\_pos == -1)

{

return 0;

}

if (this->matrixSmezhosty[uzel1\_pos][uzel2\_pos] < 0) return -1;

return this->RebroList[this->matrixSmezhosty[uzel1\_pos][uzel2\_pos]][0];

}

template <class T>

void Graph<T>::InsertUzel(const T& nameUzel, const int x, const int y, TextBox^ textBox\_console)

{

if (!this->IsFull())

{

int indexPust = -1;

for (int i = 0; i < UzelList.size(); i++)//таблицу с локацией мы не очищаем, а просто перезаписываем на "пустые" места, либо добавляем новые строки

{

if (UzelList[i][0] == -1) {

indexPust = i; break;

}

}

//запоминание положения узла

std::vector<T> Temp;

Temp.push\_back(nameUzel);

Temp.push\_back(x);

Temp.push\_back(y);

if (indexPust == -1)

{

this->UzelList.push\_back(Temp);

}

else

{

this->UzelList[indexPust][0] = nameUzel;

this->UzelList[indexPust][1] = x;

this->UzelList[indexPust][2] = y;

}

Temp.clear();

}

else textBox\_console->Text += "Граф переполнен";

}

template<class T>

void Graph<T>::lnsertRebro(const T& NameUzel1, const T& NameUzel2, const int& vecRebra, const int X, const int Y, TextBox^ textBox\_console)

{

int uzelPos1 = this->GetUzelPos(NameUzel1),

uzelPos2 = this->GetUzelPos(NameUzel2);

if (uzelPos1 != (-1) && uzelPos2 != (-1))

{

if (this->matrixSmezhosty[uzelPos1][uzelPos2] != -1 && this->matrixSmezhosty[uzelPos2][uzelPos1] != -1 )

{

textBox\_console->Text += "Ребро между вершинами уже есть.\r\n";

}

else

{

int indexPust = -1;

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

{

if (RebroList[i][0] == -1) {

indexPust = i; break;

}

}

//добавление в вектор ребер нового ребра

std::vector<T> Temp;

Temp.push\_back(vecRebra);

Temp.push\_back(X);

Temp.push\_back(Y);

if (indexPust == -1)

{

this->RebroList.push\_back(Temp);

this->matrixSmezhosty[uzelPos1][uzelPos2] = this->matrixSmezhosty[uzelPos2][uzelPos1] = RebroList.size() - 1;

}

else

{

this->RebroList[indexPust][0] = vecRebra;

this->RebroList[indexPust][1] = X;

this->RebroList[indexPust][2] = Y;

this->matrixSmezhosty[uzelPos1][uzelPos2] = this->matrixSmezhosty[uzelPos2][uzelPos1] = indexPust;

}

Temp.clear();

}

}

else

{

textBox\_console->Text += "Обеих вершин (или одной из них) нет в графе.\r\n";

}

}

template <class T >

void Graph<T>::Print(TextBox^ textBox\_console)

{

if (!this->IsEmpty())

{

textBox\_console->Text += "Матрица смежности:\r\n";

textBox\_console->Text += String::Format("{0,5:F4}", "-");

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

{

if(this->UzelList[i][0] != -1)

textBox\_console->Text += String::Format("{0,6:F5}", (i + 1).ToString());

}

textBox\_console->Text += "\r\n";

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

{

if (this->UzelList[i][0] != -1)

{

textBox\_console->Text += String::Format("{0,5:F4}", (i + 1).ToString());

for (int j = 0; j < this->UzelList.size(); j++)

{

String^ string = "-1";

if (this->matrixSmezhosty[i][j] != -1) string = this->RebroList[this->matrixSmezhosty[i][j]][0].ToString();

textBox\_console->Text += String::Format("{0,5:F4}", string);

}

textBox\_console->Text += "\r\n";

}

}

textBox\_console->Text += "\r\n";

}

else textBox\_console->Text += "Граф пуст.";

}

template <class T>

inline void Graph<T>::DeleteUzel(const int NameUzel)

{

if (!this->IsEmpty())

{

int indexUzelDelete = this->GetUzelPos(NameUzel);

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

{

this->matrixSmezhosty[i][indexUzelDelete] = -1;

}

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

{

this->matrixSmezhosty[indexUzelDelete][i] = -1;

}

this->UzelList[indexUzelDelete][0] =

this->UzelList[indexUzelDelete][1] =

this->UzelList[indexUzelDelete][2] = -1;

}

}

template <class T>

inline void Graph<T>::DeleteRebro(const int& NameUzel1, const int& NameUzel2)

{

int indexRebroDelete = this->GetIndexRebra(NameUzel1, NameUzel2);

//заменим в локации линий данные данной линии на -1

this->RebroList[indexRebroDelete][0] =

this->RebroList[indexRebroDelete][1] =

this->RebroList[indexRebroDelete][2] = -1;

//заменим на -1 в матрице

for (int i = 0; i < this->maxSize; i++)

{

for (int j = 0; j < this->maxSize; j++)

{

if (matrixSmezhosty[i][j] == indexRebroDelete)

{

this->matrixSmezhosty[i][j] = -1;

}

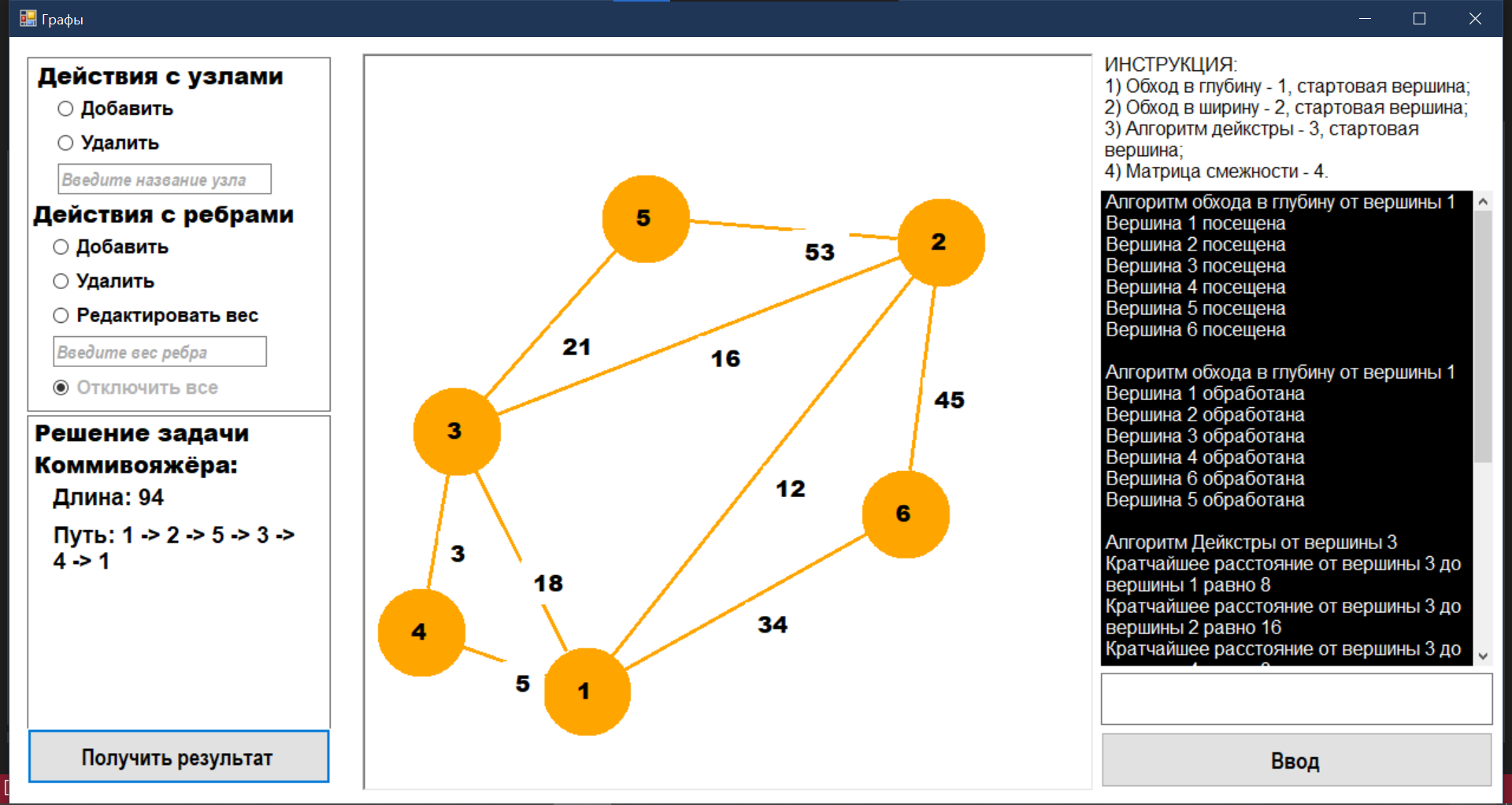
}

}

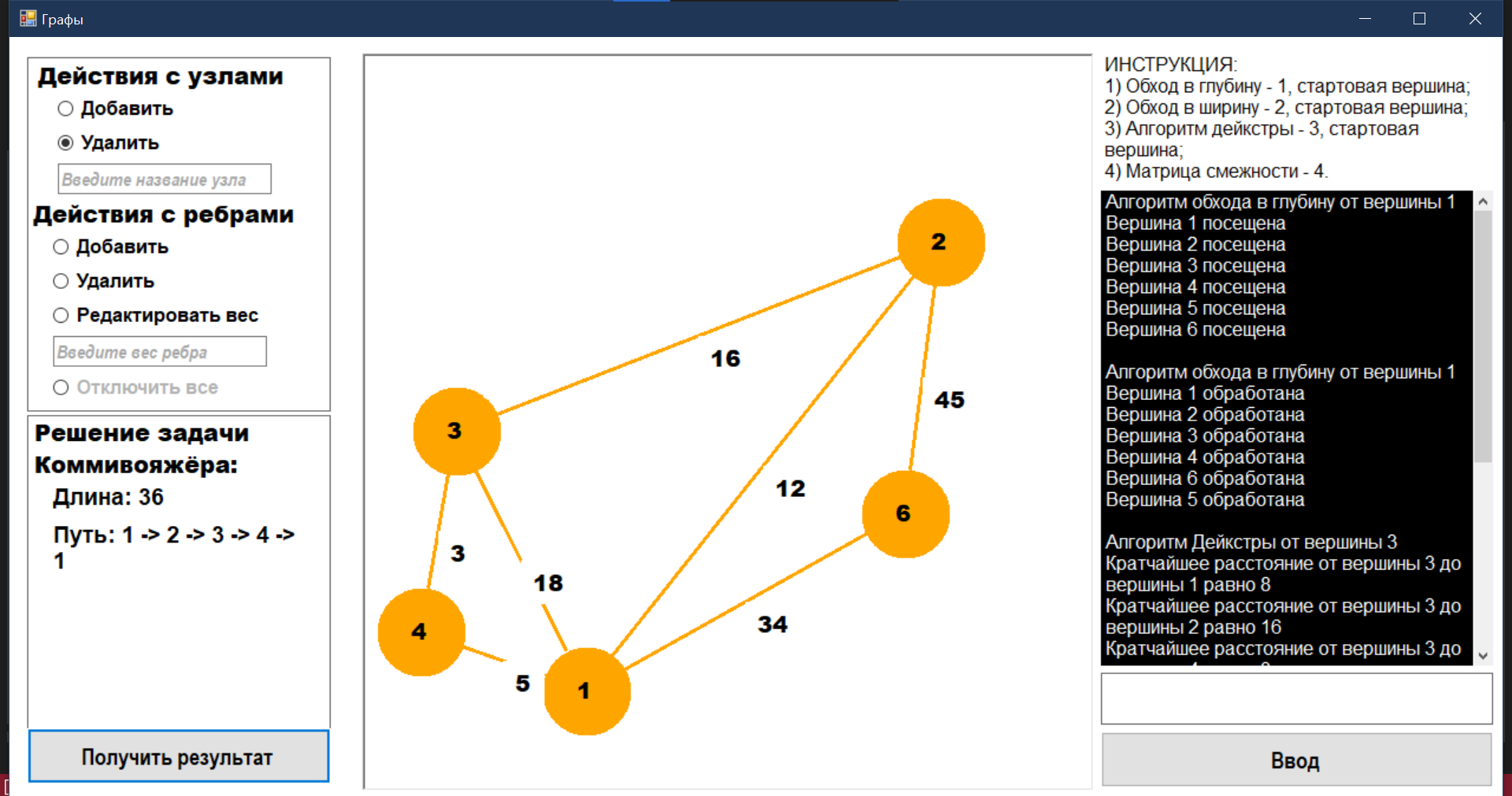
}

# ПРИЛОЖЕНИЕ В

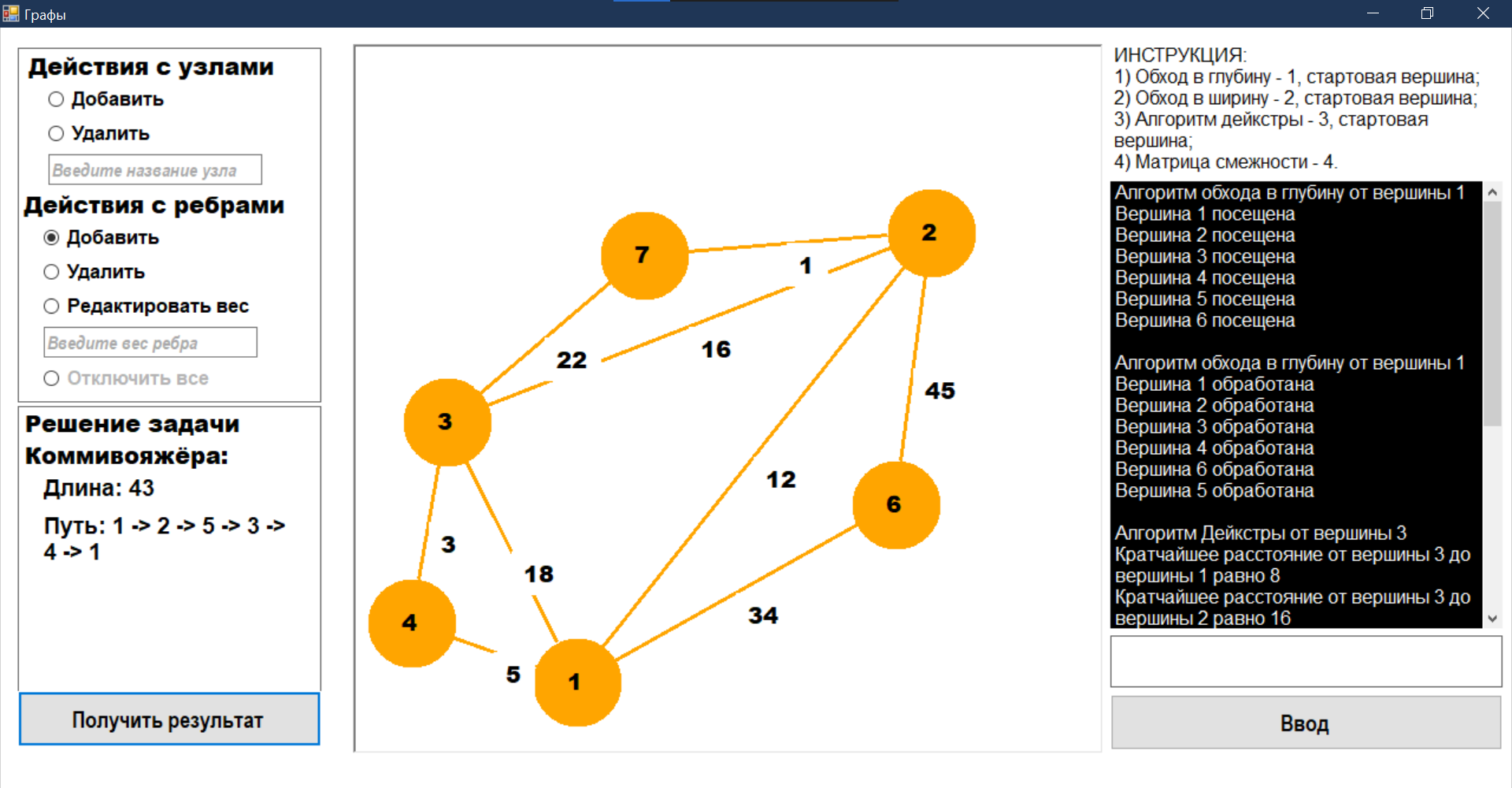
**Работа кода**



*Рисунок В 1 – решение задачи Коммивояжера*

**

*Рисунок В 2 – удаление узла*

**

*Рисунок В 3 – добавление нового узла с заданными ребрами*