Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической и прикладной информатики

Курсовая работа по дисциплине

«Структуры данных и алгоритмы»

Факультет:	Прикладной математики и информатики
Группа:	ПМИ-93
Студент:	Мельников Святослав
Преподаватель:	Еланцева Ирина Леонидовна

Новосибирск

Условие задачи

Заданы две системы двухсторонних дорог с одним и тем же множеством городов (железные и шоссейные дороги).
Найти минимальный по длине путь из города A в город B (который может проходить как по железным,
так и по шоссейным дорогам) и места пересадок с одного вида транспорта на другой на этом пути.

2. **Анализ задачи**

2.1. Исходные данные задачи

В первой строке входного файла input.txt записывается общее число городов n.
Во второй строке входного файла записаны названия городов через пробел.
В третьей строке записаны названия городов выступающие в роли начальной точки и пункта назначения A и B.
С четвертой строки и до конца файла содержится информация о дорогах. тип дороги> <название города 1> <название города 2> <длина данной дороги>, записанная через пробел. (Если дороги между городами нет, можно либо написать в графе <длина данной дороги> 0, либо не указывать эти два города вовсе).

Пример входных данных:

3
Новосибирск Красноярск Зеленогорск
Новосибирск Зеленогорск
ЖД Новосибирск Зеленогорск 11
ЖД Новосибирск Красноярск 8
М53 Красноярск Зеленогорск 1

2.2. Результат:

Если программа сработала корректно, то файл выходных данных будет выглядеть так:

В первой строке выходного файла output.txt выводится фраза «Расстояние от города A до города B» и кратчайшее расстояние между ними.
Со второй строки и до конца выходного файла выводится фраза «Точка пересадки» <Название города> (В котором совершена пересадка).

Иначе в файле будет текстовое сообщение об ошибке.

Пример выходных данных:

Кратчайшее расстояние от города Новосибирск до города Зеленогорск
9
Точка пересадки: Красноярск

2.3 Решение:

Математическая модель – мультиграф, неориентированный, взвешенный, несвязный, помеченный.

Определение:

Мультиграф - граф, в котором может быть пара вершин, которая соединена более чем одним ребром (ненаправленным). В данной задаче, подразумевается больше чем одна дорога одного из двух типов. (Железные дороги, либо шоссейные трассы).

Неориентированный граф – граф, ребра которого не имеют направления. В данной задаче ребра заданы двухсторонними дорогами, поэтому граф неориентированный.

Взвешенный граф – граф, ребра которого имеют вес, определяют какую-либо характеристику. В данной задаче вес ребра характеризует длину дороги от между городами.

Несвязный граф – граф, допускающий вершину не связанную с другими вершинами ребрами. В мультиграфе данная вершина может быть связана сама с собой. В данной задаче нельзя исключать такой вариант.

Помеченный граф – граф, вершины которого отображают какое-либо название. В данной задаче, метками являются названия городов, вводимые пользователем.

Анализ графа:

1. Проверим, существует ли начальный пункт маршрута, конечный пункт и не совпадает ли конечный пункт маршрута с пунктом отправления. Так как во внутреннем представлении данных, все города будут иметь индексы (от 0 до n-1, где n - общее число городов), то заранее заведем две переменные, которые будут хранить индексы городов A и B и присвоим им значение равное бесконечности (Константное значение равное максимально возможному значению, хранимому в типе данных int, а в данном случае необходимое для того, чтобы при наличии большого количества городов, индексы не совпали). Затем считываем все названия городов с файла в массив, и ищем в этом массиве города A и B. Как только города будут найдены, заведенные переменные A и изменят своё значения, и будут равны целочисленным значениям, соответствующим их индексам в массиве названий городов. Если одна из переменных, характеризующих индекс города A, либо индекс города B не изменит своё значение, то начальный пункт и конечный пункт маршрута не существуют, либо они совпадают, иначе продолжаем.

2. Проверим, не является ли начальный, либо конечный пункт изолированными городами. Для начала заполняем матрицы смежности железных дорог, и шоссейных трасс. Из этих двух матриц формируем общую матрицу смежности, в которой будут содержаться кратчайшие дороги между городами. После этого, ищем сумму длин в строке матрицы. Если сумма равна нулю и номер строки, в которой сумма длин равна нулю совпадает с индексом пункта А или индексом пункта В, то функция, осуществляющая операцию проверки на изоляцию, вернет 0,что означает – Решения задачи нет, иначе продолжаем. (К сожалению, данный способ может найти только вершины изолированные от всех, но он допускает вариант, когда города А и В могут быть в изолированных системах).
3. Два города связаны несколькими дорогами одного типа. (Свойства мультиграфа). Если два города связаны несколькими дорогами одного типа, то в матрицу смежности дорог данного типа будет занесена кратчайшая из дорог.
4. Город соединен сам с собой петлей. (Свойства мультиграфа). Данное свойство никак не повлияет на корректность работы программы, так как при составлении матрицы кратчайших путей по главной диагонали любые значения будут заменены нулями. Это необходимо для проверки на изолированность города, то есть пункт

Анализ входных данных:

1. Проверим количество городов на положительное значение. Если количество городов будет задано неположительным значением, то в файл будет выведена фраза: “Ошибка, введено неверное количество городов”, иначе продолжаем.
2. Проверка на отрицательную длину дороги, производится во время считывания данных с файлы. Если будет найдена хотя бы одна длина дороги, заданная отрицательным числом, то в файле появится сообщение: “Проверьте корректность входных данных. Введено неверное расстояние между городами <город 1> и <город 2>”, но программа продолжит работу без занесения этой длины в матрицу смежности определенного типа дорог.
3. Проверка на тип дороги. Программа может различить только два типа дорог – железные дороги и шоссейные трассы. Если в графе тип дороги> во входном файле указан тип «ЖД», то считается, что между городом 1 и городом 2 проложена железная дорога, если указан любой другой тип (главное без разделительных пробелов), например: «М53», «А-101», «Би-2», «РусскаяДорога», то эти дороги будут сразу определены как шоссейные трассы.

Следовательно, решение задачи может быть найдено и будет являться корректным, только если выполняются все следующие условия:

1. Количество городов задано положительным числом
2. Начальный пункт и конечный пункт маршрута существуют, и они не совпадают.
3. Начальный пункт и конечный пункт маршрута не являются изолированными от общей системы дорог.

   (необязательные условия, влияющие только на корректность программы)
4. Все длины дорог являются неотрицательными числами
5. Тип дороги введен в соответствии с условием: «ЖД», либо любой другой.

После анализа графа и анализа входных данных непосредственно приступаем к определению формальной постановки задачи и составлению алгоритма её решения.

Формальная постановка задачи:

Во взвешенном, неориентированном, несвязном и помеченном мультиграфе, состоящем из двух систем дорог, отличных друг от друга по типу, найти кратчайший путь из пункта A в пункт B, и вершины, в которых будет происходить смена систем дорог.

Следовательно, данную задачу можно разбить на две крупные подзадачи.
1. Нахождение кратчайшего пути между двумя пунктами, через составление матриц общей системы, в которой будут храниться кратчайшие расстояния между городами, как по железным дорогам, так и по шоссейным трассам, и использованием в этой матрицы алгоритма Дейкстры.

2. Нахождение точек пересадки, по средствам составления матрицы, хранящей в себе типы дорог, соединяющие города и восстановления пути от города A до города B.

Алгоритм решения поставленных подзадач:

Для того чтобы приступить к решению поставленных подзадач, мы должны осуществить проверку входных данных на корректность, в соответствие с обязательными условиями, которые прописаны выше.

Если первые два обязательных условия выполняются, то можем продолжать.

1. Нахождение кратчайшего пути.

Для начала создаем два двухмерных массива, размерностью n*n, хранящие матрицы смежностей железных дорог и шоссейных трасс. Заполняем их нулями, для того, чтобы во входных данных не указывать пары городов, между которыми связь по данному типу дорог отсутствует. Затем считываем данные с файла до момента, пока не конец файла, и изменяем матрицы по следующему принципу:1)Находим индексы введенных городов. 2)Если <тип дороги> совпадает с «ЖД», то обращаемся к пункту связанному с железными дорогами, иначе с шоссейными трассами. 3) Если в ячейки матрицы соединяющая эти два города стоит начальное значение (0) и считанная длина пути больше 0, то изменяем матрицу смежности определенного типа дороги. Иначе проверяем, является ли новая длина пути меньше имеющейся и положительной. Если да, то изменяем матрицу смежности. Иначе, если длина пути меньше нуля, выводим сообщение в файл, о том, что входные данные введены некорректно.

После заполнения данных о дорогах, составляем единую матрицу дорог между городами. Она будет содержать в себе кратчайшие пути между городами и составляться следующим образом. 1) По главной диагонали будут расставлены нули, чтобы исключить возможные петли, которые могут повлиять на проверку начального и конечного пункта на изолированность. 2) Проверяем, существуют ли в данной ячейки железные и шоссейные дороги, если есть, то выбираем минимально значение из двух. 3) Если в выбранной ячейки не существует одной из дорог, то заполняем ячейку имеющимися данными.

После заполнения матриц кратчайших путей, осуществляем проверку на обязательное условие об изолированности городов. Считаем сумму элементов в строке матрицы кратчайших дорог. Если одна из строк с индексами A или B равны нулю, то решения задачи нет, потому что минимум один город изолирован от общей системы дорог.

Перед запуском алгоритма Дейкстры объявляем одномерный массив, в который будем записывать кратчайшие расстояния от стартовой точки до остальных городов. Инициализируем элементы массива, присваивая им значения бесконечности (Так как вершины не посещены, и расстояние до них ещё неизвестно). А элемент массива с индексом A помечаем нулем.

Алгоритм Дейкстры.

1. Заводим одномерный массив для хранения информации о посещенных городах. Все элементы массива изначально равны 0.
очереди. Обновляем данные о них в одномерном массиве кратчайшего пути от города A, теперь там будет находится сумма кратчайшего расстояния от текущей стартовой вершины до вершины A (в начале (от A до A) это значение равно нулю) и длина ребра от A до города-соседа. Продолжаем эту операцию, пока не обойдем всех соседей. По завершении

операции, ставим метку в массив посещенных городов. И переходим к следующему городу.

. Продолжаем обход с города, который мы еще не посетили (отталкиваясь от массива визитов), но изменили значения кратчайшего пути до этого города (относительно города А), то есть города, являющегося соседом городу А. Повторяем операцию, описанную выше, обходя всех городов-соседей, относительно нашей новой точки старта, для того чтобы уменьшить найденные коротки дистанции в массиве кратчайших расстояний относительно города А.

4. Повторяем подобные действия, пока не посетим все связанные вершины.

После чего, осуществляем проверку, изменили ли мы значение кратчайшего расстояния до города В. Если нет, то города А и В находятся в разных замкнутых системах графов. Иначе выводим в файл значение, равное кратчайшему расстоянию до пункта В – ответ на подзадачу номер

2. Нахождение точек пересадки.

Заводим двухмерный массив для хранения актуальных коротких дорог. Заполняем его следующим образом. Сравниваем ячейку, отличную от нуля, из массива коротких дорог с ячейкой из массива железных дорог. Если ячейки совпали, то в аналогичной ячейки массива актуальных дорог ставим единицу. Если нет, то двойку. Если ячейка равна нулю, то оставляем 0, как показатель отсутствия дороги.

Вновь заполняем массив посещенных городов бесконечностями. (Потому что теперь массив будет содержать индексы городов, и нельзя допустить, чтобы они совпали с какой-либо константой).

Начинаем поиск с конца. Поэтому нулевому элементу массива присваиваем индекс конечного города В. Создаем переменную отвечающее за длину и присваиваем ей значение равное кратчайшему расстоянию от А до В (ответ на подзадачу 1). Далее запускаем цикл, в котором будем изменять индекс города В, пока он не станет равным городу А.

1. Просматриваем все города на наличие связи с городом В.

2. Как только связь обнаружена, то проверяем следующее условие: Если разность длины от А до В и длины от В до найденного города равна длине от А до найденного города, то значит, что переход в город В был совершен с этой вершины, следовательно изменяем индекс города В на индекс города – перехода. Иначе продолжаем поиск.

3. После завершения обхода, и восстановления пути, передаем массив хранящий в себе индексы городов, посещенных во время пути и анализируем его, используя матрицу актуальных дорог.

4. Если дорога от точки B до города-перехода не совпадает с дорогой от города-перехода до следующего посещенного города, то этот город переход является точкой пересадки, выводим полученный результат в файл. Иначе продолжаем, пока не дойдем до города A.
Если точек пересадок не обнаружено, то на в файл будет выведено сообщение: “Точки пересадки отсутствуют”.

3. Структуры данных, используемых для представления исходных данных и результатов задачи

Внешнее представление данных:

Представление входных данных:

Целое число, количество городов

Строки, соответствующие названиям городов. Число строк равно количеству вершин

Строки, название города A и города B

Строки, тип дороги, города между которыми проложена дорога, целая число, длина дороги

Представление выходных данных:

Если решение задачи не было найдено, выводится:

«Ошибка, введено неверное количество городов» - Количество городов задано неположительным числом.

Путешествие закончилось, не успев начаться!» - Начальный пункт и конечный пункт маршрута не существуют, либо они совпадают.

3. «Летите на самолете, город изолирован от общей системы дорог!» - Если города A и B являются, изолированы от общей системы дорог, или находятся в несвязанных между собой системах.

Если решение задачи было найдено, но могло быть некорректно, выводится:

1. «Проверьте корректность входных данных. Введено неверное расстояние между городами <город 1> и <город 2>» - Если некоторые длины дорог представлены отрицательными значениями, следовательно, считаны не будут.

Если решение найдено, то выводится:

1. «Кратчайшее расстояние между городом <Название города A> и Название города B> »

Целое число

2. Если точки пересадки есть:
«Точки пересадки: <Название города>»

Иначе:

«Точек пересадок нет»

Внутреннее представление данных:

Представление входных данных:

Целочисленная переменная, которая будет динамически задавать размерность всем массивам в программе (для одномерных, размерность будет до n, для двухмерных n*n)

Заводим одномерный массив, элементами которого будут строки, в которых записаны названия городов.

Заводим две строковых переменных, названия города A и название города

Заводим три строковых переменных, характеризующие тип дороги, город 1, город 2.
Целочисленная переменная, характеризующая длину пути, между городами 1 и 2.

Заводим два двухмерных массива, которые будут содержать матрицы смежностей железных и шоссейных дорог и заполненными целой переменной, описанной выше.

Представление выходных данных:

В случае отсутствия решений — строка «Решения нет»

В случае наличия решений — две строки : названия исходного и конечного городов в одной и последовательность строк — названия городов в порядке прохождения в другой.

Укрупненный алгоритм решения задачи

4.1. Укрупненный алгоритм решения задачи

Открываем файл.

Считываем входные данные.
Проверяем на обязательные условия 1 и 2.
Заполняем матрицы смежностей железных и шоссейных дорог.
Заполняем общую матрицу смежности коротких дорог.
Находим кратчайшее расстояние от точки A до точки B.
Выводим найденное расстояние в файл.
Заполняем матрицу актуальных коротких дорог.
Восстанавливаем путь от B до A.
Находим точки пересадки на этом пути.
Выводим данные о точках пересадки в файл.

}

4.2. Укрупненный алгоритм нахождения кратчайшего пути

Заводим одномерный массив для хранения кратчайших расстояний от точки А.Заполняем его бесконечностью. Элементу с индексом А присваиваем значение 0.

Заводим одномерный массив для хранения информации о посещенных вершинах.

Запускаем Алгоритм Дейкстры.

Обходим всех соседей города А. Обновляем данных о кратчайших расстояниях о них в массиве.

После обхода всех соседей, помечаем город А в массиве визитов как посещенный, и продолжаем обход, начиная с города-соседа, также обновляя данный о кратчайших расстояниях.

Продолжаем эту операцию до тех пор, пока не посетим все связанные города.

Укрупненный алгоритм нахождения точек пересадки.

Заводим двухмерный массив для хранения данных актуальных дорог.

Заводим массив посещенных вершин.

Восстанавливаем маршрут.

Просматриваем все города связанные с городом В.

Проверяем условие: Разность (Длины от А до В) и (Длины от В до города-соседа) = Длина от А до соседа.

Изменяем индекс города В = индекс города-соседа.

Повторяем операцию восстановления маршрута, пока В!=А.

Используя таблицу актуальных дорог проверяем: совпадает ли дорога от города В (введенного значения) до города-соседа, с дорогой от города-соседа до следующего посещенного города.

Структура программы

Текст программы разбит на два модуля.

Модуль 1 – Brain.cpp – содержит функции, осуществляющие работу модуля 2.

Модуль 2 – worker.cpp (main) – главный трудяга. Выполняет всю тяжелую работу: считывание с файл, реализация алгоритма программы, запись в файл.

5.1 Состав модуля brain.cpp

Функция *EmptyRoads:*

– назначение:

Заполнение матриц смежностей нулями.

- прототип функции:

```
int** EmptyRoads(int** r, int n);
```

-параметры:

r – матрица смежностей

n – количество городов/ размерность массива

Функция *newgraph:*

– назначение:

Создание объединенной матрицы смежности с короткими дорогами.

- прототип функции:

```
int** newgraph(int** rw, int** rd, int** ed, int n);
```

-параметры:

rw – матрица смежностей железных дорог

rd – матрица смежностей шоссейных трасс

ed – обобщенная матрица смежностей

n – количество городов/ размерность массива

Функция *Isolation:*

– назначение:

Проверка точек A и B на изолированность от общей системы дорог.

- прототип функции:

```
int Isolation(int** ed, int n, int A, int B);
```

-параметры:

ed – обобщенная матрица смежностей

n – количество городов/ размерность массива

A – индекс начального города

B – индекс конечного города

Функция *Actulroad:*

– назначение:

Создание матрицы смежности, хранящей данный о типах актуальных коротких дорог.

- прототип функции:

```
int** ActualRoad(int** ar, int** rw, int** rd, int** ed, int n);
```

-параметры:

ar – матрица смежности актуальных дорог

rw – матрица смежностей железных дорог

rd – матрица смежностей шоссейных трасс

ed – обобщенная матрица смежностей

n – количество городов/ размерность массива

Функция *ThisIsTheWay:*

– назначение:

Реализация алгоритма Дейкстры, поиск кратчайшего пути.

- прототип функции: `int* ThisIsTheWay(int* sh, int** ed, int n);`

-параметры:

`sh – массив кратчайших расстояний от города A`

ed – обобщенная матрица смежностей

n – количество городов/ размерность массива

Функция *BuildWay:*

– назначение:

Восстановление пути.

- прототип функции:

```
int* BuildWay(int** ed, int* sh, int n, int A, int B, int** ar, int* v);
```

-параметры:

`sh – массив кратчайших расстояний от города A`

ed – обобщенная матрица смежностей

n – количество городов/ размерность массива

A – индекс начального города

B – индекс конечного города

ar – матрица смежности актуальных дорог

– массив посещенных вершин

5.2Состав модуля worker.cpp (main)

Главная функция main:

– назначение:

Определение входных и выходных данных. Чтение данных с файла. Реализацию алгоритма по решению поставленной задачи(проверка на обязательные условия, решение подзадач).Организация связи с пользователем. Запись результата в файл.

- прототип функции:

Текст программы на языке Си (C++)

Queen. h

```
#pragma once
#ifndef QUEEN_H
#define QUEEN_H
int const Inf = 1000000000;
int** EmptyRoads(int** r, int n);
int** newgraph(int** rw, int** rd, int** ed, int n);
int Isolation(int** ed, int n, int A, int B);
int** ActualRoad(int** ar, int** rw, int** rd, int** ed, int n);
int* ThisIsTheWay(int* sh, int** ed, int n);
int* BuildWay(int** ed, int* sh, int n, int A, int B, int** ar, int* v);

#endif QUEEN_H
```

worker.cpp

```
#include "Queen.h"
#include <iostream>
#include <locale.h>
#include <algorithm>
#include <fstream>
using namespace std;

int main()
{
    fstream fin;
    fin.open("input.txt");
    ofstream fout;
    fout.open("out.txt");
    setlocale(LC_ALL, "ru");
       int n;
    fin >> n;
       if (n > 1)
       {
              int A = Inf, B = Inf, i, j;
              string a, b;
              string* town = new string[n]; // Заводим массив стрингов для хранения названий
городов
              for (i = 0; i < n; i++)
                     fin >> town[i];
              fin >> a >> b;
              for (i = 0; i < n; i++)
              {
```

```
                        if (a == town[i]) A = i;
                        else if (b == town[i]) B = i;
                }
                if (B == Inf || A == Inf)
                {
                        fout << "Путешествие закончилось, не успев начаться!";
                }
                else
                {
                        string typeroad, town1, town2;
                        int townind1, townind2;
                        int way;
                        int** rw = new int* [n]; //готовим таблицу смежности железных дорог
                        for (i = 0; i < n; i++)
                                rw[i] = new int[n];
                        rw = EmptyRoads(rw, n);
                        int** rd = new int* [n];//готовим таблицу смежности шоссейных дорог
                        for (i = 0; i < n; i++)
                                rd[i] = new int[n];
                        rd = EmptyRoads(rd, n);
                        while (fin) //Ввод систем дорог
                        {
                                fin >> typeroad >> town1 >> town2 >> way;
                                for (int e = 0; e < n; e++)
                                {
                                        if (town1 == town[e]) townind1 = e;
                                        else if (town2 == town[e]) townind2 = e;
                                }
                                if (typeroad == "ЖД")
                                {
                                        if (rw[townind1][townind2] == 0 && way > 0)
                                        {
                                                rw[townind1][townind2] = way;
                                                rw[townind2][townind1] = way;
                                        }
                                        else if (way < rw[townind1][townind2] && way > 0)
                                        {
                                                rw[townind1][townind2] = way;
                                                rw[townind2][townind1] = way;
                                        }
                                        else if (way < 0) fout << "Проверьте корректность входных
данных. Введено неверное расстояние между городами " << town1 << " и " << town2 << '\n';
                                }
                                else
                                {
                                        if (rd[townind1][townind2] == 0 && way > 0)
                                        {
                                                rd[townind1][townind2] = way;
                                                rd[townind2][townind1] = way;
                                        }
                                        else if (way < rd[townind1][townind2] && way > 0)
                                        {
                                                rd[townind1][townind2] = way;
                                                rd[townind2][townind1] = way;
                                        }
                                        else if (way < 0) fout << "Проверьте корректность входных
данных. Введено неверное расстояние между городами " << town1 << " и " << town2 << '\n';
                                }
                        }
                        int ctrl = 1;
                        int** ed = new int* [n]; //составляем таблицу смежности кратчайших
дорог
                        for (i = 0; i < n; i++)
                                ed[i] = new int[n];
                        ed = newgraph(rw, rd, ed, n);
                        ctrl = Isolation(ed, n, A, B);
                        if (ctrl != 0)
```

```
                    {
                            int** ar = new int* [n]; //составляем таблицу актуальны
кратчайших дорог
                            for (i = 0; i < n; i++)
                                    ar[i] = new int[n];
                            ar = ActualRoad(ar, rw, rd, ed, n);
                            int* sh = new int[n]; //Заводим массив для хранения кратчайших
расстояний от пункта A
                            for (i = 0; i < n; i++)
                                    sh[i] = Inf;
                            sh[A] = 0;
                            sh = ThisIsTheWay(sh, ed, n);
                            if (sh[B] != Inf)
                            {
                            fout << "Кратчайшее расстояние от города " << town[A] << " до
города " << town[B] << '\n';
                            fout << sh[B] << '\n';

                                    int* v = new int[n]; //Заводим массив для хранения
посещенных городов
                                    for (int i = 0; i < n; i++) v[i] = Inf;
                                    v = BuildWay(ed, sh, n, A, B, ar, v);
                                    i = 0;
                                    while (v[i] != Inf && i != n)
                                    {
                                            i++;
                                    }
                                    int k = i;
                                    fout << '\n';
                                    int act;
                                    int cnt = 0;
                                    for (i = 1; i < k - 1; i++)
                                    {
                                            act = ar[v[i - 1]][v[i]];
                                            if (act != ar[v[i]][v[i + 1]])
                                            {
                                                    fout << "Точка пересадки: " << town[v[i]] <<
'\n';
                                                    cnt++;
                                            }
                                    }
                                    if (!cnt) fout << "Точек пересадок нет!" << '\n';
                                    cout << "Программа сработала, беги смотреть результат!";
                            }
                            else fout << "Лети на самолете, город изолирован от общей системы
дорог:)" << '\n';
                    }
                    else fout << "Лети на самолете, город изолирован от общей системы
дорог:)" << '\n';
                     }
            }
            else fout << "Ошибка, введено неверное количество городов!";
        return 0;
              }
```

Brain.cpp

```
#include "Queen.h"
#include <iostream>
#include <locale.h>
#include <algorithm>
#include <fstream>
using namespace std;

int** EmptyRoads(int** r, int n)
{
       int i, j;
```

```
	for (i = 0; i < n; i++)
	{
		r[i][i] = 0;
		for (j = i + 1; j < n; j++)
		{
			r[i][j] = 0;
			r[j][i] = r[i][j];
		}
	}
	return r;
}
int** newgraph(int** rw, int** rd, int** ed, int n)
{
	int i, j;
	for (i = 0; i < n; i++)
	{
		ed[i][i] = 0;
		for (j = i + 1; j < n; j++)
		{
			if (rw[i][j] && rd[i][j])
			{
				ed[i][j] = min(rw[i][j], rd[i][j]);
				ed[j][i] = ed[i][j];
			}
			else if (rw[i][j])
			{
				ed[i][j] = rw[i][j];
				ed[j][i] = ed[i][j];
			}
			else
			{
				ed[i][j] = rd[i][j];
				ed[j][i] = ed[i][j];
			}
		}
	}
	return ed;
}
int** ActualRoad(int** ar, int** rw, int** rd, int** ed, int n)
{
	int i, j;
	for (i = 0; i < n; i++)
	{
		ar[i][i] = 0;
		for (j = i + 1; j < n; j++)
		{
			if (ed[i][j] == rw[i][j] && ed[i][j] !=0)
			{
				ar[i][j] = 1;
				ar[j][i] = ar[i][j];
			}
			else if (ed[i][j] == rd[i][j] && ed[i][j] != 0)
			{
				ar[i][j] = 2;
				ar[j][i] = ar[i][j];
			}
			else
			{
				ar[i][j] = 0;
				ar[j][i] = ar[i][j];
			}

		}
	}
	return ar;
}
int Isolation(int** ed, int n, int A, int B)
```

```
{
	int* check = new int[n];
	int i, j, ctrl = 1;
	for (i = 0; i < n; i++)
	{
		for (j = 0; j < n; j++)
		{
			check[i] += ed[i][j];
		}
		if (check[i] == 0 && (i == B || i==A))
		{
			ctrl = 0;
		}
	}
	return ctrl;
}
void showed(int** ed, int n)
{
	int i, j;
	for (i = 0; i < n; i++)
	{
		for (j = 0; j < n; j++)
		{
			cout << ed[i][j] << " ";
		}
		cout << '\n';
	}
}
int* ThisIsTheWay(int* sh, int** ed, int n)
{
	int i, j;
	int* v = new int[n]; //Заводим массив для хранения посещенных городов, 1 - посещен, 0
- нет
	for (i = 0; i < n; i++)
		v[i] = 0;
	int close, shortway; //Заводим локальные переменные отвечающие за индекс ближайшего
города, кратчайшую дорогу, соответсвенно
	do {
		close = Inf;
		shortway = Inf;
		for ( i = 0; i < n; i++)
		{
			if ((v[i] == 0) && (sh[i] < shortway)) // Проверяем, посещена ли
вершина, и является ли текущий путь до нее кратчайшим
			{
				shortway = sh[i]; //Если да, то обновляем данные
				close = i;
			}
		}
		if (close != Inf)
		{
			for ( i = 0; i < n; i++) //Проверяем, является ли новый короткий путь
до города, кратчайшим. Сравниваем с прошлым значением.
			{
				if (ed[close][i] > 0)
				{
					if (shortway + ed[close][i] < sh[i])
					{
						sh[i] = shortway + ed[close][i];
					}
				}
			}
			v[close] = 1;
		}
	} while (close < Inf);
	return sh;
}
```

```
int* BuildWay (int**ed, int*sh, int n, int A, int B, int**ar, int* v)
{
  v[0] = B; // начинаем поиск с конца
  int k = 1;
  int length = sh[B]; // Расстояние до города B

  while (B != A) // Пока не вернулись в первый город
  {
       for (int i = 0; i < n; i++) // просматриваем все города на наличия связи
         if (ed[i][B] != 0)
         {
              if (length - ed[i][B] == sh[i])// определяем длину пути из предыдыдущего
города, если длина совпала с полученной, то переход был отсюда
              {
                     length -= ed[i][B];
                B = i;
                v[k] = i;
                k++;
              }
         }
  }

  return v;
      }
```

Тесты

№	input.txt	out.txt
	Новосибирск Тюмень Вологда **Новосибирск Вологда** **ЖД Новосибирск Тюмень 150** **РД Тюмень Вологда 15**	**Ошибка, введено неверное количество городов!**

Комментарий: Проверка первого обязательного условия.

№	input.txt	out.txt
	Новосибирск Тюмень Вологда **Новосибирск Новосибирск** **ЖД Новосибирск Тюмень 150** **РД Тюмень Вологда 15**	**Путешествие окончилось, не успев начаться☺**
	Новосибирск Тюмень Сочи **Новосибирск Армавир**	**Путешествие окончилось, не успев начаться☺**

	ЖД Новосибирск Тюмень 150 **РД Тюмень Сочи 900**	

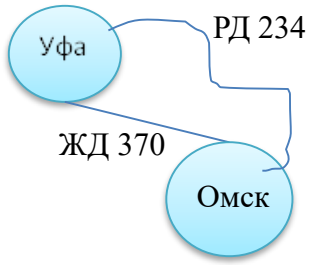
Комментарий: Проверка второго обязательного условия. Начальный и конечный пункт должны существовать, быть в перечни городов и не совпадать.

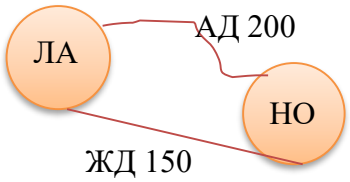
№	input.txt	out.txt
	Майами Орел Уфа Омск **Уфа Сочи** **ЖД Уфа Омск 370** **РД Омск Уфа 234** **ЖД Уфа Орел 290** **РД Омск Орел 600**	**Летите на самолете, город изолирован от общей системы дорог☺**

Комментарий: Проверка на третье обязательное условие. Изолированность городов.

№	input.txt	out.txt
	НО ЛА Уфа Омск **Уфа ЛА** **ЖД Уфа Омск 370** **РД Омск Уфа 234** **ЖД НО ЛА 150** **АД НО ЛА 200**	**Летите на самолете, город изолирован от общей системы дорог☺**

Комментарий: Проверка на третье обязательное условие, но когда города А и В находятся в разных замкнутых системах.





№	input.txt	out.txt
	Сочи Орел Уфа Омск	**Проверьте корректность входных данных. Введено неверное расстояние**

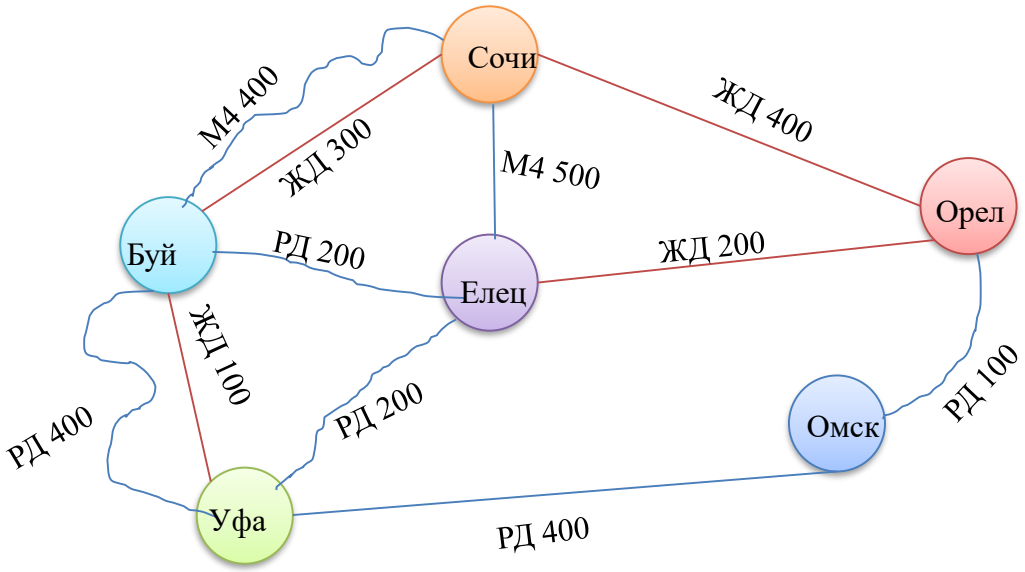
	Уфа Орел **ЖД Уфа Омск 370** **РД Омск Уфа 234** **ЖД Уфа Орел -290** **РД Омск Орел 600**	**между городами Уфа и Орел.** **Кратчайшее расстояние от города Уфа до города Орел** **Точка пересадки: Омск**

Комментарий: Проверка на чтение отрицательной длины дороги.
Вывод комментария о некорректном вводе данных.

№	input.txt	out.txt
	Сочи Буй Елец Орел Уфа Омск **Сочи Омск** **ЖД Сочи Буй 300** **ЖД Орел Сочи 400** **ЖД Буй Уфа 100** **ЖД Елец Орел 300** **М4 Сочи Буй 400** **М4 Сочи Елец 500** **М4 Буй Елец 200** **РД Буй Уфа 400** **РД Елец Уфа 200** **РД Орел Омск 100** **РД Уфа Омск 400**	**Кратчайшее расстояние от города Сочи до города Омск** **Точка пересадки: Орел**
	Сочи Буй Елец Орел Уфа Омск **Омск Буй** **ЖД Сочи Буй 300** **ЖД Орел Сочи 400** **ЖД Буй Уфа 100** **ЖД Елец Орел 300** **М4 Сочи Буй 400** **М4 Сочи Елец 500** **М4 Буй Елец 200** **РД Буй Уфа 400**	**Кратчайшее расстояние от города Сочи до города Омск** **Точка пересадки: Орел** **Точка пересадки: Елец**

	РД Елец Уфа 200 **РД Орел Омск 100** **РД Уфа Омск 400**	

Комментарий: Проверка на выборе любого города из списка за начальный пункт и конечный, а также проверка на правильность работы программы.



Список использованной литературы

Хиценко, В.П. Структуры данных и алгоритмы: методические указания к курсовой работе для 1 курса ФПМиИ (направление 010500 - Прикладная математика и информатика, специальность 010503 - Математическое обеспечение и администрирование информационных систем) дневного отделения / В.П. Хиценко, Т.А. Шапошникова. – Новосибирск : Изд-во НГТУ, 2008. – 55 с.

Беляев С. Н. Решение олимпиадных задач по программированию, учебное пособие. Красноярск, 2017