|  |  |  |
| --- | --- | --- |
| 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 | | |
| « ИТиОП» | | |
|  | | |
|  | | |
|  | | |
|  | Факультет: | Прикладной математики и информатики |
| Групп а: | ПМИ-11 |
| Студент: | Мироненко Алиса |
|  |  |
|  |  |
|  |  |
| Преподаватель: | Еланцева Ирина Леонидовна |
|  |  |
|  | | |
| Новосибирск | | |
| 2022 | | |
|  | | |

1. **Условие задачи**

В системе двусторонних дорог найти путь из города A в город B с минимальной величиной P+S, где S – сумма длин дорог, а P – сумма пошлин за проезд по ним.

1. **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 начального и конечного пункта пути (A и B).  
С четвертой строки и до конца файла содержится информация о дорогах.   
<пошлина за проезд по данной дороге> <название города 1> <название города 2> <длина данной дороги>.

*Пример входных данных:*

4

Прага Брно Острава Литомышль

Прага Острава

5 Острава Литомышль 2

2 Литомышль Прага 15

3 Прага Брно 10

4 Брно Острава 7

2.2. Результат:

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

В первой строке выходного файла out.txt выводится фраза «Наименьшая сумма длин дорог и пошлин за проезд между городами A и B» и значение минимальной суммы длин дорог и пошлин за проезд.  
Далее выводится траектория пути порядке посещения, после чего фраза «Сумма длин дорог» и значение суммы длин дорог, фраза «Сумма пошлин за проезд» и значение суммы пошлин за проезд.

В противном случае (если города A и B) совпадают, хотя бы один из городов изолирован или одного из городов нет во входных данных, программа выведет сообщение об ошибке.

Также сообщение об ошибке будет в случае получения некорректных данных (отрицательных длин дорог или пошлин за проезд).

*Пример выходных данных:*

Наименьшая сумма длин дорог и пошлин за проезд по ним между городами Прага и Острава

24

Путь в порядке посещения:

Прага -> Брно -> Острава -> FINAL

Сумма длин дорог:

17

Сумма пошлин за проезд:

7

2.3 Решение:

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

*Основные определения:*

Графом (неориентированным графом) G(V,E) называется совокупность двух множеств, где V – конечное непустое множество элементов, называемых вершинами, а E – множество неупорядоченных пар различных элементов множества V (эти пары называются ребрами).

Взвешенный граф может быть представлен своей матрицей весов W =[wij], где wij – вес ребра, соединяющего вершины i, j = 1,2, ... , m. Веса несуществующих ребер полагаются равными ∞ или 0 в зависимости от задачи. Матрица весов является простым обобщением матрицы смежности.

Мультиграф - граф, в котором может быть пара вершин, которая соединена более чем одним ребром (ненаправленным).

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 между городами, а также значения пошлин за проезд.

Несвязный граф – граф, допускающий вершину не связанную с другими вершинами ребрами. В мультиграфе вершина может быть связана сама с собой (петля).

Помеченный граф – граф, вершины которого отображают какое-либо название. В данной задаче, метками являются названия городов, считываемые с файла.

*Анализ графа:*

1. Сначала проверяем существуют ли начальный и конечный пункты маршрута среди считываемого с файла множества городов, не совпадают ли два этих пункта (петля). Во внутреннем представлении данных города будут иметь индексы от 0 до n-1, создадим переменные для хранения индексов A и B (int A, B). Изначально присваиваем этим переменным значение равное Infinity (100000), дабы значений индексов изначально не могли быть равны индексам других считанных с файла городов (при большом количестве городов). Затем считываем с файла названия всех городов в строковый массив, после чего ищем в этом массиве названия A и B. После чего уже самим переменным A и B (в программе) присваиваем соответствующие порядковые номера (индексы) в массиве названий.
2. Если одна из переменных A или B по-прежнему останется равна Infinity, то города A и B не существуют, либо совпадают, тогда программа выведет сообщение "Начальный и конечный пункты совпадают или не существуют".
3. Далее следует проверка на изолированность A и B. Сначала заполняем две весовых матрицы: 1-ая для пошлин за проезд, 2-ая для значений длин дорог между городами. После чего формируем из них единую весовую матрицу посредством суммирования 1-ой и 2-ой матрицы. Каждый элемент полученной матрицы (3) будет равен P+S, где P – пошлина за проезд, а S – длина пути. После чего ищем сумму элементов строки в полученной матрице (3). Если сумма элементов равна 0, то город является полностью изолированным от остальных. (Данный вариант может определить только полную изоляцию от других городов, но допускает вариант, когда A и B находятся в отдельных изолированных системах дорог).
4. Вариант: город соединен сам с собой петлей (свойство мультиграфа). Данное свойство никак не повлияет на корректность работы программы.

*Анализ входных данных:*

1. Проверим количество городов на положительное значение. Если количество городов будет задано неположительным значением, то в файл будет выведена фраза: “ERROR: введено некорректное 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: “ERROR: длина пути не может быть отрицательна”.

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.
4. Все длины дорог являются неотрицательными числами.

*Формальная постановка задачи:*

Во взвешенном, неориентированном, несвязном и помеченном мультиграфе найти путь из города A в город B с наименьшей суммой P+S, где P – сумма пошлин за проезд, S – сумма длин дорог.

Данную задачу можно разбить на 2 крупные подзадачи:

1 Нахождение наименьшей суммы P+S с помощью использования алгоритма Дейкстры.

2 Восстановление самого пути, нахождение суммы пошлин за проезд и суммы длин дорог.

*Алгоритм решения поставленных подзадач:*

Изначально проводится проверка на корректность полученных данных.

Если все условия выполняются, можем продолжать.

1. Нахождение кратчайшего пути.

Создаем два двухмерных массива, размерностью n\*n, хранящие матрицы весовые матрицы длин дорог и пошлин за проезд по ним. 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, пока не конец файла, и изменяем матрицы по следующему принципу: Если в ячейки матрицы соединяющая эти два города стоит начальное значение (0) и считанная длина пути (пошлина за проезд) больше 0, то изменяем матрицу смежности (две матрицы строятся по одному и тому же принципу). Если длина пути меньше нуля, выводим сообщение в файл, о том, что входные данные введены некорректно.

После заполнения данных о длинах дорог и пошлин за проезд, формируем единую матрицу (P+S), суммируя две созданные матрицы.

После заполнения матрицы (P+S), осуществляем проверку на обязательное условие об изолированности городов. Считаем сумму элементов в строке матрицы (P+S). 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 минимальные суммы P+S от стартовой точки до остальных городов. Инициализируем элементы массива, присваивая им значения бесконечности (константа Infinity), т. к. вершины не посещены, и расстояние до них ещё неизвестно. Элемент массива с индексом A помечаем нулем.

Алгоритм Дейкстры.

1. Начинаем обход с города А. Далее обходим соседние вершины по очереди. Обновляем данные о них в одномерном массиве минимальной суммы P+S от города А до других городов, теперь там будет хранится минимальная сумма P+S от текущей стартовой вершины до вершины A. Продолжаем эту операцию, пока не обойдем всех соседей. По завершении операции, ставим метку в массив посещенных городов. И переходим к следующему городу.

2. Продолжаем обход с города, который мы еще не посетили (имеется массив посещенных городов), но изменили значения кратчайшего пути до этого города (от A), т. е. от соседа А. Повторяем операцию, описанную выше, обходя всех городов-соседей, относительно нашей новой точки старта, для того чтобы уменьшить найденные суммы P+S в массиве минимальных сумм P+S.

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

После чего, осуществляем проверку, изменили ли мы значение минимальной суммы P+S от города A до города B. Если нет, то города A и B находятся в разных замкнутых системах графов. Иначе выводим в файл значение, равное кратчайшему расстоянию до пункта B – ответ на подзадачу номер 1.

4. Восстановление самого пути, нахождение суммы пошлин за проезд и суммы длин дорог.

Заполняем массив посещенных городов бесконечностями (константа Infinity). (Чтобы суммы P+S нельзя было спутать ни с каким другим значениями).

Начинаем поиск с конца. Поэтому нулевому элементу массива присваиваем индекс конечного города B. Создаем переменную отвечающее за сумму P+S и присваиваем ей значение минимальной сумме P+S от A до B (ответ на подзадачу 1). Далее запускаем цикл, в котором будем изменять индекс города B, пока он не станет равным городу A.

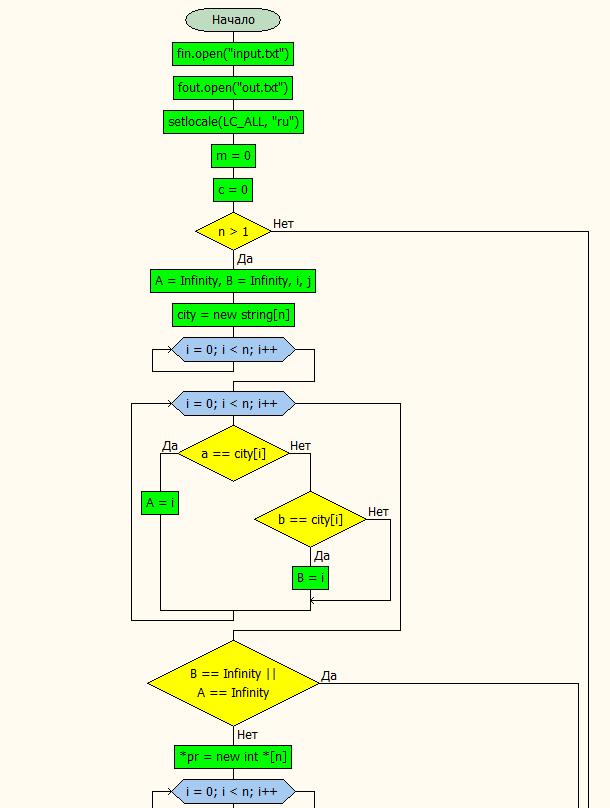
Просматриваем все города на наличие связи с городом B.

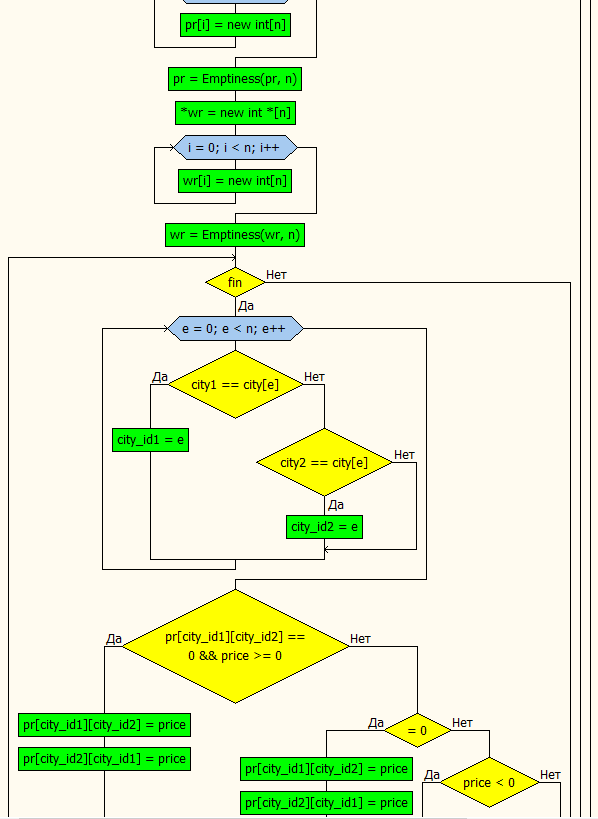
Как только связь обнаружена, то проверяем следующее условие: если разность длины от A до B и длины от B до найденного города равна длине от А до найденного города, то значит, что переход в город B был совершен с этой вершины, изменяем индекс города B на индекс города – перехода. Иначе продолжаем поиск. Таким образом, мы полностью восстанавливаем путь, записав в одномерный массив индексы посещенных городов в обратном порядке.

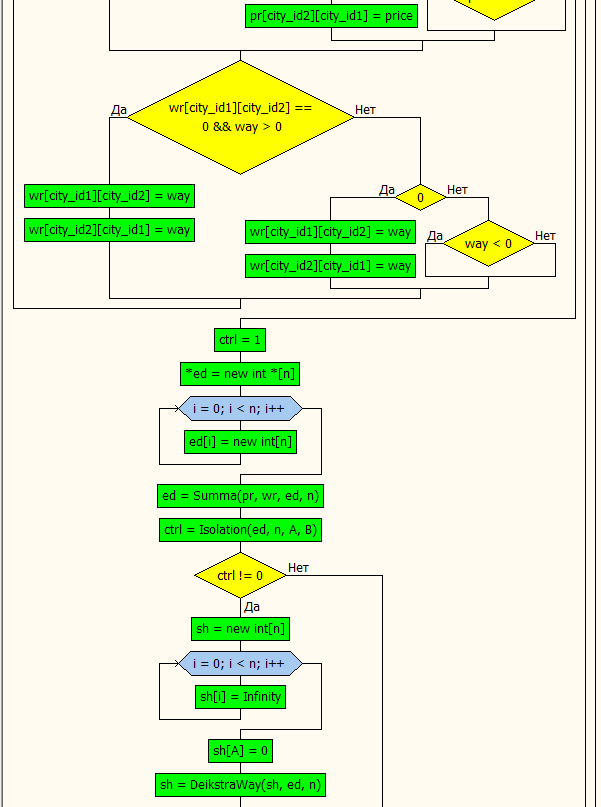
Далее находим суммы длин дорог и суммы пошлин за проезд. Сумма длин дорог находятся следующим образом. Строится цикл for, если значение в ячейке массива, в который были записаны индексы посещенных городов меньше бесконечности, то суммируем в отдельную переменную значение из массива дорог с индексами <индекс города> <индекс города - 1>. Сумма пошлин за проезд находится аналогично. Выводим все полученные данные в отдельный файл.

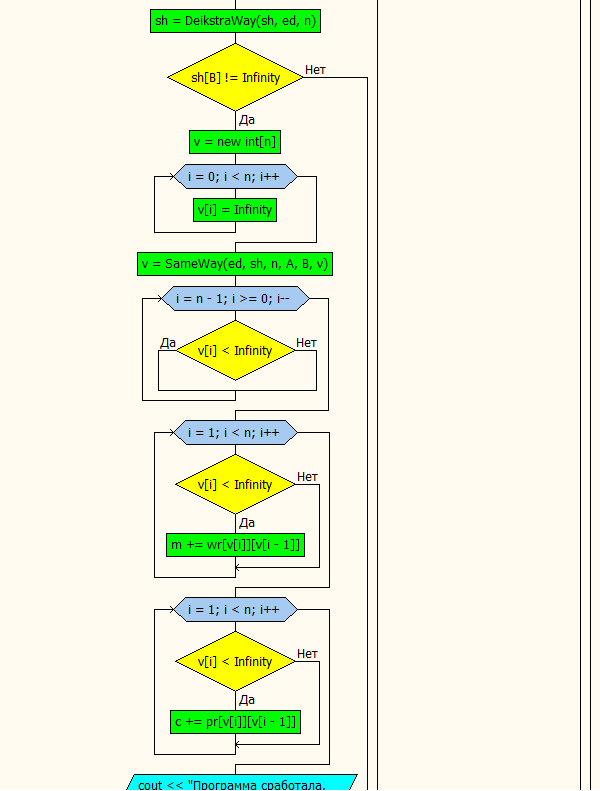
**3. Алгоритм решения задачи (блок-схема)**

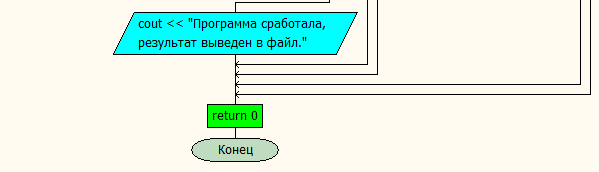
**Main.cpp**



****

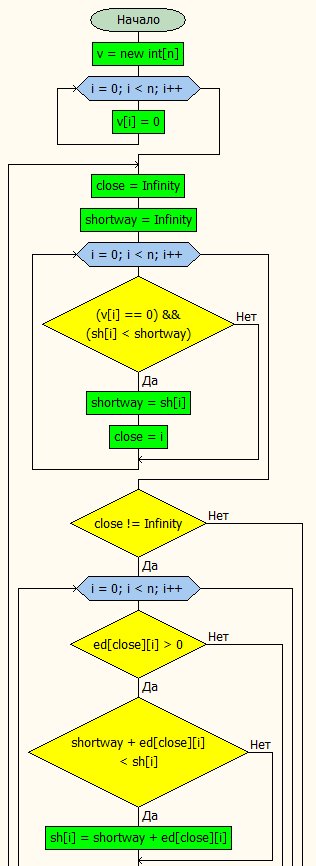
****

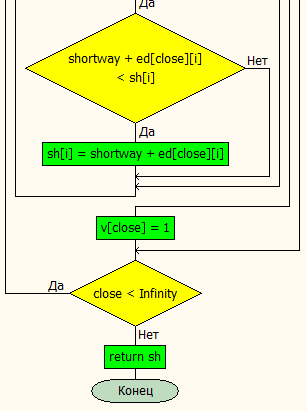
****

****

**Helper.cpp**

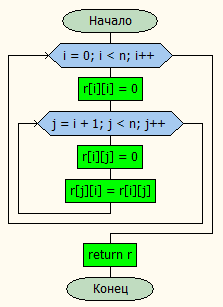
int\* DeikstraWay(int\* sh, int\*\* ed, int n) //поиск кратчайшего пути, алгоритм Дейкстры

**DeikstraWay**

****

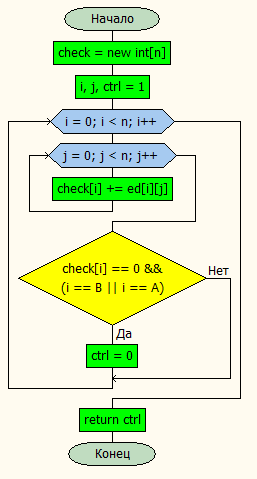
**Emptiness**

int\*\* Emptiness(int\*\* r, int n) //функция зануления массива

****

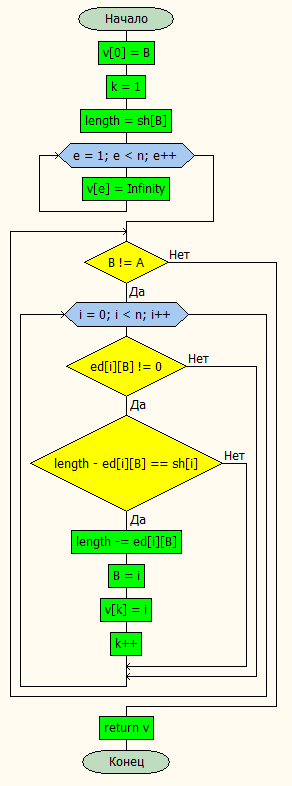
**Isolation**

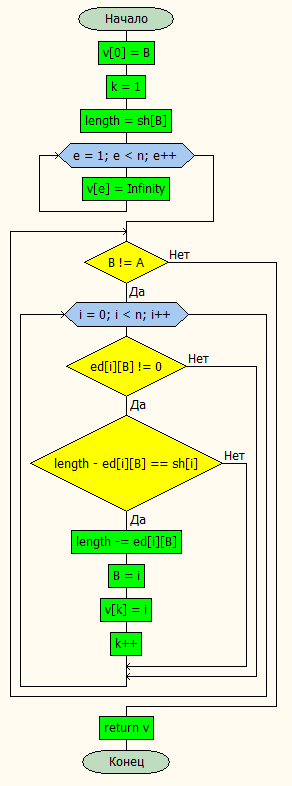
int Isolation(int\*\* ed, int n, int A, int B) //проверка A и B на изолированность

****

**SameWay**

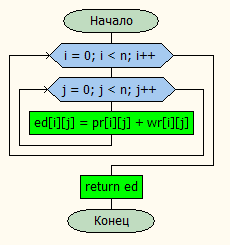
int\* SameWay(int\*\* ed, int\* sh, int n, int A, int B, int\* v) //восстановление пути

****

****

**Summa**

int\*\* Summa(int\*\* pr, int\*\* wr, int\*\* ed, int n) //создаем матрицу суммы P+S (длина дороги + пошлина за проезд по ней)

****

**4. 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

Целое число – пошлина за проезд по дороге, строковый тип - два города, между которыми существует дорога, целое число - длина дороги

*Представление выходных данных:*

Если решение задачи не было найдено, выводится:

1. «ERROR: введено некорректное количество городов» - Количество городов задано неположительным числом.

2. «Начальный пункт и конечный пункт маршрута не существуют, либо они совпадают» - предложение говорит само за себя.

3. «Город изолирован от общей системы дорог» - если один из городов (A или B) изолирован от общей системы дорог.

Если решение может быть найдено, но некорректно:

1. «ERROR: введено некорректное количество городов» - количество городов неположительно.

2. «ERROR: цена за проезд не может быть отрицательна» - цена за проезд хотя бы по одной дороге не положительна.

3. «ERROR: длина пути не может быть отрицательна» - длина пути хотя бы по одной дороге не положительна.

Если решение найдено, то выводится:

1. Наименьшая сумма длин дорог и пошлин за проезд по ним между городами A и B

<целое число>

Путь в порядке посещения:

<город> -> <город> -> <…> -> <город> -> FINAL

Сумма длин дорог:

<целое число>

Сумма пошлин за проезд:

<целое число>

Внутреннее представление данных:

*Представление входных данных:*

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

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

Создаем две строковых переменных, названия города A и название города B.

Создаем две строковых переменных для названий город 1, город 2.

Создаем целочисленные переменные для хранения значений пошлин за проезд и длин дорог.  
Целочисленная переменная, характеризующая длину пути, между городами 1 и 2.

Заводим два двухмерных массива, которые будут содержать весовые матрицы смежностей (для длин дорог и пошлин за проезд).

*Представление выходных данных:*

В случае наличия решений — в консольное окно выводится строка «Программа сработала, результат выведен в файл.». В файл выводятся следующие значения:  
минимальная сумма P+S, названия городов в порядке посещения, сумма длин дорог, сумма пошлин за проезд.

4. **Укрупненный алгоритм решения задачи**

4.1. Укрупненный алгоритм решения задачи

{ Открываем файл.

Считываем входные данные.  
Проверяем условия корректности работы программы.  
Заполняем весовые матрицы для длин дорог и для пошлин за проезд по ним.  
Формируем и заполняем единую матрицу (P+S).  
Находим наименьшую сумму P+S от точки A до точки B.  
Выводим найденное значение в файл.  
Восстанавливаем путь от B до A.   
Находим сумму пошлин за проезд.  
Находим сумму длин дорог.  
Выводим найденные данные в файл.

}

* 1. Укрупненный алгоритм нахождения минимальной суммы P+S

{

Заводим одномерный массив для хранения кратчайших расстояний от точки A.Заполняем его бесконечностями. Элементу с индексом A 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 A в массиве посещений как посещенный, и продолжаем обход, начиная с города-соседа, также обновляя данные о минимальных суммах P+S.

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

}

}

5. **Структура программы**

Программа состоит из 3-х файлов: Mainer.cpp (основная программа), Header.h – файл заголовков, Helper.cpp – файл со вспомогательными функциями.

Состав файла Helper.cpp

**Функция Emptiness**

int\*\* Emptiness(int\*\* r, int n) //функция зануления массива

Смысл: заполнение массива нулями.

Аргументы:

r – будущая весовая матрица

n – количество городов (размерность массива)

**Функция Summa**

int\*\* Summa(int\*\* pr, int\*\* wr, int\*\* ed, int n) //создаем матрицу суммы P+S (длина дороги + пошлина за проезд по ней)

Смысл: суммирование двух массивов (формирование матрицы P+S)

Аргументы:

pr – матрица пошлин за проезд

wr – матрица длин дорог

ed – формируемая матрица

n – количество городов (размерность массива)

**Функция Isolation**

int Isolation(int\*\* ed, int n, int A, int B) //проверка A и B на изолированность

Смысл: проверка городов A и B на изолированность (отсутствие в принципе дорог к другим городам) или на совпадение A и B

Аргументы:

ed – матрица P+S

n – количество городов (размерность массива)

A – индекс города А

B – индекс города В

Функция DeikstraWay

int\* DeikstraWay(int\* sh, int\*\* ed, int n)

Смысл: поиск минимального значения P+S посредством алгоритма Дейкстры

Аргументы:

sh – массив для записи минимальных сумм P+S

ed – матрица P+S

n – количество городов (размерность массива)

**Функция SameWay**

int\* SameWay(int\*\* ed, int\* sh, int n, int A, int B, int\* v) //восстановление пути

Смысл: восстановление пути, формирование массива с индексами городов в обратном порядке посещения (остальные значения массива – бесконечности)

Аргументы:

ed – матрица P+S

sh – одномерный массив минимальных сумм P+S

n – количество городов (размерность массива)

A – индекс города А

B – индекс города В

v – массив для записи индексов посещенных городов в обратном порядке

1. **Код программы на языке Си (С++)**

**Header.h**

#pragma once

#ifndef HEADER\_H

#define HEADER\_H

int const Infinity = 100000;

int\*\* Emptiness(int\*\* r, int n);

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

int\* DeikstraWay(int\* sh, int\*\* ed, int n);

int\* SameWay(int\*\* ed, int\* sh, int n, int A, int B, int\* v);

int\*\* Summa(int\*\* rw, int\*\* rd, int\*\* ed, int n);

#endif HEADER\_H

**Helper.cpp**

#include "Header.h"

#include <iostream>

#include <locale.h>

#include <algorithm>

#include <fstream> //библиотека для работы со строками

using namespace std;

int\*\* Emptiness(int\*\* r, int n) //функция зануления массива

{

int i, j; //итераторы создаем

for (i = 0; i < n; i++)

{

r[i][i] = 0; //заполняем диагональные элементы нулями

for (j = i + 1; j < n; j++)

{

r[i][j] = 0;

r[j][i] = r[i][j]; //симметрично\* заполняем нулями

}

}

return r;

}

int\*\* Summa(int\*\* pr, int\*\* wr, int\*\* ed, int n) //создаем матрицу суммы P+S (длина дороги + пошлина за проезд по ней)

{

int i, j;

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

ed[i][j] = pr[i][j] + wr[i][j];

}

}

return ed;

}

int Isolation(int\*\* ed, int n, int A, int B) //проверка A и B на изолированность

{

int\* check = new int[n]; //создаем указатель на новый одномерный массив размера n

int i, j, ctrl = 1; //создаем итераторы и переменную определитель изолированности (0 - изолированы, 1 - нет)

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

check[i] += ed[i][j]; //записываем в ячейку сумму элементов матрицы смежности по строке

}

if (check[i] == 0 && (i == B || i == A)) //если сумма элементов = 0 (из A или B нет дороги ни в один город) или их индексы совпадают (значит, это 1 город)

{

ctrl = 0;

}

}

return ctrl;

}

int\* DeikstraWay(int\* sh, int\*\* ed, int n) //поиск кратчайшего пути, алгоритм Дейкстры

{

int i, j; //создаем итераторы

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

do {

close = Infinity;

shortway = Infinity;

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]; //eсли да, то обновляем данные

close = i;

}

}

if (close != Infinity)

{

for (i = 0; i < n; i++) //проверяем, является ли новый короткий путь до города, кратчайшим. Сравниваем с предыдущим значением.

{

if (ed[close][i] > 0)

{

if (shortway + ed[close][i] < sh[i])

{

sh[i] = shortway + ed[close][i];

}

}

}

v[close] = 1;

}

} while (close < Infinity); //пока индекс ближайшего города меньше бесконечности

return sh;

}

int\* SameWay(int\*\* ed, int\* sh, int n, int A, int B, int\* v) //восстановление пути

{

v[0] = B; // начинаем поиск с конца

int k = 1;

int length = sh[B]; // Расстояние до города В

for (int e = 1; e < n; e++) {

v[e] = Infinity;

}

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; //массив, состоящий из индексов городов в обратном порядке посещения

}

**Mainer.cpp**

#include "Header.h"

#include <iostream>

#include <locale.h>

#include <algorithm>

#include <fstream> //библиотека для работы со строками

using namespace std;

int main()

{

fstream fin;

fin.open("input.txt"); //открываем файл с исходными данными

ofstream fout; //полключаем файловый вывод

fout.open("out.txt"); //файловый вывод в out.txt

setlocale(LC\_ALL, "ru"); //подключаем библиотеку русского языка

int n; //создаем переменную для количества городов

int m = 0; //переменная для вывода суммы длин дорог

int c = 0; //переменная для вывода суммы пошлн за проезд

fin >> n; //считываем первую строку файла в переменную n

if (n > 1) //если количество городов > 1

//------------------решение первой подзадачи: поиск значения минимальной суммы P+S-------------------

{

int A = Infinity, B = Infinity, i, j; //присваиваем начальному и конечному пунктам значения беск., чтобы они не совпадали с существ. индексами городов, создаем итераторы

string a, b; //создаем переменные строкового типа для хранения названий начального и конечного пункта

string\* city = new string[n]; // Заводим строковый одномерный массив для хранения названий городов

for (i = 0; i < n; i++)

fin >> city[i]; //считываем со второй строки файла названия городов

fin >> a >> b; //с 3й строки файла считываем названия A и B

for (i = 0; i < n; i++) //проходим по массиву с названиями городов

{

if (a == city[i]) A = i; //присваиваем А порядковый индекс в массиве городов

else if (b == city[i]) B = i; //присваиваем B порядковый индекс в массиве городов

}

if (B == Infinity || A == Infinity)

{

fout << "Начальный и конечный пункты совпадают или не существуют";

}

else //A и B существуют и не совпадают

{

int price;

string city1, city2;

int city\_id1, city\_id2; //переменные для индексов городов A и B

int way;

int\*\* pr = new int\* [n]; //готовим таблицу смежности для пошлин за проезд (весовая матрица)

for (i = 0; i < n; i++)

pr[i] = new int[n];

pr = Emptiness(pr, n);

int\*\* wr = new int\* [n];//готовим таблицу смежности для длин дорог (весовая матрица)

for (i = 0; i < n; i++)

wr[i] = new int[n];

wr = Emptiness(wr, n);

while (fin) //считывание и ввод систем дорог

{

fin >> price >> city1 >> city2 >> way;

for (int e = 0; e < n; e++)

{

if (city1 == city[e]) city\_id1 = e;

else if (city2 == city[e]) city\_id2 = e;

}

//создали весовую матрицу для цен за проезд по дорогам

if (pr[city\_id1][city\_id2] == 0 && price >= 0) //между A и B нет прямой дороги и цена за проезд > 0

{

pr[city\_id1][city\_id2] = price;

pr[city\_id2][city\_id1] = price;

}

else if (price < pr[city\_id1][city\_id2] && price >= 0)

{

pr[city\_id1][city\_id2] = price;

pr[city\_id2][city\_id1] = price;

}

else if (price < 0) {

fout << "ERROR: цена за проезд не может быть отрицательна, города: " << city1 << " и " << city2 << '\n';

return 0;

}

//создали весовую матрицу для длин путей

if (wr[city\_id1][city\_id2] == 0 && way > 0) //между A и B нет прямой дороги и длина пути > 0

{

wr[city\_id1][city\_id2] = way;

wr[city\_id2][city\_id1] = way;

}

else if (way < wr[city\_id1][city\_id2] && way > 0)

{

wr[city\_id1][city\_id2] = way;

wr[city\_id2][city\_id1] = way;

}

else if (way < 0) {

fout << "ERROR: длина пути не может быть отрицательна, города: " << city1 << " и " << city2 << '\n';

return 0;

}

}

int ctrl = 1;

int\*\* ed = new int\* [n]; //составляем таблицу суммы P+S, пошлина за проезд + длина дороги

for (i = 0; i < n; i++)

ed[i] = new int[n];

ed = Summa(pr, wr, ed, n);

ctrl = Isolation(ed, n, A, B);

if (ctrl != 0) //если города не изолированы

{

int\* sh = new int[n]; //Заводим массив для хранения кратчайших расстояний от пункта A, подготовка к исп. алг. Дейкстры

for (i = 0; i < n; i++)

sh[i] = Infinity;

sh[A] = 0;

sh = DeikstraWay(sh, ed, n); //используем алгоритм Дейкстры

if (sh[B] != Infinity) //если расстояние от A до B не равно oo

{

fout << "Наименьшая сумма длин дорог и пошлин за проезд по ним между городами " << city[A] << " и " << city[B] << '\n';

fout << sh[B] << '\n';

//----------решение подзадачи 2: поиск самого пути

int\* v = new int[n]; //Заводим массив для хранения посещенных городов

for (int i = 0; i < n; i++) v[i] = Infinity;

v = SameWay(ed, sh, n, A, B, v); //поиск самого пути как траектории

fout << "Путь в порядке посещения:" << '\n';

//вывод названий городов (массив в обратном порядке)

for (int i = n - 1; i >= 0; i--) {

if (v[i] < Infinity) {

fout << city[v[i]];

fout << " -> ";

}

}

fout << "FINAL" << '\n';

//вывод длин дорог

fout << "Сумма длин дорог:" << '\n';

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

if (v[i] < Infinity) {

m += wr[v[i]][v[i - 1]];

}

}

fout << m << '\n';

//вывод суммы пошлин за проезд

fout << "Сумма пошлин за проезд:" << '\n';

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

if (v[i] < Infinity) {

c += pr[v[i]][v[i - 1]];

}

}

fout << c << '\n';

cout << "Программа сработала, результат выведен в файл.";

}

else fout << "Город изолирован от общей системы дорог" << '\n';

}

else fout << "Город изолирован от общей системы дорог" << '\n';

}

}

else fout << "ERROR: введено некорректное количество городов";

return 0;

1. }**Набор тестов**

**1: Проверка на положительность количества городов.**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| -3  Венгерберг Марибор Элландер  Венгерберг Элландер  1 Венгерберг Марибор 5  2 Венгерберг Элландер 6  3 Марибор Элландер 7 | ERROR: введено некорректное количество городов |

Комментарий: программа сработала корректно.

**2: Проверка совпадение А и В**

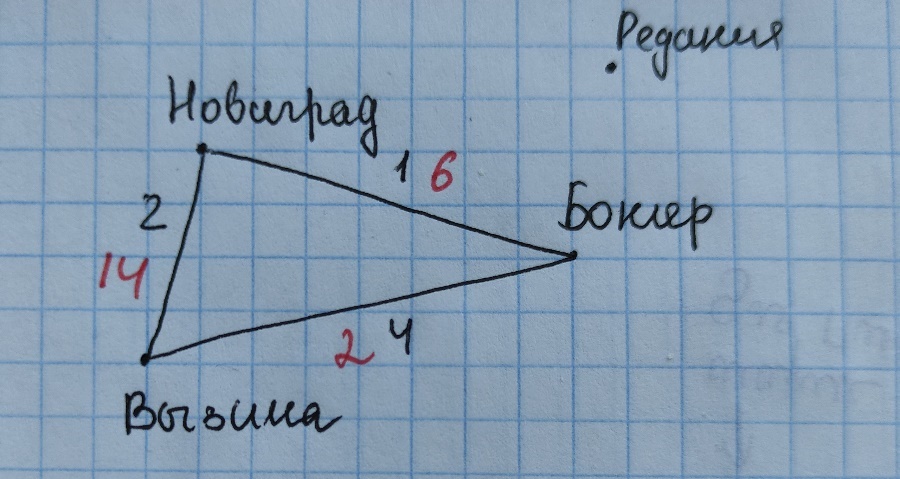
|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 4  Вызима Ривия Гвиндеф Альдерберг  Вызима Вызима  1 Вызима Ривия 6  2 Вызима Альдерберг 14  4 Гвиндеф Ривия 2  5 Гвиндеф Альдерьерг 17 | Начальный и конечный пункты совпадают или не существуют |

Комментарий: программа сработала корректно.

**3: Проверка на изолированность A от общей системы дорог**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 4  Вызима Редания Боклер Новиград  Вызима Редания  1 Новиград Боклер 6  2 Вызима Новиград 14  4 Вызима Боклер 2 | Город изолирован от общей системы дорог |

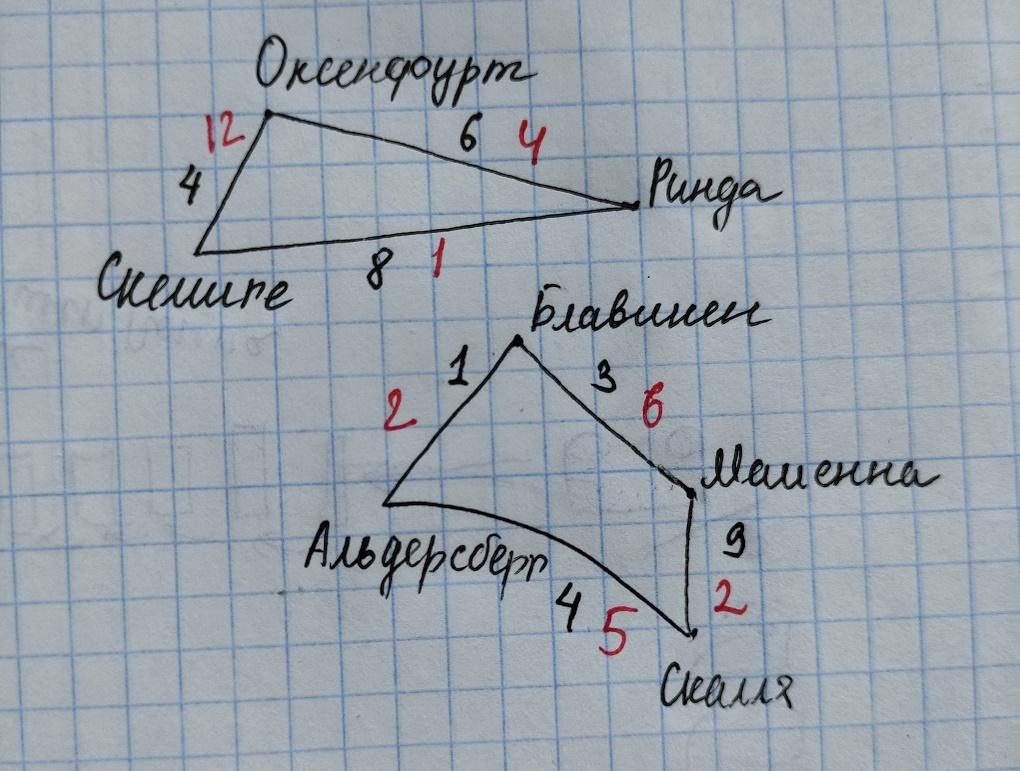
Комментарий: программа сработала корректно.



**4: Проверка на нахождение А и В в двух системах изолированных дорог**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 7  Оксенфурт Ринда Скеллиге Блавикен Альдерсберг Скалля Маиенна  Блавикен Оксенфурт  6 Оксенфурт Ринда 4  4 Оксенфурт Скеллиге 12  8 Скеллиге Ринда 1  3 Блавикен Маиенна 6  1 Блавикен Альдерсберг 2  9 Скалля Маиенна 2  4 Скалля Альдерсберг 5 | Город изолирован от общей системы дорог |

Комментарий: программа сработала корректно.



**4: Проверка на отсутствие одного из городов (A и B) в файле**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 4  Вызима Редания Боклер Новиград  Вызима Скеллиге  1 Новиград Боклер 6  2 Вызима Новиград 14  4 Вызима Боклер 2 | Начальный и конечный пункты совпадают или не существуют |

Комментарий: программа сработала корректно

**5: Проверка на положительное значение длины дороги**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 4  Вызима Редания Боклер Новиград  Вызима Новиград  1 Новиград Боклер 6  2 Вызима Новиград -14  4 Вызима Боклер 2 | ERROR: длина пути не может быть отрицательна, города: Вызима и Новиград |

Комментарий: программа сработала корректно

**6: Проверка на положительное значение длин дорог (2)**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 4  Вызима Редания Боклер Новиград  Вызима Новиград  1 Новиград Боклер -6  2 Вызима Новиград -14  4 Вызима Боклер 2 | ERROR: длина пути не может быть отрицательна, города: Новиград и Боклер |

Комментарий: программа сработала корректно, в случае, когда в файле имеется несколько значений отрицательных длин дорог, программа выведет города, между которыми первыми была встречена неположительная дорога (аналогично с пошлинами за проезд)

**7: Проверка на положительное значение пошлин за проезд**

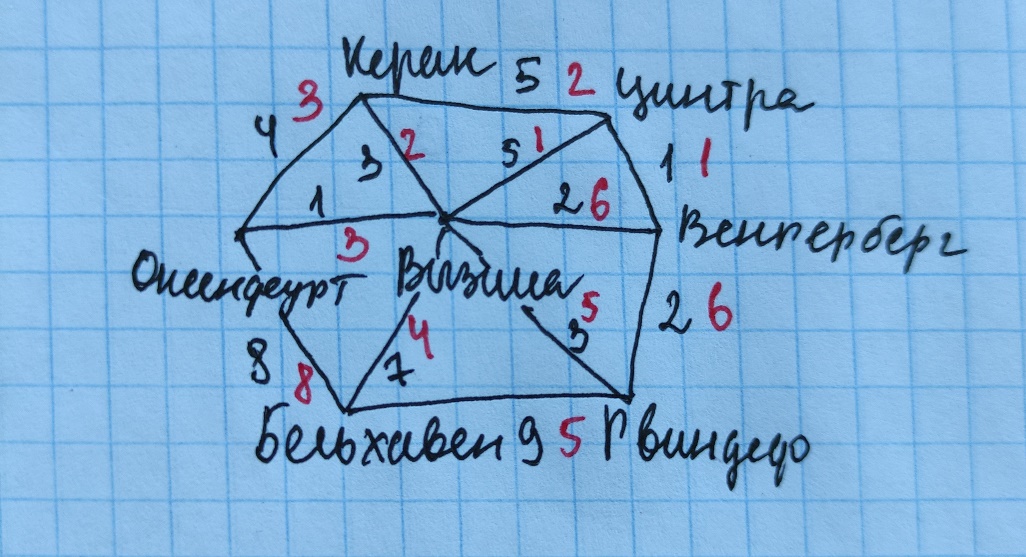
|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 4  Вызима Редания Боклер Новиград  Вызима Новиград  1 Новиград Боклер 6  -2 Вызима Новиград 14  4 Вызима Боклер 2 | ERROR: цена за проезд не может быть отрицательна, города: Вызима и Новиград |

Комментарий: программа сработала корректно

**8: Проверка верного нахождения суммы P+S и траектории пути**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 7  Цинтра Керак Венгерберг Гвиндеф Бельхавен Оксенфурт Вызима  Цинтра Оксенфурт  3 Вызима Керак 2  5 Вызима Цинтра 1  2 Вызима Венгерберг 6  3 Вызима Гвиндеф 5  7 Вызима Бельхавен 4  1 Вызима Оксенфурт 3  5 Керак Цинтра 2  1 Цинтра Венгерберг 1  2 Венгерберг Гвиндеф 6  9 Гвиндеф Бельхавен 5  8 Бельхавен Оксенфурт 8  4 Оксенфурт Керак 3 | Наименьшая сумма длин дорог и пошлин за проезд по ним между городами Цинтра и Оксенфурт  10  Путь в порядке посещения:  Цинтра -> Вызима -> Оксенфурт -> FINAL  Сумма длин дорог:  4  Сумма пошлин за проезд:  6 |

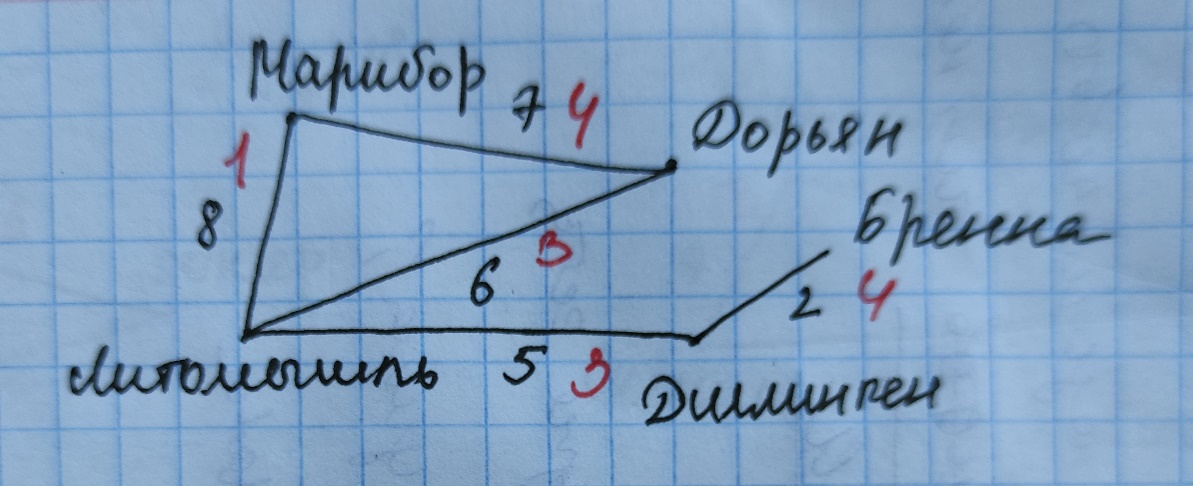
Комментарий: программа сработала корректно



**9: Проверка верного нахождения суммы P+S и траектории пути**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 5  Бренна Марибор Дорьян Литомышль Диллинген  Бренна Литомышль  7 Марибор Дорьян 4  8 Марибор Литомышль 1  6 Дорьян Литомышль 3  5 Литомышль Диллинген 3  2 Диллинген Бренна 4 | Наименьшая сумма длин дорог и пошлин за проезд по ним между городами Бренна и Литомышль  14  Путь в порядке посещения:  Бренна -> Диллинген -> Литомышль -> FINAL  Сумма длин дорог:  7  Сумма пошлин за проезд:  7 |

Комментарий: программа сработала корректно



**10: Проверка верного нахождения суммы P+S и траектории пути**

|  |  |
| --- | --- |
| **input.txt** | **out.txt** |
| 4  Литомышль Ривия Цинтра Лирия  Лирия Ривия  2 Литомышль Лирия 6  3 Литомышль Ривия 5  4 Ривия Цинтра 3  5 Цинтра Лирия 2 | Наименьшая сумма длин дорог и пошлин за проезд по ним между городами Лирия и Ривия  14  Путь в порядке посещения:  Лирия -> Цинтра -> Ривия -> FINAL  Сумма длин дорог:  5  Сумма пошлин за проезд:  9 |

Комментарий: программа сработала корректно

Изображение выглядит как текст

Автоматически созданное описание

1. **Результат работы программы**

На всех тестах программа выдала ожидаемый результат, следовательно, задача решена верно.

**Список литературы**

1. 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.
2. Методические указания к практическим занятиям и выполнению РГР по курсу «Дискретная математика» - М.Э. Рояк, канд. техн. наук, доц. С.Х. Рояк, ассист., 1998.