МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

ТЕМА: «АЛГОРИТМ ФЛОЙДА-УОРШЕЛЛА»

Студентка гр. 8382 _____ Кузина А.М.

Студентка гр. 8382 _____ Кулачкова М.К.

Студентка гр. 8382 _____ Рочева А.К.

Руководитель _____ Фирсов М.А.

Санкт-Петербург
2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Кузина А.М. группы 8382
Студентка Кулачкова М.К. группы 8382
Студентка Рочева А.К. группы 8382

Тема практики: алгоритм Флойда-Уоршелла

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: Флойда-Уоршелла.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 01.07.2020

Дата защиты отчета: 00.07.2020

	_____	
Студентка	_____	Кузина А.М.
Студентка	_____	Кулачкова М.К.
Студентка	_____	Рочева А.К.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

Целью учебной практики является разработка приложения для визуализации алгоритма Флойда-Уоршелла. Приложение создается на языке Java и должно обладать графическим интерфейсом. Пользователю должна быть предоставлена возможность отрисовки используемых структур данных (графа и соответствующей матрицы смежности), а также пошагового выполнения алгоритма с пояснениями. Приложение должно быть понятным и удобным для использования.

Задание выполняется командой из трех человек, за которыми закреплены определенные роли. Выполнение работы и составление отчета осуществляются поэтапно.

SUMMARY

The purpose of training practice is to create an application which would visualize the Floyd-Warshall algorithm. The application should be written in Java programming language and must implement a graphical user interface. The user must be provided with possibilities to view data structures in use (the graph and the respective adjacency matrix) and the step-by-step execution of the algorithm with commentaries. The application must be transparent and handy.

The task is fulfilled by a team of three members, each of them assigned with certain obligations. Implementation of the task and report composition should be gradual.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью учебной практики является создание приложения, визуализирующего работу алгоритма Флойда-Уоршелла, предназначенного для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа. Приложение должно быть написано на языке Java и снабжено понятным и удобным в использовании графическим интерфейсом. Пользователю должна быть предоставлена возможность ввести исходные данные в самой программе с клавиатуры или загрузить их из файла. Результат работы алгоритма также должен выводиться на экран и по требованию сохраняться в файл. Должна быть предоставлена возможность как моментального отображения результата, так и визуализации пошагового выполнения алгоритма.

Задание выполняется командой из трех человек, за каждым из которых закреплены определенные обязанности – реализация графического интерфейса, логики алгоритма, проведение тестирование и сборка проекта. Готовая программа должна корректно собираться из исходников в один исполняемый jar-архив. В ходе сборки должны выполняться модульные тесты и завершаться успехом. Также на момент завершения практики должен быть составлен подробный отчет, содержащий моделирование программы, описание алгоритмов и структур данных, план тестирования, исходный код и др.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Требования к вводу исходных данных

Исходными данными для реализуемого приложения является граф, в котором будет осуществляться поиск путей. Граф задается списком ребер в формате $v_i v_j w_{ij}$, где v_i, v_j – смежные вершины, w_{ij} – вес (длина) ребра между ними. Необходимо предоставить пользователю возможность ввода исходных данных как с клавиатуры в самой программе, так и из текстового файла.

1.2. Требования к выводу результата

Результат выполнения алгоритма должен выводиться на экран в виде таблицы, а также сохраняться в текстовый файл по требованию пользователя.

1.3. Требования к визуализации

Необходимо реализовать удобный и понятный пользователю графический интерфейс. Должна быть предоставлена возможность отрисовки заданного графа, выполнение алгоритма по требованию пользователя необходимо осуществлять моментально с выводом результата или пошагово. При пошаговом выполнении алгоритма каждый этап должен быть снабжен пояснениями.

На рисунке 1 изображена диаграмма прецедентов проекта, описывающая функционал программы.

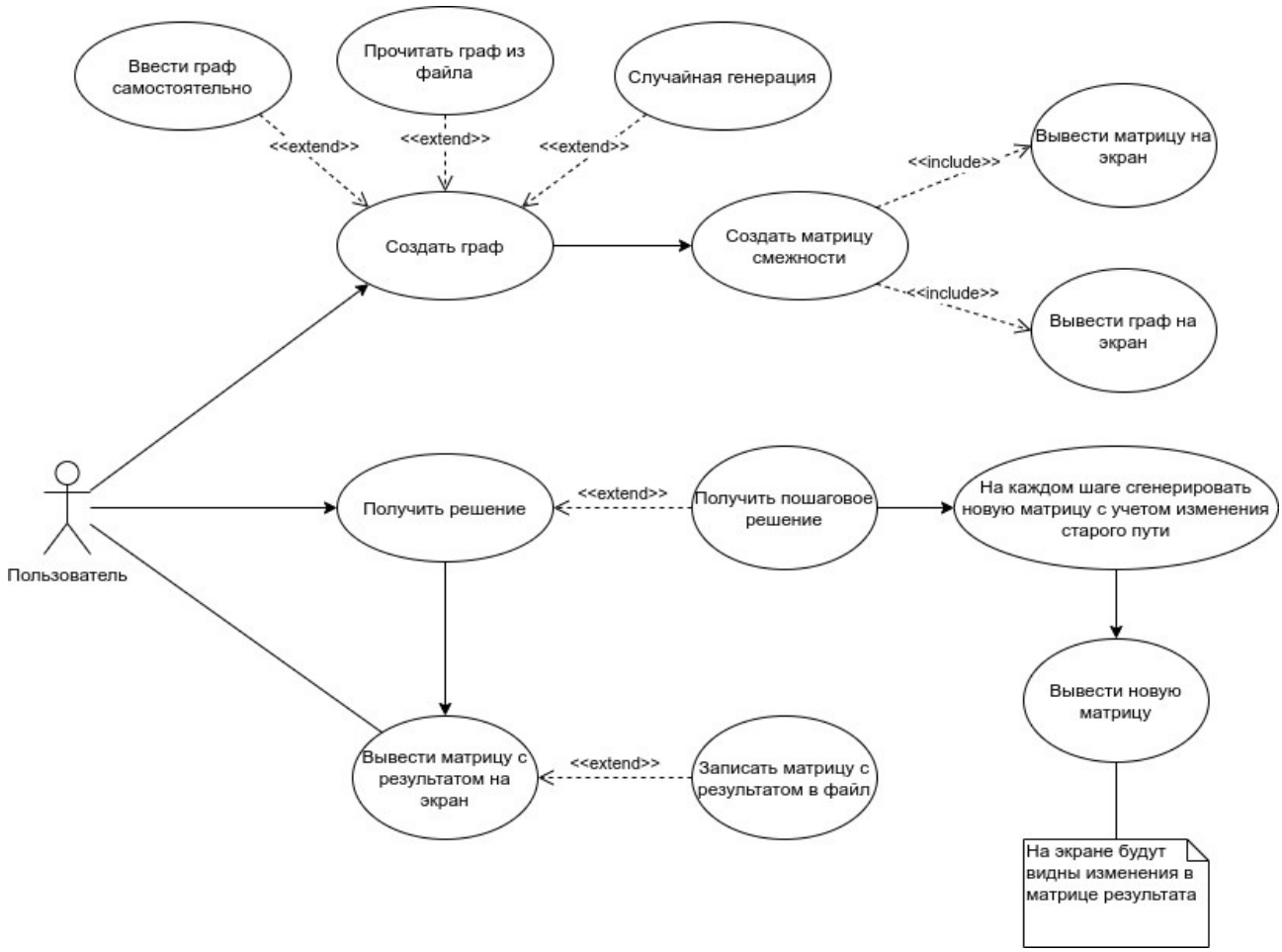

Рисунок 1 - Диаграмма прецедентов

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

К 02.07.2020 должны быть распределены роли между членами бригады, составлена диаграмма прецедентов программы, а также создана директория с исходным кодом и скриптом сборки.

К 04.07.2020 должны быть размещены все элементы интерфейса, составлены UML-диаграмма классов программы с пояснениями, а также UML-диаграмма состояний программы.

К 06.07.2020 необходимо сделать случайную генерацию изначальных графов с проверкой корректности вводимых данных, решение алгоритма при нажатии на кнопку графического интерфейса с отображением конечного результата работы алгоритма, а также добавить в отчет описание алгоритма и план тестирования.

К 08.07.2020 должна быть добавлена возможность визуализации пошагового выполнения алгоритма, должны быть сделаны тесты для созданных структур данных и функций алгоритма согласно плану тестирования, в отчет добавлено описание алгоритма пошагового отображения работы алгоритма.

К 10.07.2020 проект должен быть полностью готов, программа должна корректно собираться, в ходе сборки должны выполняться и успешно завершаться модульные тесты.

2.2. Распределение ролей в бригаде

Кузина А.М. отвечает за разработку графического интерфейса.

Кулачкова М.К. отвечает за реализацию логики алгоритма.

Рочева А.К. отвечает за тестирование и сборку приложения.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Алгоритм Флойда-Уоршелла, реализуемый в программе, предназначен для обработки графа. Хранение графа осуществляется при помощи класса **Graph**. Граф включает в себя массив вершин, которые представляют собой объекты класса **Vertex**, и матрицу смежности – двойной массив связок между вершинами, которые хранятся в виде объектов класса **Connection**. Класс **Vertex** состоит из поля с именем вершины, которое не может быть изменено в ходе работы приложения, и метода, возвращающего имя вершины. Класс **Connection** хранит вес ребра, соединяющего вершины, или -1, если такого ребра нет, длину кратчайшего пути между вершинами и строку, содержащую сам путь. Класс также содержит методы, возвращающие значения приватных полей, и метод, обновляющий кратчайший путь и его длину.

Объект класса **Graph** создается и хранится в основном классе программы – классе **App**. К нему же привязан графический интерфейс. Чтение исходных данных для создания графа может осуществляться как из файла, так и с клавиатуры. Вывод результата обработки графа также может осуществляться как в файл, так и на экран. В связи с этим создаются классы **FileGraphIO** и **ScreenGraphIO**, которые реализуют интерфейс **GraphIO**, производящий ввод/вывод данных. Эти классы осуществляют взаимосвязь между пользовательским интерфейсом и графом.

На рисунке 2 изображена UML-диаграмма классов программы. Она будет дополнена в ходе дальнейшей работы над проектом.

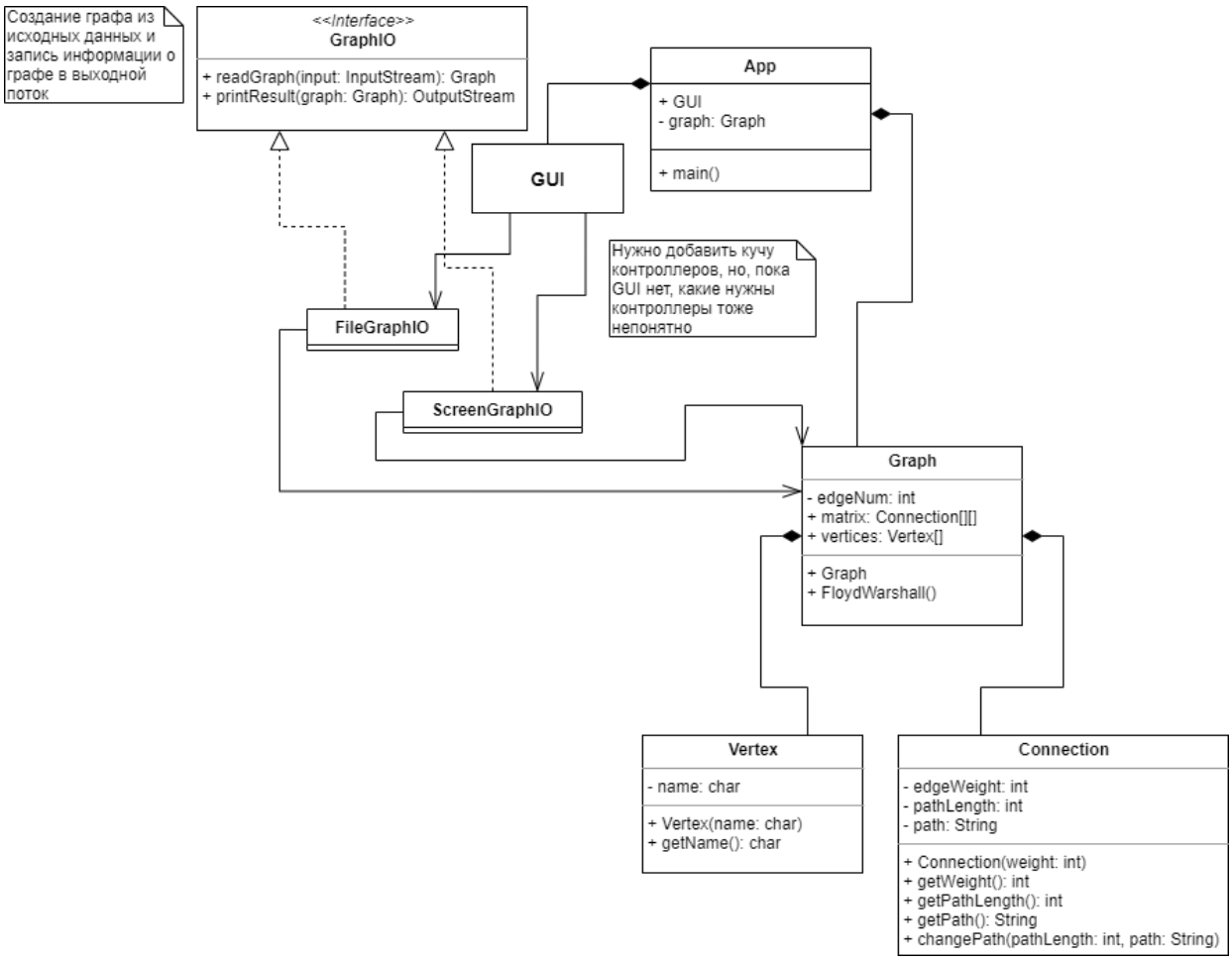

Рисунок 2 - Диаграмма классов

3.2. Описание алгоритма

Алгоритм Флойда-Уоршелла реализован в методе *FloydWarshall* класса **Graph**. Исходный код алгоритма *пока не* представлен в приложении А. На каждом шаге метод изменяет матрицу, которая содержит длины кратчайших путей между всеми вершинами.

Алгоритм содержит три цикла, в которых обходятся все вершины графа. В двух внутренних циклах рассматриваются ячейки матрицы кратчайших путей, и текущий кратчайший путь из одной вершины в другую сравнивается с путем, проходящим через вершину, рассматриваемую во внешнем цикле, т.е. суммой путей из начальной вершины во внешнюю и из внешней в конечную. Если путь через внешнюю вершину короче текущего пути, в матрице кратчайших путей изменяется кратчайший путь между вершинами.

3.3. Основные методы

На рисунке 3 представлена диаграмма состояний программы (рисунок находится в diagrams/states).

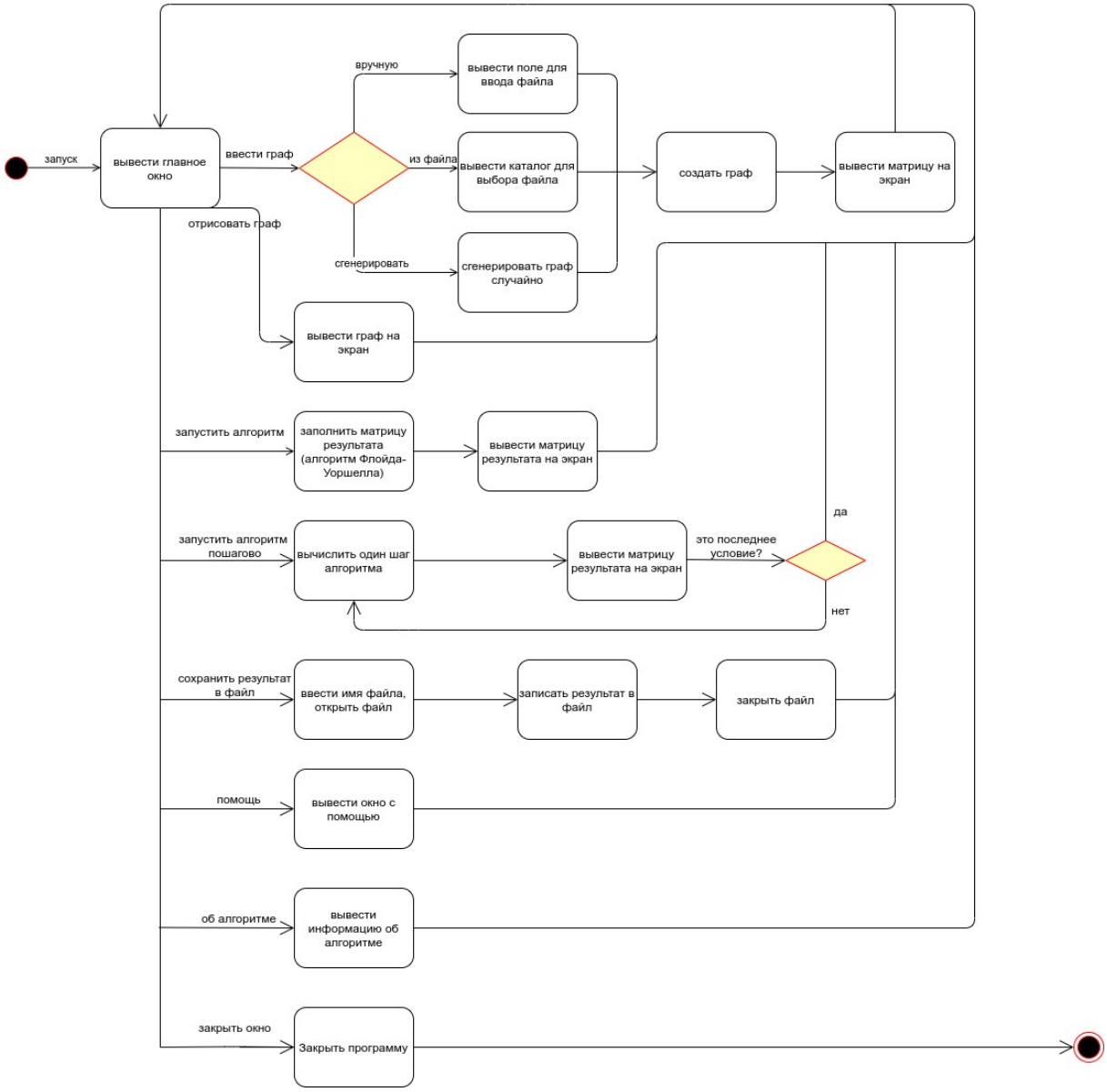

Рисунок 3 — Диаграмма состояний

4. ТЕСТИРОВАНИЕ

4.1. План тестирования программы

1. **Вступление**

   Объектом тестирования является программа для визуализации работы алгоритма Флойда-Уоршелла по поиску кратчайших путей в графе.

2. **Функционал, который будет протестирован**

   - Случайная генерация графа
   - Создание графа по введенным данным
   - Создание матрицы смежности по введенным данным
   - Работа алгоритма

3. **Подход к тестированию**

   Уровень тестирования: модульное

   Специальные средства тестирования: тестирование будет проходить с помощью фреймворка автоматического тестирования JUnit.

4. **Критерии успешности тестирования**

   Программа считается законченной, если все разработанные тесты выполняются без ошибок

5. **Критерии прекращения тестирования**

   Программа возвращается на доработку, если хотя бы один из тестов обнаружил ошибку. После исправления ошибки программа снова передается на тестирование.

4.2. Тестовые случаи

1. Случайная генерация графа

На вход подается количество вершин в будущем графе. Тестируется метод generateRandom(int numVertices) класса GraphGenerator. На выходе ожидается объект класса Graph с количеством вершин, равным числу вершин на входе, и с случайно сгенерированным количеством ребер (каждая вершина должна быть инцидентна минимум одному ребру и максимум (numVertices-1)

числу ребер, ребро из каждой вершины должно вести в отличную от нее вершину).

Что подается на вход	Тестируемый метод	Что ожидается на выходе
Число, меньшее единицы	public Graph generateRandom(int numVertices)	Исключение
Число, большее значения в константе maxVertices	public Graph generateRandom(int numVertices)	Исключение
Единица	public Graph generateRandom(int numVertices)	Граф с одной вершиной и с нулевым количеством ребер
Другое число	public Graph generateRandom(int numVertices)	Граф с полученным на вход количеством вершин и со случайным набором ребер. Ребра должны удовлетворять ранее описанным условиям

2. Создание графа по введенным данным

Тестируются методы класса Graph (проверяется правильность структуры). На вход подается строка, содержащая описания ребер. Тестируются работа конструктора класса Graph (косвенно), метода getMatrix() и getVertices(). На выходе ожидается двумерный массив типа int, представляющий матрицу смежности построенного графа (при тестировании метода getMatrix()) и строка с именами вершин (при тестировании метода getVertices()). Размер матрицы смежности должен соответствовать количеству вершин в графе (|M| = V*V), значения элементов на главной диагонали должно равняться нулю, в матрице не должно быть элементов со значениями меньше минуса единицы.

Создается граф, строки передаются на вход конструктору графа. Затем вызываются методы getVertices() или getMatrix().

Что подается на вход	Тестируемый метод	Что ожидается на выходе
Пустая строка	public String getVertices()	null
Некорректная строка	public String getVertices()	Исключение
Непустая строка,	public String getVertices()	Строка, состоящая из имен

соответствующая всем требованиям (v_1 v_2 $w_{v1, v2}$ \n)		вершин графа (без пробелов). Имена не повторяются.

Что подается на вход	Тестируемый метод	Что ожидается на выходе
Пустая строка	public int[][] getMatrix()	null
Некорректная строка	public int[][] getMatrix()	Исключение
Непустая строка, соответствующая всем требованиям (v_1 v_2 $w_{v1, v2}$ \n)	public int[][] getMatrix()	Двумерный массив типа int, отвечающий всем требованиям

Так же тестируется метод getName() класса Vertex. На вход конструктору объекта подается имя вершины типа char, на выходе метода ожидается это же имя.

Что подается на вход	Тестируемый метод	Что ожидается на выходе
Символ	public char getName()	Этот же символ

3. Создание матрицы смежности по введенным данным

Тестируются методы класса Connection и метод getMatrix() класса Graph на соответствие матрицы смежности с входной строкой. Для тестирования методов класса Connection создается объект этого же класса с параметрами конструктора равными весу ребра из одной вершины в другую и кратчайшим путем между ними типа String (изначально кратчайший путь из вершины *a* и в вершину *b* равен *ab*). Тестируются методы getWeight(), getPathLength(), getPath() и changePath(int pathLength, String path). На вход методу changePath() подается новый кратчайший путь и новая длина. Этот метод будет тестироваться совместно с остальными методами (сначала вызывается метод changePath, затем по очереди снова проводятся все тесты предыдущих методов).

Что подается на вход	Тестируемый метод	Что ожидается на выходе
Вес ребра, путь	public int getWeight()	Вес ребра
Вес ребра, путь	public int getPathLength()	Длина пути
Вес ребра, путь	public String getPath()	Путь

После вызова метода changePath(int pathLength, String path) (на входе указаны аргументы этого метода)

Что подается на вход	Тестируемый метод	Что ожидается на выходе
Новая длина пути и новый путь	public int getWeight()	Вес ребра (не должен измениться)
Новая длина пути и новый путь	public int getPathLength()	Новая длина пути
Новая длина пути и новый путь	public String getPath()	Новый путь

Так же тестируется метод getMatrix() класса Graph на соответствие входной строке. Значения всех M[i][j] должно равняться весу ребра между вершинами i и j. Значения M[i][j] должно равняться минус единице при условии, что ребра межу вершинами i и j нет. В матрице должны быть указаны все ребра из входной строки и не должно быть создано лишних (не тестируется правильность структуры, ожидается, что она уже ранее протестирована).

Что подается на вход	Тестируемый метод	Что ожидается на выходе
Строка из одного ребра	public int[][] getMatrix()	Матрица со значениями минус один во всех элементах кроме одного
Новая длина пути и новый путьpublic int[][] getMatrix()	public int[][] getMatrix()	Матрица, отвечающая условиям
Строка из (numVertex)*(numVertex-1) ребер	public int[][] getMatrix()	Матрица с положительными значениями всех элементов кроме элементов на главной диагонали

4. Работа алгоритма

Тестируется метод FloydWarshall() класса Graph. Входными данными является поле класса Connection[][] matrix. Алгоритм работает напрямую с этой матрицей, изменяя значения пути в ее элементах (в объектах класса Connection)

Что подается на вход	Тестируемый метод	Что ожидается на выходе
Матрица с единственным положительным элементом (в графе только одно ребро)	public void FloydWarshall()	Пути элементов в матрице не должны измениться
Матрица со всеми положительными элементами (кроме элементов на главной диагонали)	public void FloydWarshall()	Длина пути всех элементов матрицы (кроме элементов на главной диагонали) должна остаться положительной