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

ОТЧЁТ

по учебной практике

Тема: Визуализация работы алгоритм Краскала

Студент гр. 9383 Хотяков Е.П.

Студентка гр. 9383 Чебесова И.Д.

Студентка гр. 9383 _____ Лапина А.А.

Преподаватель _____ Фиалковский М.С.

Санкт-Петербург
2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Хотяков Е. П. группы 9383

Студентка Чебесова И.Д. группы 9383

Студент Лапина А.А. группы 9383

Тема практики: Визуализация работы алгоритм Краскала

Задание на практику:

Командная итеративная разработка визуализации алгоритма на Java с графическим интерфейсом.

Алгоритм: Краскала.

Сроки прохождения практики: 01.07.2021 – 11.07.2021

Дата сдачи отчета: 06.07.2021

Дата защиты отчета: 06.07.2021

Студент гр. 9383 Хотяков Е.П.

Студентка гр. 9383 Чебесова И.Д.

Студентка гр. 9383 _____ Лапина А.А.

Преподаватель _____ Фиалковский М.С.

АННОТАЦИЯ

Основной целью работы является получение навыков программирования на языке Java и освоение парадигмы объектно-ориентированного программирования. Цель достигается командной работой путём разработки программы с графическим интерфейсом, использующей указанные в задании алгоритмы. В этой работе необходимо реализовать алгоритм, находящий минимальное остовное дерево. Пользователь задает количество ребер и вершин в графе, а затем и сами ребра с весом. Для проверки корректности программы используются JUnit тесты.

SUMMARY

The main goal of the work is to acquire programming skills in the Java language and master the paradigm of object-oriented programming. The goal is achieved by teamwork by developing a program with a graphical interface using the algorithms specified in the task. In this work, it is necessary to implement an algorithm that finds the minimum spanning tree. The user sets the number of edges and vertices in the graph, and then the edges themselves with weight. JUnit tests are used to check the correctness of the program.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача практики состоит в разработке приложения, позволяющего находить минимальное остовное дерево. Программа визуализирует пошаговое нахождение остовного дерева на графе. Для его поиска используется алгоритм Краскала.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

Спецификация проекта «Визуализация работы алгоритм Краскала»

1.1. Описание задачи

Пользователь вводит два числа - количество вершин V и количество ребер N. Затем он вводит сами ребра в следующем формате: start finish weight — то есть стартовая вершина, конечная и вес ребра. (На данный момент в качестве вершин могут использоваться только числа, но возможно, что по итогу в качестве вершин будет любая строка). После ввода данных сортируем ребра и вполняем поиск минимального остовного дерева с помощью алгоритма Краскала, который визуализируется с помощью библиотеки swing.

1.2. Интерфейс

Интерфейс программы состоит из:

• Окно для ввода данных пользователем с кнопками начала запуска алгоритма («Start»), повторного ввода данных («Restart») и меню («Options»).

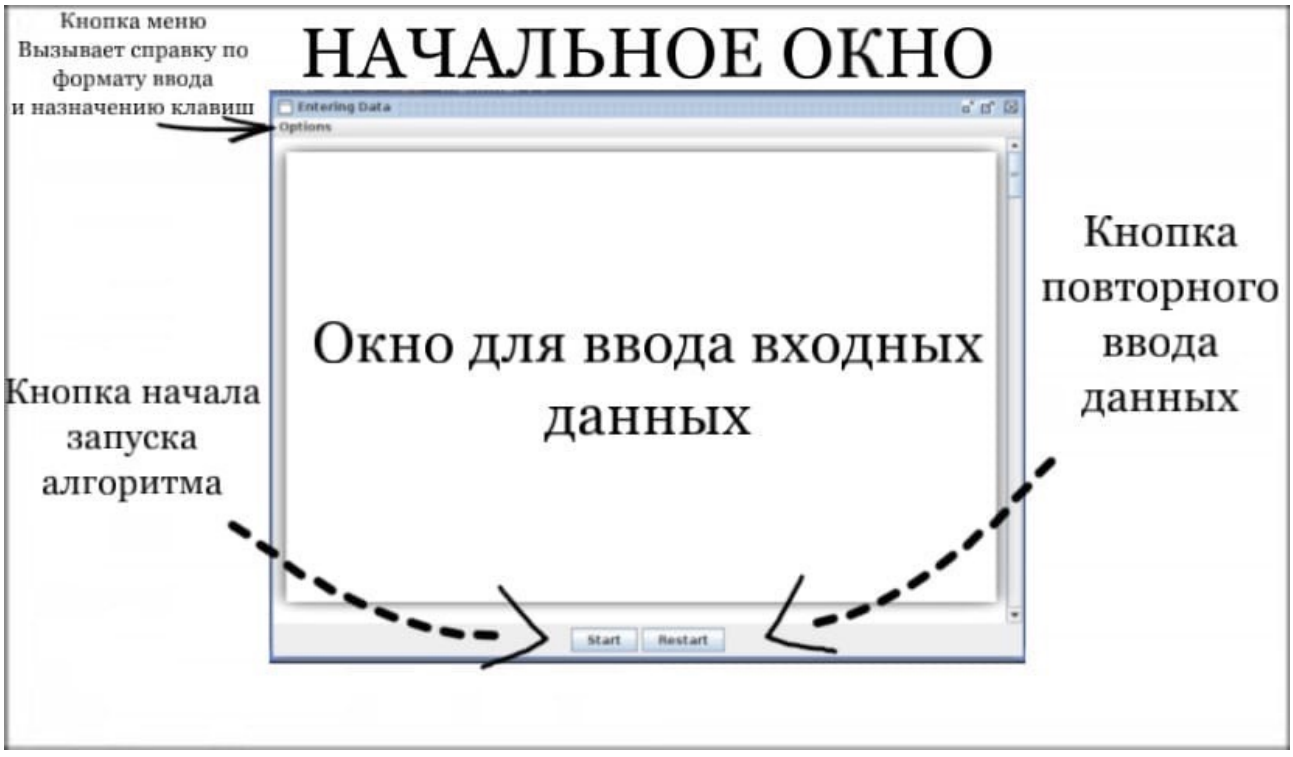

Рисунок 1 — Начальное окно

• Кнопка «Start» отвечает за начало работы алгоритма, закрытие начального окна и открытие нового. После ее нажатия данные также будут проанализированы, и, в случае возникновения ошибки, будет появляться окно с соответствующим сообщением.

• Кнопка «Restart» служит для сброса входных данных и выводит сообщение пользователю с просьбой ввести новые данные.

• Кнопка импорт из файла.

• Кнопка «стрелка в кружочке» возвращает алгоритм в начало, печатая соответствующее сообщение в текстовое поле окна (Этот текст нужен для демонстрации кнопок и потом будет удален, вместо чего будет меняться гарф).

• Кнопка «шаг назад» делает шаг назад алгоритма и выводит сообщение.

• Кнопка «шаг вперед» выводит сообщение и делает шаг вперед у алгоритма.

• Кнопка «> >» сразу прогоняет алгоритм и выводит итоговый результат.

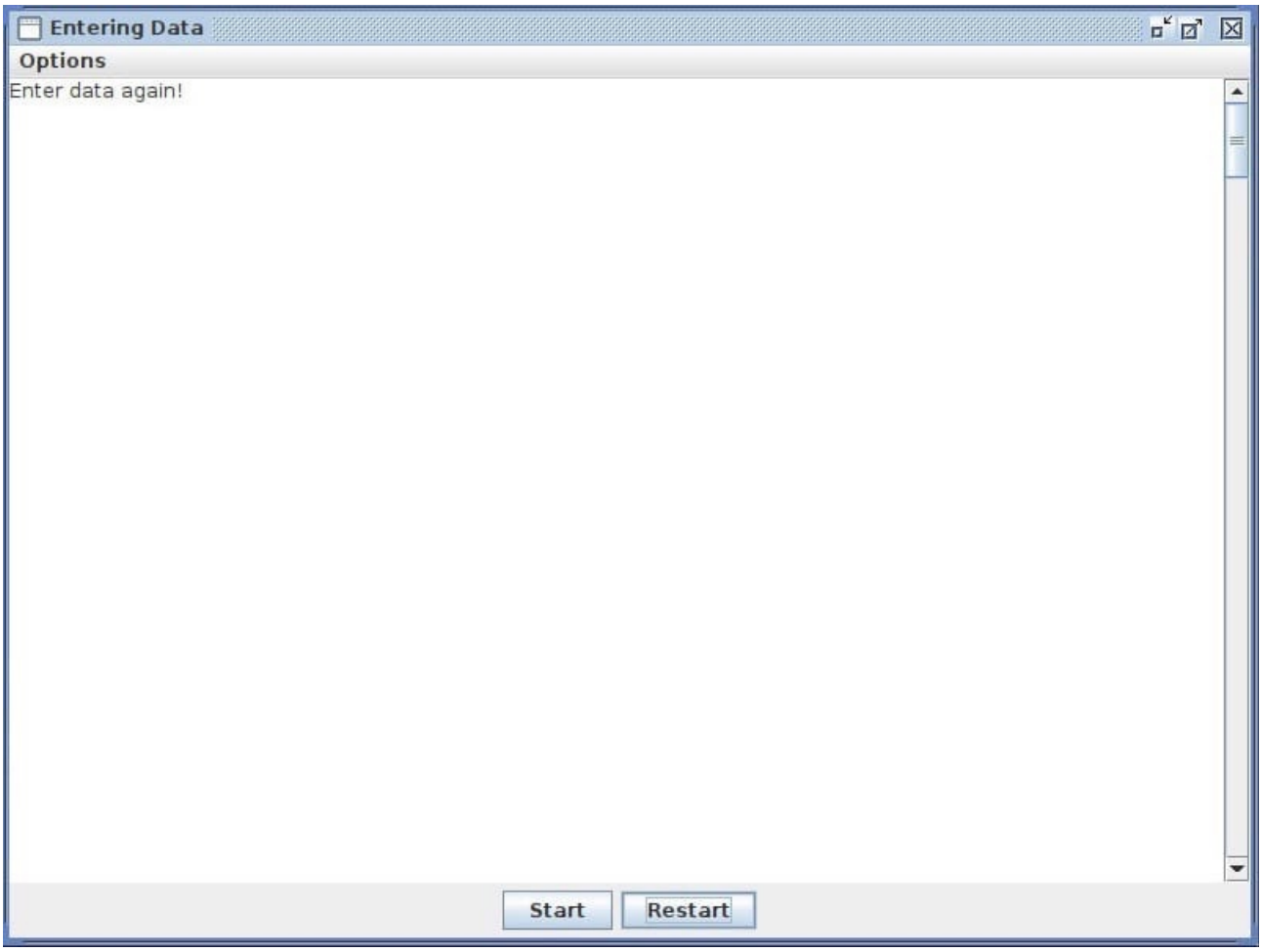

Рисунок 2 — Демонстрация работы программы после нажатия кнопки «Restart»

• Меню с пунктом «Help!», после вызова которого появляется новое окно с информацией по формату ввода и назначении кнопок.

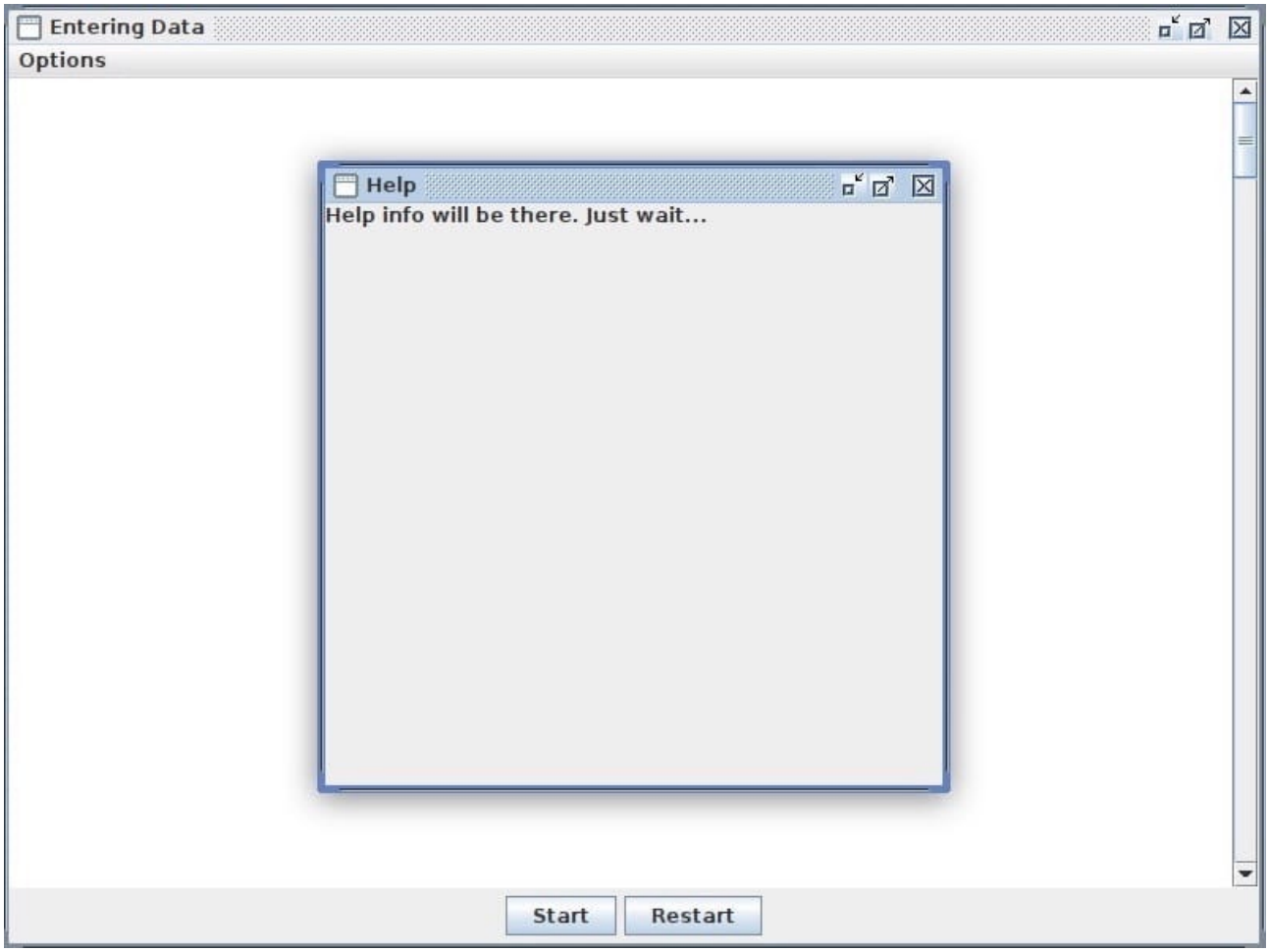

Рисунок 3 — Окно «Help!» с информацией о вводе данных

• Основное окно работы алгоритма. Оно содержит четыре кнопки отвечающие за возврат к началу работы алгоритма, шаг назад, шаг вперёд, и переход к результату работы. На экране будет начерчен граф и массив рёбер. На каждом шаге будут изменятся цвета рёбер для наглядной демонстрации рассматриваемого шага.

Рисунок 4 — Основное окно работы алгоритма

• Окно в процессе работы приложения. В нём также есть возможность вызвать окно Help! с информацией о происходящих на экране событиях

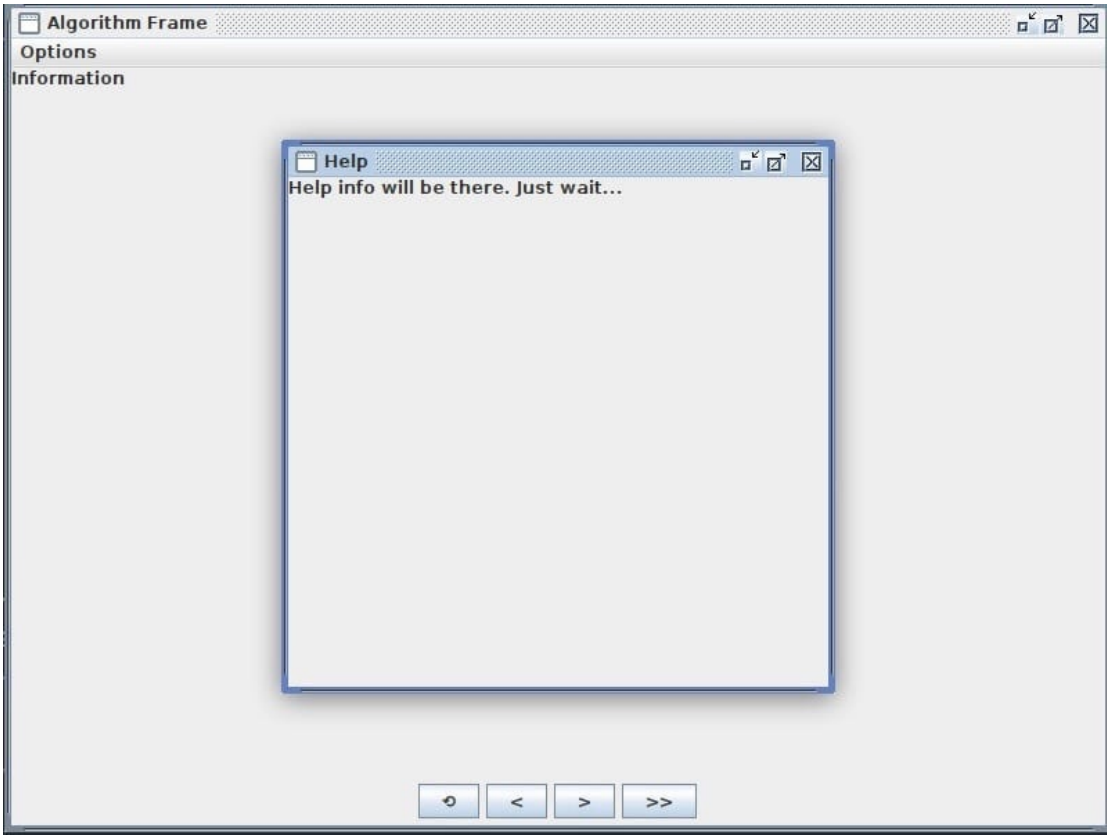

Рисунок 5 — Демонстрация окна во время выполнения алгоритма

1.3. Входные данные

Первая строка входных данных должна содержать два числа N и M, разделенные пробелом — количество вершин и количество ребер графа. Далее следуют M строк, каждая из которых содержит по три целых числа, разделенные пробелами. Первые два из них разные, в пределах от 0 до N - 1 каждое, и обозначают концы соответствующего ребра, третье — в пределах от 1 до 1000000000 и обозначает длину этого ребра.

Пример:
3 3
0 1 1
0 2 2
1 2 3

1.4. Выходные данные

Выводится граф и массив рёбер. На каждом шаге будут изменятся цвета рёбер для наглядной демонстрации рассматриваемого шага. В итоге будет найдено минимальное остовное дерево и выписаны: вес МОД и ребра, входящие в МОД.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. К 4 июля начать работу над пользовательским интерфейсом,

обсудить применения алгоритмов для решения поставленной

задачи.

2. К 5 июля подготовить прототип пользовательского интерфейса и

спецификацию проекта.

3. К 6 июля реализовать основные классы и методы для реализации алгоритма.

4. К 6 июля подготовить вывод окон в приложении с помощью swing.

5. К 7 июля реализовать интерфейс: вывод графа, добавление дуг, графическая демонстрация работы алгоритма.

6. К 8 июля организовать пробное соединение интерфейса и логики.

7. К 9 июля закончить работу по логике интерфейсу программы.

8. К 9 июля протетировать программу, написать Unit-тесты.

8. К 11 июля завершить разработку проекта.

2.2. Распределение ролей в бригаде

• Чебесова Ирина – фронтенд;

• Лапина Анастасия – тестирование, документация;

• Хотяков Евгений– алгоритмист, лидер.

3. ФУНКЦИИ И СТРУКТУРЫ ПРОГРАММЫ

3.1. *class* EnterData*:*

*Х*ранит в себе список всех введенных ребер, а также количество этих ребер и количество вершин в графе.

В конструкторе класса происходит считывание данных.

Методы:

• метод для сортировки списка ребер. Ребра сортируются по весу в порядке возрастания.

3.2. *class* Kruscal*:*

Реализовывает алгоритм. Он хранит в себе:

Поля:

• tree_id - массив, хранящий для каждой вершины номер множества, к которому вершина принадлежит. (Изначально все вершины принадлежат разным множествам);

• result - массив, в который добавляются ребра, входящие в МОД;

• data - экземпляр класса EnterData, хранит в себе введенные пользователем данные;

• cost - суммарный вес МОД;

• currentStep - индекс, указывающий, на каком шаге алгоритма мы находимся;

• saveStack - стэк, который используется для сохранения данных при возврате на шаг назад;

• circleFlag - массив, который для каждого этапа хранит информацию о том, был ли найден цикл на данном шаге (true) или нет (false);

Методы:

В конструкторе инициализируются начальные данные и сортируется массив с ребрами по возрастанию по весу этих ребер.

• union - метода для объединения двух множеств вместе.

- checkConnected - используется после прохождения по всем ребрам графа. Возвращает true, если не удалось получить МОД (граф оказался несвязным) и false, если МОД получен.

- isEnd - проверяет, находится ли алгоритм на конечном этапе.

- nextStep - метод реализует единичную итерацию алгоритма Краскала.

- toFinal - метод прогоняет алгоритм от текущей итерации до конца алгоритма.

- prevStep - метод реализует единичный шаг назад по алгоритму.

- toStart - метод возвращает алгоритм на начальный этап.

- printResult - возвращает строку, в которой записаны все ребра итогового МОД.

- getCost() - возвращает значение cost — суммарный вес МОД.

4. ТЕСТИРОВАНИЕ

4.1. Ручное тестирование

Пример 1

Входные данные:

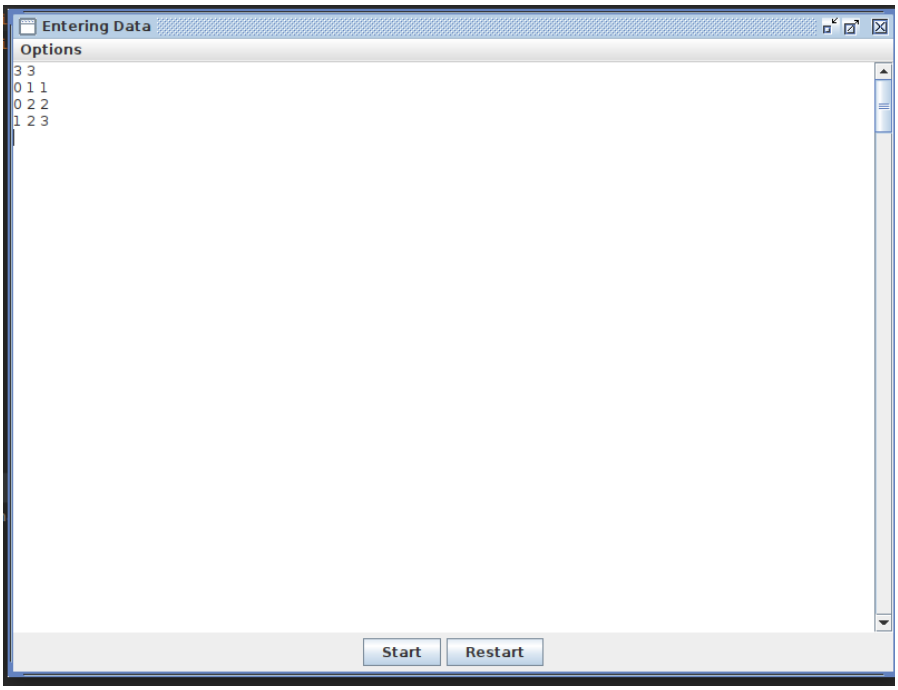

Рисунок 6 – Входные данные для примера 1

Выходные данные:

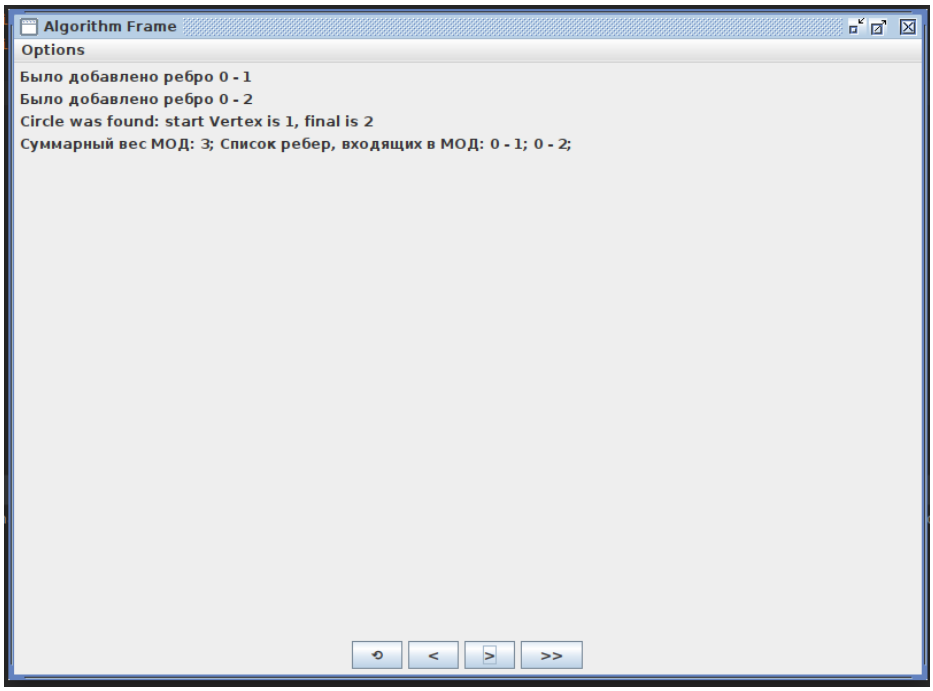

Рисунок 7 – Выходные данные для примера 1

Пример 2

Входные данные:

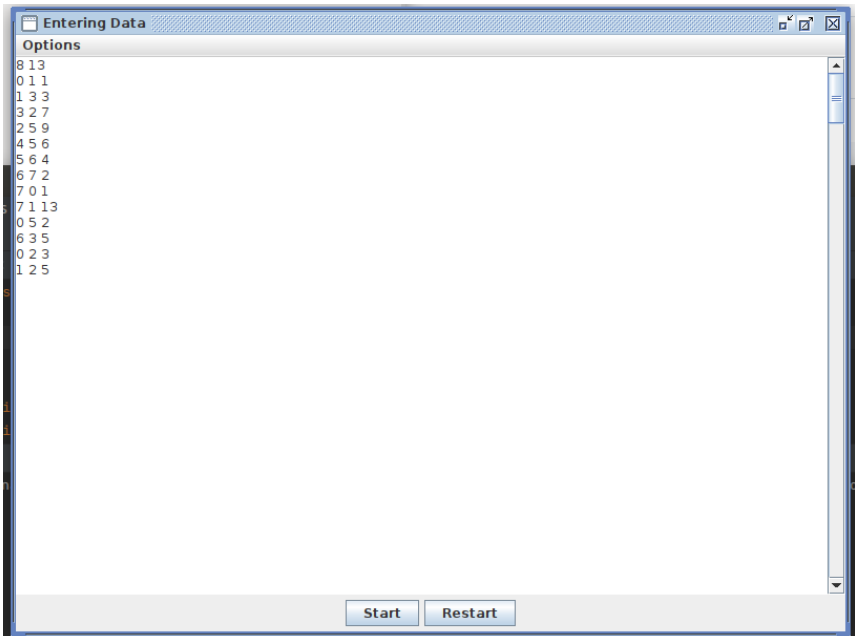

Рисунок 8 – Входные данные для примера 1

Выходные данные:

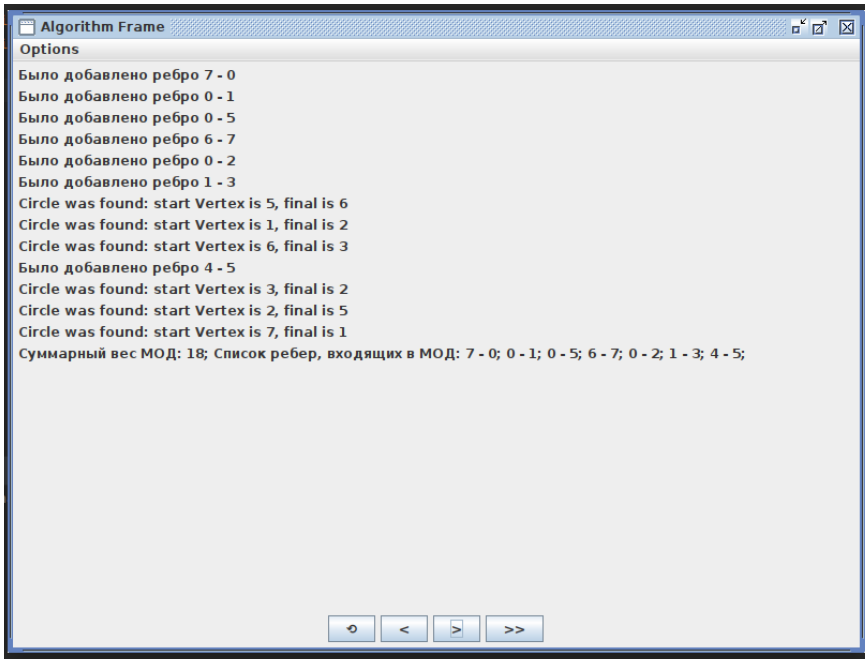

Рисунок 9 – Выходные данные для примера 1

4.2. Unit-тестирование

Для класса **EnterData** применено Unit-тестирование с использованием библиотеки Junit 4. Создан экземпляр класса, проверенна корректность ввода данных. Реализовано 2 теста – входные данные считываются корректно.

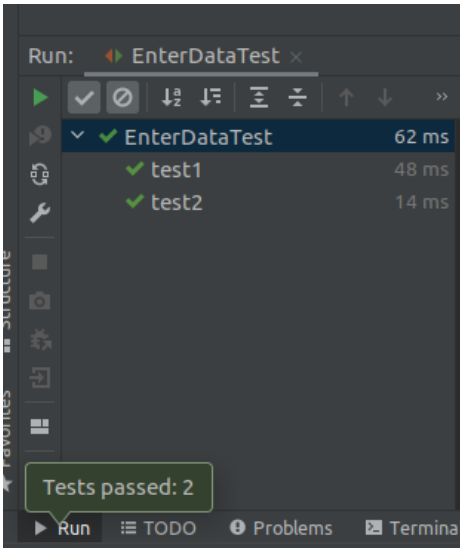

Рисунок 10 – Демострация успешно пройденных тестов для класса EnterData

Для класса **Kruscal** применено Unit-тестирование с использованием библиотеки Junit 4. Создан экземпляр класса, проверенна корректность работы метода toFinal(). Изначально было проверено значение поля cost – суммарный вес МОД (должен равняться 0), затем применен метод toFinal() и проверен суммарный вес МОД – новое значение поля cost. Удачно пройдено 2 теста из 2.

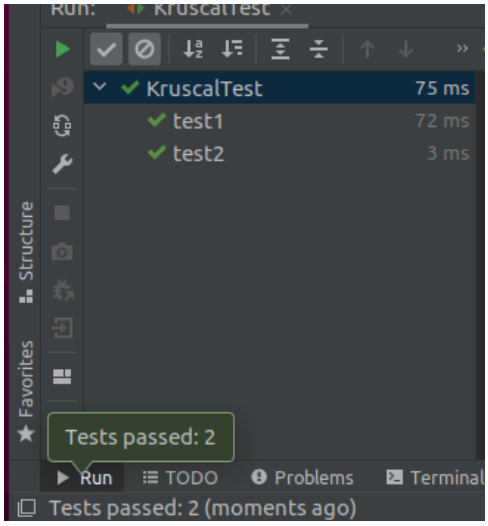

Рисунок 11 – Демострация успешно пройденных тестов для класса Kruscal

ЗАКЛЮЧЕНИЕ

В соответствии с требованиями технического задания была реализована программа для нахождения минимального остовного дерева графа, задаваемого пользователем. Разработан пользовательский интерфейс. Он позволяет считывать введенные данные, анализировать и демонстрировать работу алгоритма Краскала.

В результаты работы были освоены методы работы в команде, распределены обязанности. Также команда была ознакомлена языком программирования Java, в частности с фреймворком Swing для создания графического интефейса, библиотекой JUnit 4 для модульного тестирования.

ПРИЛОЖЕНИЕ А
РАЗРАБОТАННЫЙ КОД