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

ОТЧЕТ

по учебной практике

Тема: Разработка визуализатора алгоритма на Java. Минимальное остовное дерево: алгоритм Прима

Студентка гр. 0383 _____ Александрович В. П.

Студент гр. 0383 _____ Парфенов В. М.

Студентка гр. 0383 _____ Куртова К. А.

Руководитель _____ Ефремов М. А.

Санкт-Петербург
2022

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студентка Александрович В. П. группы 0383

Студент Парфенов В. М. группы 0383

Студентка Куртова К. А. группы 0383

Тема практики: Разработка визуализатора алгоритма на Java. Минимальное остовное дерево: алгоритм Прима.

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: алгоритм Прима для нахождения минимального остовного дерева.

Сроки прохождения практики: 29.06.2022 – 12.07.2022

Дата сдачи отчета: 12.07.2020

Дата защиты отчета: 12.07.2020

Студентка гр. 0383	_____	Александрович В. П.
Студент гр. 0383	_____	Парфенов В. М.
Студентка гр. 0383	_____	Куртова К. А.
Руководитель	_____	Ефремов М. А.

АННОТАЦИЯ

Целью проекта является разработка итеративного визуализатора алгоритма Прима с графическим интерфейсом на языке Java. Главными задачами проекта являются создание плана работы и разработка приложения согласно этому плану. Разработка осуществляется в команде, каждый участник которой выполняет свою роль. Конечным результатом является приложение с графическим интерфейсом, пошагово визуализирующее выполнение алгоритма Прима для поиска минимального остовного дерева на графе, заданном одним из нескольких способов: граф либо генерируется самой программой, либо считывается из текстового файла, либо вводится самим пользователем с использованием инструментов, предоставленных программой. Проект предполагает разработку программы на Java с JDK версии 17 с GUI, реализованным с помощью Swing. Для сборки проекта используется Maven, программа покрывается тестами Junit 5.

SUMMARY

The objective of the project is to develop an iterative visualizer of the Prim's algorithm with a graphical interface in Java language. The main tasks of the project are to create a work plan and develop applications according to this plan. Development is carried out in a team, each participant performing their role. The end result is a GUI application that visualizes the execution of the Prim's algorithm step by step. The graph is defined in one of several ways: it is either generated by the program itself, or read from a text file, or entered by the user themselves with the use of tools provided by the program. Is it presumed to build a Java program with JDK version 17 with a GUI implemented with Swing. Maven is used to build the project. The program is covered by Junit 5 tests.

СОДЕРЖАНИЕ

Оглавление

ВВЕДЕНИЕ

Целью проекта является итеративная разработка приложения с графическим интерфейсом на Java, предоставляющего пользователю инструменты для взаимодействия с ним. Приложение должно визуализировать выполнение алгоритма Прима. Алгоритм Прима строит минимальное остовное дерево взвешенного связного неориентированного графа.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

1.1.1. Требования к вводу исходных данных

Входные данные могут быть заданы одним из нескольких способов:

1) Загружены из файла формата json.

2) Введены пользователем вручную с помощью инструментов, предоставленных программой. Пользователю предоставляется несколько режимов: режим ввода вершин, режим ввода ребер. Также пользователь может отчистить весь холст. Вершины добавляются пользователем щелчком мыши на свободную часть холста. Ребра добавляются выбором двух вершин в режиме добавления ребер, пользователя просят ввести вес создаваемого ребра.

3) Сгенерированы самой программой. Данные о графе генерируются и визуализируются программой. Полученный граф должен удовлетворять условиям, требующимися для выполнения алгоритма (т. е. быть связным, неориентированным и взвешенным).

В первых двух случаях обязательна проверка на связность графа.

1.1.2 Требования к визуализации

Графический интерфейс приложения должен содержать панель инструментов и холст, на котором будет представлен граф.

На панели инструментов содержится несколько кнопок, отвечающих за следующие действия:

1) Кнопка “Очистить” удаляет все элементы с холста.

2) Кнопка “Загрузить” позволяет загрузить файл формата .json с информацией о графе.

3) Кнопка “Сгенерировать” добавляет на холст граф, сгенерированный самой программой.

4) Кнопка “Запуск алгоритма” запускает нахождение минимального остовного дерева, выделяя соответствующие ребра другим цветом.

Диаграмма сценариев представлена на рис. 1, макет графического интерфейса представлен на рис. 2.

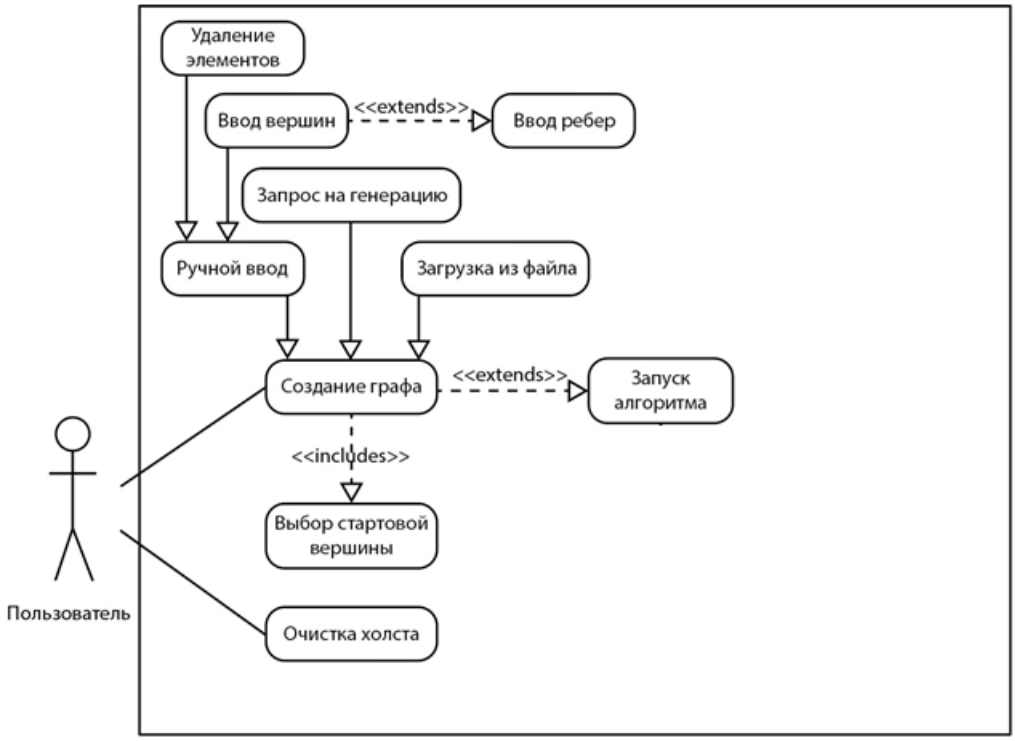

Рисунок 1 — Диаграмма сценариев использования

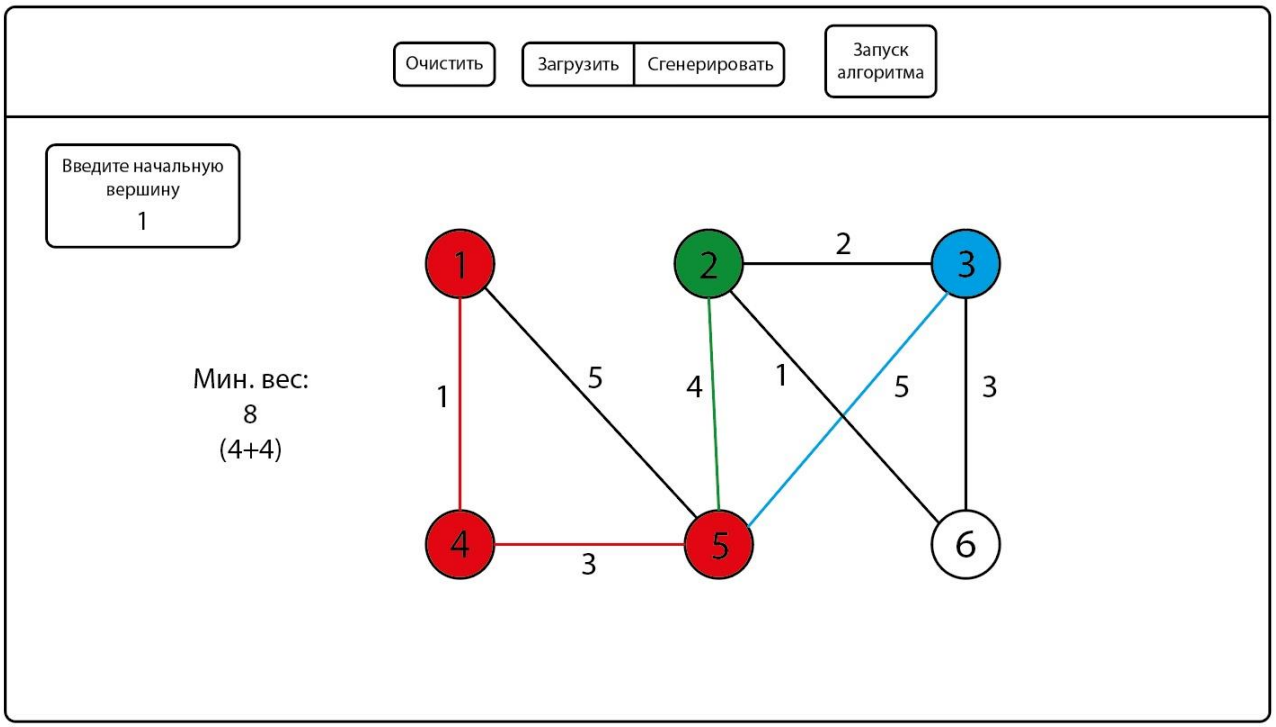

Рисунок 2 — Макет графического интерфейса приложения

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

02.07 — создание файла отчёта, заполнение титульного листа с постановкой задачи и сроками, написание раздела спецификаций, описание ролей участников команды; согласование спецификаций; создание Maven-проекта.

04.07 — Создание прототипа интерфейса: интерфейс программы на заглушках, реализация структур данных и алгоритма, тесты для структур данных и алгоритма; создание диаграмм классов и описание сущностей.

06.07 — Рабочий прототип: реализация всех типов генерации данных, выполнение и отображение работы алгоритма; представление диаграмм последовательности, описание тестовых случаев.

08.07 – Корректная работа кнопок, отвечающих за пошаговое исполнение алгоритма, реализация структур данных, отвечающих за пошаговое выполнение алгоритма, реализация тестов для этих структур; создание диаграммы состояний для описания процесса пошагового исполнения алгоритма, пояснения к диаграмме, описание тестовых случаев, описание интерфейса взаимодействия с пошаговым выполнением алгоритма.

10.07 — Сборка проекта в jar-архив, представление итогового отчёта.

2.2. Распределение ролей в бригаде

Александрович Валерия — покрытие программы тестами, связь интерфейса и внутренних структур данных

Парфенов Владислав — визуализация, создание интерфейса программы

Куртова Карина — реализация алгоритма и структур данных

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Структуры данных, использующиеся для представления графа.

1) interface Graph

Определяет поведение для абстрактного графа (добавление вершины и ребра, очищение списка вершин и рёбер). Имплементируется классом WeightedConnectedGraph.

2) interface Weighted

Описывает поведение взвешенного графа (добавить вес вершине, запросить вес вершины). Имплементируется классом WeightedConnectedGraph.

3) interface Connected

Описывает поведение связного графа (метод проверки графа на связность). Имплементируется классом WeightedConnectedGraph.

4) class WeightedConnectedGraph

Имплементирует описанные выше интерфейсы. Содержит список вершин AlgoNode и методы для работы с ним.

5) abstract class Node

Абстрактный класс, представляющий вершину графа. Он Него наследуются

6) abstract class Edge

Класс, представляющий абстрактное ребро графа.

7) class AlgoNode

Наследуется от класса Node. Этот класс используется для представления графа, на котором работает алгоритм Прима. Содержит структуру, хранящую вершины, инцидентные данной и соответствующие вершины между ними.

8) class AlgoEdge

Наследуется от класса Edge. Этот класс используется для представления графа, но котором работает алгоритм Прима.

Структура данных, отвечающая за работу алгоритма.

1) class PrimAlgo

Содержит метод, реализующий алгоритм Прима.

UML-диаграмма данных классов представлена на рис. 3.

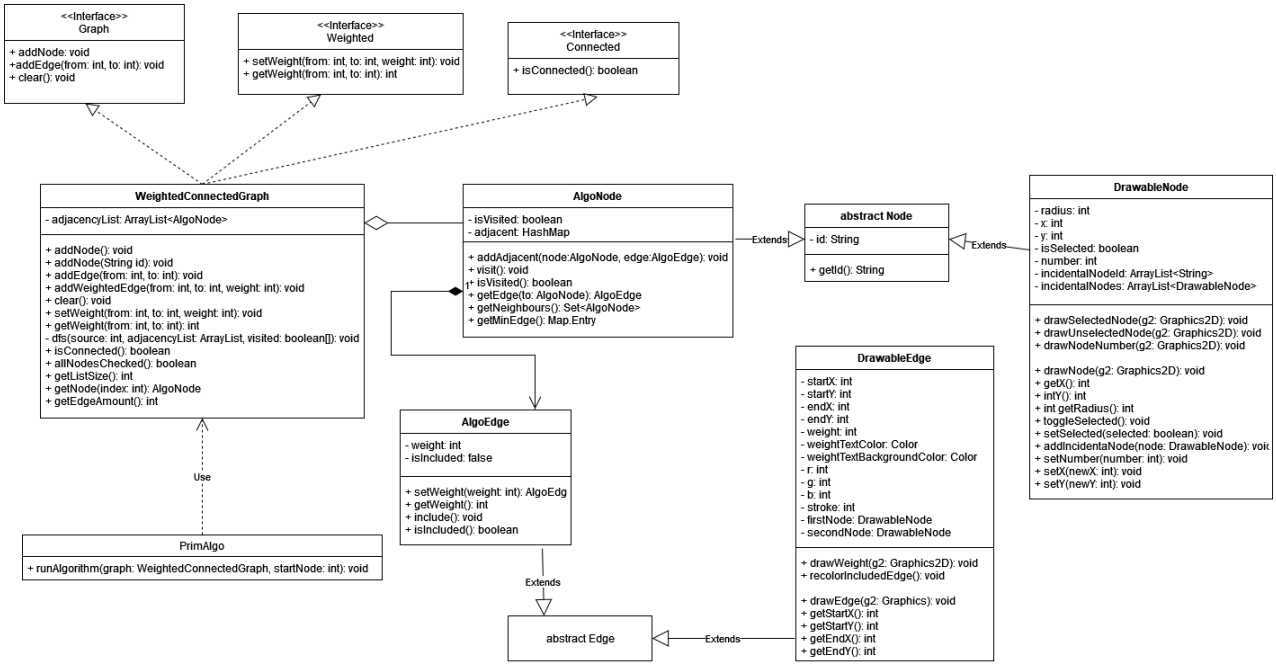

Рисунок 3 — UML-диаграмма классов, отвечающих за построение графа и реализацию алгоритма

Структуры данных, использующаяся в реализации интерфейса приложения.

1) Class AppWindow

Класс, отвечающий за открытие главного экрана приложения и взаимодействие пользователя с программными компонентами.

2) Class Canvas

Класс, отвечающий за рабочую область приложения (там, где располагается граф).

3) Class Menu

Класс, отвечающий за отображение меню и кнопок на нем в верхней части приложения.

4) Class Button

Абстрактный класс, который является обобщением всех кнопок в меню.

5) Enum Buttons содержит все названия кнопок, которые есть в приложении.

Структура данных, использующаяся для генерации входных данных (графа).

1) RandomGraphGenerator – генератор случайного связного двунаправленного взвешенного графа. Класс имеет один статический метод – public static WeightedConnectedGraph createRandomGraph(int nodeAmount, int minEdgeAmount, int maxEdgeAmount, int minWeight, int maxWeight), который получает на вход количество вершин, границы количества ребер и их весов. Метод возвращает связный граф с заданными параметрами.

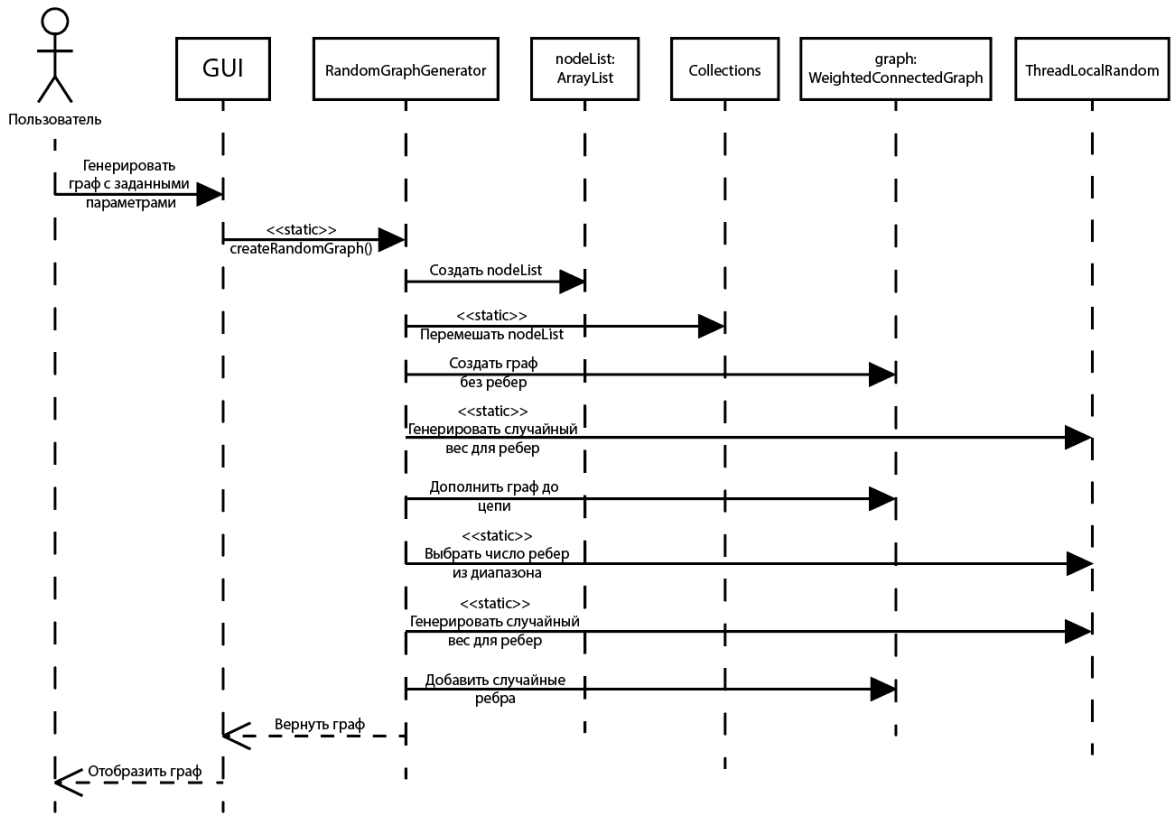

Рисунок 4 — Диаграмма последовательности для случайной генерации графа.

Структура данных, отвечающая за компонент Canvas:

1) Canvas – класс, отвечающий за ввод графа пользователем вручную. Он наследуется от JPanel. Также в этом классе добавляются mouse и action listeners, которые нужны для обработки щелчков мыши и реагирования на нажатие кнопок. Здесь имеется несколько полей. DrawableGraph graph – объект графа, который необходимо отрисовывать и изменять, selectedNodes – массив,

содержащий индексы вершин, которые были выделены, draggableNode – вершина, которая в данный момент перетаскивается и form, отвечающий за форму, куда вводятся данные о весе ребра.

a) Метод paint использует Graphics для отображения вершин и ребер, нарисованных пользователем.
b) Метод addNewNode добавляет новую вершину.
c) Метод createWeighedEdgeBetweenTwoNodes добавляет взвешенное ребро между двумя вершинами.
d) Метод isAbleToPutNode проверяет, достаточно ли далеко вершина находится от соседних вершин.
e) unselectAllNodes – убрать выделение со всех вершин.
f) submitEdgeWeight – метод, который называется при нажатии на кнопку в компоненте form.
g) Метод getGraph возвращает граф, который необходимо отрисовать на холсте.
h) Создано два подкласса ClickListener и DragListener, которые расширяют MouseAdapter и MouseMotionAdapter. В первом классе переопределен метод MouseClicked - при нажатии левой кнопки мыши создается вершина, правой кнопкой мыши выделяются вершины и если выделено две вершины, между ними создается ребро, а также mousePressed для работы с drag and drop. В классе DragListener переопределен метод mouseDragged, который просто меняется координаты вершины, которая в данный момент перетаскивается.

2) Class CanvasForm отвечает за компонент ввода веса ребра. Наследуется от JPanel. В нем имеется два поля submitButton и textField. С их помощью происходит ввод и считывание данных.

a) Метод disableForm очищает textField и блокирует доступ к кнопке и полю ввода для пользователя.
b) Метод enableForm позволяет пользователю вводить данные и отправить их.

c) Метод getData возвращает данные, полученные из textField.

3.2. Основные методы

Методы, использующиеся для работы алгоритма.

1) class AlgoNode

Метод *Map.Entry<Node, Edge> getMinEdge()*. Выбирает из списка вершин, инцидентных данной, вершину, к которой ведёт ребро с наименьшим весом. Возвращает пару вершина — минимальное ребро.

2) class PrimAlgo

1) Методы работы с графом *void addNode(), void addEdge(int firstNode, int secondNode, Integer weight).* Первый метод добавляет новую вершину в список графа, вторая добавляет новое двунаправленное ребро с заданным весом между вершинами, представленными номерами их индексов.
2) Метод *boolean isDisconnected()* возвращает true, если были пройдены ещё не все вершины графа, false – иначе.
3) Метод *void runAlgorithm()* запускает работу алгоритма Прима. В результате получаем структуру графа с различными состояниями вершин и ребер.
4) Метод *String minimumSpanningTree()* возвращает строку, содержащую информацию о минимальном остовном дереве, в формате *[вершина1] [вершина2] [вес ребра между вершинами]* на каждой строке.

4. ИНТЕРФЕЙС ПРИЛОЖЕНИЯ

4.1. Создание главного экрана приложения.

Для создания интерфейса будет использоваться Swing. Для начала необходимо создать главное окно, которое будет называться appWindow. Класс main будет создавать экземпляр класса appWindow, который будет открывать приложение. Класс appWindow является наследником класса JFrame. В конструкторе были описаны некоторые базовые опции, такие как размер экрана, закрытие окна по нажатии на крестик, layout, а также название приложения. Дальше было создано два компонента Menu и Canvas, первый из которых отвечает за вывод меню, а второй за рабочую область.

4.2. Создание интерфейса меню.

Класс Menu является наследником JPanel. В его конструкторе происходит создание экземпляров различных кнопок. Затем, в том же месте, кнопки добавляются на панель Menu и затем класс меню выводится в классе appWindow.

4.3. Создание классов кнопок.

Был создан класс Button, который является наследником класса JButton. В его конструкторе задаются параметры кнопки, а именно текст кнопки и ActionListener, который необходимо выполнить по нажатии на конкретно взятую кнопку. В enum Buttons содержаться названия всех кнопок, которые есть в приложении.

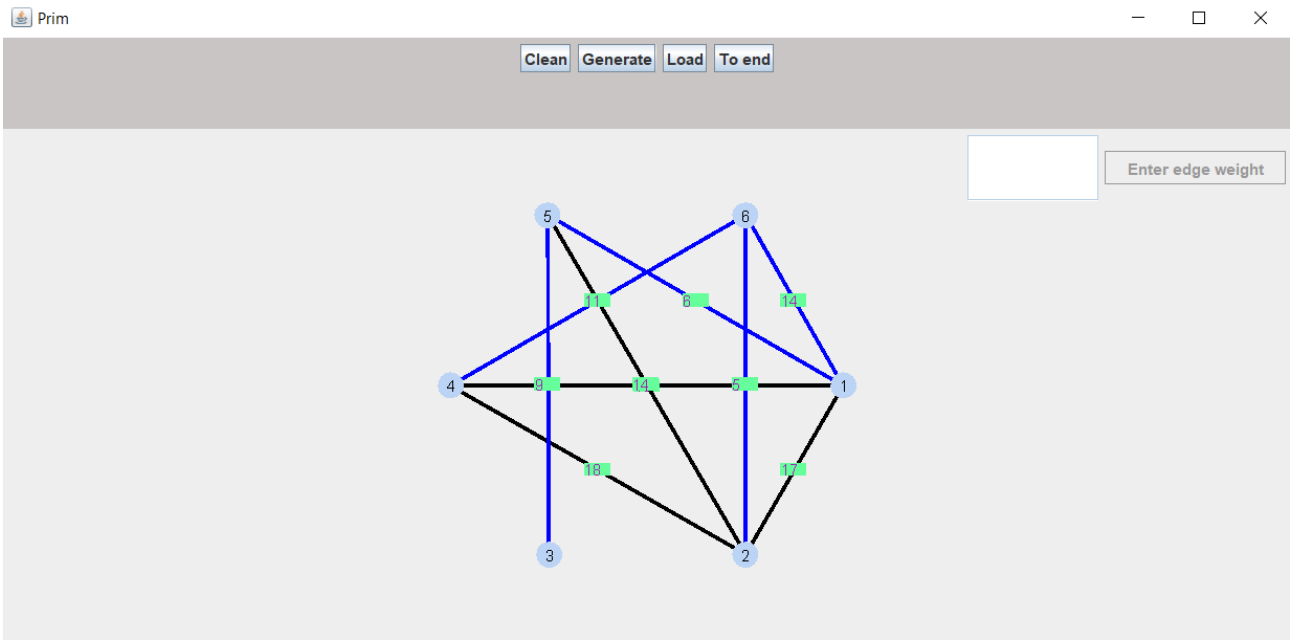

Рисунок 4 — Скриншот интерфейса

5. ТЕСТИРОВАНИЕ

5.1. Тестирование алгоритма Прима для связного графа

Для тестирования реализации алгоритма Прима был использован фреймворк JUnit. Тесты были разделены на следующие группы:

1) Граничные случаи.

   А) Граф, состоящий из единственной вершины.

   Б) Связный граф, состоящий из вершин, соединенных ребрами с нулевыми весами.

2) Позитивные тесты.

   А) Связный граф, состоящий из двух вершин. Алгоритм запускался из обеих вершин поочередно.

   Б) Связный граф, состоящий из десяти вершин, соединенных ребрами с различными весами.

Все тесты показали правильность исполнения алгоритма.

3) Негативные тесты.

   А) Граф с нулевым количеством вершин.

   Б) Несвязный граф.

   В) Граф с отрицательным весом ребер.

Результаты тестирования алгоритма Прима представлены в таблице 1.

Таблица 1:

Название	Входной граф (1я верш. – вес – 2я верш.)	Проверки	Комментарии
Graph with one node	Одна вершина Запуск из вершины 1	Node[0] is visited	ВЕРНО
Graph with two nodes (start in first node)	1 – 2 – 2 Запуск из вершины 1	Node[0] is visited Node[1] is visited Edge[0][1] is included	ВЕРНО

Graph with two nodes (start in second node)	1 – 2 – 2 Запуск из вершины 2	Node[0] is visited Node[1] is visited Edge[1][0] is included	ВЕРНО
Graph with edges of zero weight	1 – 0 – 2 2 – 0 – 3 1 – 0 – 3 Запуск из вершины 1	Node[0] is visited Node[1] is visited Node[2] is visited Edge[0][1] is included Edge[0][2] is included Edge[1][2] is not included Edge[2][1] is not included	ВЕРНО
Graph with ten nodes	1 – 4 – 2 2 – 2 – 3 3 – 1 – 4 1 – 10 – 5 2 – 1 – 6 3 – 5 – 7 4 – 5 – 8 6 – 3 – 7 7 – 4 – 8 6 – 7 – 9 7 – 2 – 10 9 – 1 – 10 Запуск из вершины 4	Node[0] is visited …. Node[9] is visited Edge[1][0] is included Edge[2][1] is included Edge[3][2] is included Edge[1][5] is included Edge[5][4] is included Edge[5][6] is included Edge[6][7] is included Edge[6][9] is included Edge[9][8] is included Edge[0][4] is not included Edge[4][0] is not included Edge[2][6] is not included Edge[6][2] is not included Edge[3][7] is not included Edge[7][3] is not included Edge[5][8] is not included Edge[8][5] is not included	ВЕРНО
Graph with zero nodes	<empty graph> Запуск из вершины 1	PrimException "Graph can't be empty"	ВЕРНО
Disconnected graph	<disconnected graph> Запуск из вершины 1	PrimException "Graph must be connected"	ВЕРНО

Graph with negative weight	1 – 2 – 2 2 – -3 – 3	PrimException "Weight must be a positive number"	ВЕРНО

5.2. Тестирование структуры класса WeightedConnectedGraph.

Были протестированы следующие методы WeightedConnectedGraph:

1) addNode()
2) addEdge()
3) addWeightedEdge()
4) clear()
5) setWeight()
6) getEdgeAmount()

При этом все тесты были разделены на следующие группы:

1) Граничные случаи.
   А) Добавление первой вершины.
   Б) Получение количества ребер при их отсутствии.
   В) Очистка пустого графа.
2) Позитивные тесты.
   А) Добавление ребра.
   Б) Добавление взвешенного ребра.
   В) Очистка непустого графа.
   Г) Задание веса для невзвешенного ребра.
   Д) Получение количества ребер при их наличии.
3) Негативные тесты.
   А) Добавление ребра, соединяющего одну и ту же вершину.
   Б) Задание отрицательного веса на ребре.

Результаты тестирования класса WeightedConnectedGraph представлены в таблице 2.

Таблица 2:

Название	Вызов методов	Проверки	Комментарии
Test addNode() method	<empty graph> graph.addNode();	graph.getListSize() == 1	ВЕРНО

Test addEdge() method	<empty graph>
	graph.addNode();
	graph.addNode();
	graph.addEdge(1, 2);
		Edge[0][1] exists
		Edge[1][0] exists
			ВЕРНО

Test addWeightedEdge() method	<empty graph>
	graph.addNode();
	graph.addNode();
	graph.addWeightedEdge(1, 2, 10);
		Edge[0][1] exists
		Edge[1][0] exists
		Edge[0][1].weight() == 10
			ВЕРНО

Test clear() method	<empty graph>
	graph.addNode();
	graph.addNode();
	graph.addWeightedEdge(1, 2, 10);
	graph.clear();
		graph.getListSize() == 0	ВЕРНО

Test setWeight() method	<empty graph>
	graph.addNode();
	graph.addNode();
	graph.addEdge(1, 2);
	graph.setWeight(1, 2, 10);
		Edge[0][1] exists
		Edge[1][0] exists
		Edge[0][1].weight() == 10
			ВЕРНО

Test getEdgeAmount() method	<empty graph>
	graph.addNode(); (10 раз)
	graph.addEdge(1, 2);
	graph.addEdge(3, 4);
	graph.addWeightedEdge(5, 6, 10);
	graph.addEdge(7, 6);
		graph.getEdgeAmount() == 4	ВЕРНО

Trying to add edge to the same node	<empty graph>
	graph.addNode();
	try graph.addEdge(1, 1);
		WCGraphException
		"Edge can't connect same nodes"
			ВЕРНО

Trying to add weighted edge to the same node	<empty graph>
	graph.addNode();
	try graph.addWeightedEdge(1, 1, 2);
		WCGraphException
		"Edge can't connect same nodes"
			ВЕРНО

5.3. Тестирование случайной генерации графов RandomGraphGenerator

При тестировании генерации графов были рассмотрены следующие случаи:

1) Граничные случаи.

А) Создание графа с одной вершиной

2) Позитивные случаи.

А) Создание графа с положительным количеством вершин N и количеством ребер <=N-1 с положительными весами.

3) Негативные тесты.

А) Создание графа с количеством вершин <= 0

Б) Создание графа с неправильным диапазоном ребер (меньше заданного пользователем минимума или больше максимума).

В) Создание графа с отрицательным диапазоном весов ребер.

Г) Создание графа с невозможным количеством ребер (<N-1 || >N(N-1)/2)

Результаты тестирования случайной генерации представлены в таблице 3.

Таблица 3:

Название	Параметры	Проверки	Комментарии
Test generation with one node (repeat 10 times)	nodeAmount = 1 minEdgeAmount = 0 maxEdgeAmount = 0 minWeight = 1 maxWeight = 1	graph.getListSize() == 1	ВЕРНО
Test generation with ten nods (repeat 20 times)	nodeAmount = 10 minEdgeAmount = 5 maxEdgeAmount = 10 minWeight = 1 maxWeight = 20	graph.isConnected() graph.getListSize() == 10 graph.getEdgeAmount() <= 10 graph.getEdgeAmount() >= 5 Edge.weight() >= 1 Edge.weight() <= 20	ВЕРНО
Test generation with zero nodes	try nodeAmount = 0 minEdgeAmount = 0 maxEdgeAmount = 0 minWeight = 1 maxWeight = 1	GenerationException "Amount of nodes must be more than 0"	ВЕРНО
Test generation with wrong min edge amount	try nodeAmount = 6 minEdgeAmount = 4 maxEdgeAmount = 10 minWeight = 1 maxWeight = 10	GenerationException "Minimal edge amount can't be less than (node amount - 1)"	ВЕРНО
Test generation with max edge amount less than min	try nodeAmount = 6 minEdgeAmount = 5 maxEdgeAmount = 3 minWeight = 1 maxWeight = 10	GenerationException "Maximum edge amount can't be less than minimum"	ВЕРНО

Test generation with wrong max edge amount	try nodeAmount = 5 minEdgeAmount = 5 maxEdgeAmount = 11 minWeight = 1 maxWeight = 10	GenerationException "Maximum edge amount can't be more than ((node amount)(node amount - 1)/2)"	BEPHO
Test generation with negative min weight	try nodeAmount = 5 minEdgeAmount = 5 maxEdgeAmount = 9 minWeight = -2 maxWeight = 10	GenerationException "Weight must be a positive number"	BEPHO
Test generation with max weight less than min	try nodeAmount = 5 minEdgeAmount = 6 maxEdgeAmount = 9 minWeight = 6 maxWeight = 4	GenerationException "Maximum weight can't be less than minimum"	BEPHO

ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта было разработано графическое приложение на языке Java с использованием таких фреймворков, как Swing, Maven, Junit и др. Приложение позволяет ввести граф любым из нескольких способов (вручную, загрузка из файла формата .json, случайная генерация или генерация с заданными параметрами). Были реализованы тесты для проверки работы алгоритма и структур. Результат работы алгоритма Прима представлен раскраской ребер графа.