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

ОТЧЕТ

по учебной практике

Тема: Минимальное остовное дерево. Алгоритм Краскала

Студент гр. 2303		Синотова А.А.
Студент гр. 2300		Перебейнова М.С.
Студент гр. 2384		Валеева А.А.
Руководитель		Шестопалов Р.П.

Санкт-Петербург
2023

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Синотова А.А. группы 2303

Студент Перебейнова М.С. группы 2300

Студент Валеева А.А. группы 2384

Тема практики: Минимальное остовное дерево. Алгоритм Краскала

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Минимальное остовное дерево. Алгоритм Краскала

Сроки прохождения практики: 26.06.2024 – 9.07.2020

Дата сдачи отчета: 8.07.2024

Дата защиты отчета: 8.07.2024

Студент гр. 2303		Синотова А.А.
Студент гр. 2300		Перебейнова М.С.
Студент гр. 2384		Валеева А.А.
Руководитель		Шестопалов Р.П.

АННОТАЦИЯ

В процессе выполнения практического задания студенты не только применяют теоретические знания на практику, но и развивают навыки коммуникации, критического мышления и решения проблем в условиях командной работы. Каждый этап разработки требует анализа, проектирования, реализации и тестирования, что способствует формированию полноты технических компетенций и умений управления проектами.

Важным аспектом практики является также обучение культуре кодирования, включая стандарты написания кода, использование систем контроля версий (например, Git) и принципы работы в команде. Студенты учатся взаимодействовать с коллегами, обсуждать идеи, принимать конструктивную критику и вместе находить оптимальные решения.

После завершения каждого этапа разработки студенты предоставляют отчет, в котором документируют свой процесс, достижения, проблемы и найденные решения. Это помогает не только руководителю оценить прогресс студентов, но и самим участникам лучше понять свои успехи и области для улучшения.

Таким образом, практическая работа над проектом визуализации алгоритма Краскала не только дает ценный опыт в разработке программного обеспечения, но и способствует личностному развитию студентов, делая их более готовыми к профессиональной деятельности в сфере информационных технологий.

SUMMARY

In the process of completing a practical assignment, students not only apply theoretical knowledge to practice, but also develop communication skills, critical thinking and problem solving in a team environment. Each stage of development requires analysis, design, implementation and testing, which contributes to the formation of completeness of technical competencies and project management skills.

An important aspect of the practice is also teaching coding culture, including code writing standards, the use of version control systems (for example, Git) and teamwork principles. Students learn to interact with colleagues, discuss ideas, accept constructive criticism and find optimal solutions together.

After completing each stage of development, students submit a report documenting their process, achievements, problems, and solutions found. This helps not only the supervisor to assess the progress of the students, but also the participants themselves to better understand their successes and areas for improvement.

Thus, practical work on the Kraskal algorithm visualization project not only provides valuable experience in software development, but also contributes to the personal development of students, making them more ready for professional activities in the field of information technology.

СОДЕРЖАНИЕ

	Введение	6
1.	Требования к программе	8
1.1.	Исходные требования к программе	8
1.1.1.	Требования к визуализации работы алгоритма	9
1.1.2	Требования к визуализации пользовательского интерфейса	9
1.1.3	Требования к входным данным	11
2.	План разработки и распределение ролей в бригаде	13
2.1.	План разработки	13
3.	Особенности реализации	15
3.1.	Структуры данных	15
4.	Тестирование	29
4.1	Тестирование графического интерфейса	29
4.2	Тестирование сохранения и загрузки графа	29
4.3	Тестирование визуализации алгоритма	29
	Заключение	30
	Список использованных источников	32
	Приложение А. Исходный код программы	33
	Приложение В. Результаты тестирования программы	66

ВВЕДЕНИЕ

Для достижения поставленной цели практики по визуализации алгоритма Краскала на языке Java, необходимо следовать следующему плану действий:

1. Изучение алгоритма

Алгоритм Краскала является одним из основных методов построения минимального остовного дерева (MST) для взвешенных связных неориентированных графов. Он использует принцип "жадности", выбирая на каждом шаге наименьший доступный ребро, которое не образует цикл с уже добавленными рёбрами. Алгоритм Краскала работает за линейное время $O(E \log E)$, учитывая, что сортировка рёбер требует $O(E \log E)$ времени.

2. Составление спецификации разработки приложения

- Описание: Разработать консольное или графическое приложение, которое позволяет пользователю вводить данные о графе (вершины и рёбра с весами) и выводит результат работы алгоритма Краскала.
- Функциональные требования: ввод данных о графе, вывод результата работы алгоритма Краскала.
- Нефункциональные требования: простота использования, надёжность работы.

3. План разработки и распределение ролей

Архитектура: Разделить проект на модули: ввод данных, обработка данных, алгоритм Краскала, визуализация.

4. Написание и отладка модулей проекта

- Модуль ввода данных: Реализовать функционал для чтения данных о графе из файла или через консоль.
- Модуль обработки данных: Сортировка рёбер по весам.
- Модуль алгоритма Краскала: Реализация самого алгоритма.
- Модуль визуализации: Отображение графа и его остовного дерева.

5. Компиляция модулей в один проект и отладка полного приложения

После написания всех модулей, их нужно скомпилировать вместе и провести тестирование всего приложения, чтобы убедиться, что все части корректно взаимодействуют друг с другом.

6. Представление проекта руководителю

Подготовить презентацию или документацию, описывающую работу над проектом, включая выбранную архитектуру, использованные технологии, особенности реализации алгоритма Краскала и результаты тестирования.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

Исходные Требования к программе

Исходную структуру программы можно представить в качесте UML диаграммы, изображённой на рис. 1.

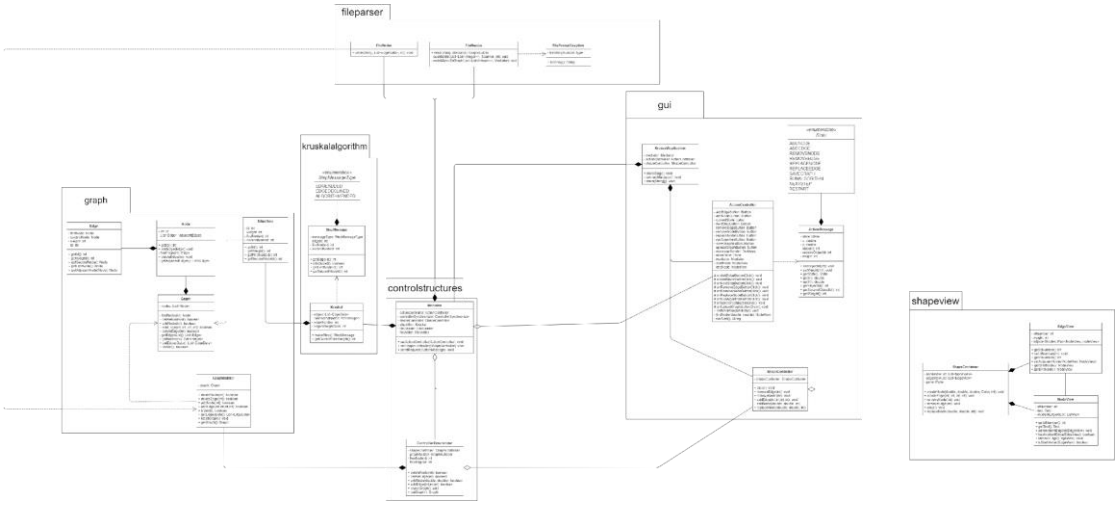

Рисунок 1 – UML-диаграмма классов разрабатываемой программы

Для обеспечения плавности и удобства использования, программа предусматривает интуитивно понятный интерфейс, который минимизирует необходимость в дополнительных инструкциях для пользователя. Интерфейс позволяет легко вводить данные, которые затем обрабатываются алгоритмом Краскала. После завершения обработки данных, программа автоматически перенаправляет пользователя к новому окну, где представлен подробный анализ выполненных алгоритмом шагов. Это окно может содержать визуализацию промежуточных решений, позволяющие пользователю самостоятельно исследовать процесс работы алгоритма.

Кроме того, программа может включать функционал для сохранения результатов работы алгоритма и последующего их анализа. Это может быть особенно полезно для образовательных целей, позволяя студентам и преподавателям изучать алгоритмы на конкретных примерах и повторять материал в любое время.

Таким образом, программа не только выполняет задачу визуализации алгоритма Краскала, но и предоставляет комплексный инструментарий для

глубокого понимания процесса решения задачи, делая её ценным инструментом как для специалистов в области информационных технологий, так и для учащихся, изучающих алгоритмы и структуры данных.

Требования к визуализации работы алгоритма

Программа, посвящённая визуализации алгоритма Краскала для поиска минимального остовного дерева, представляет собой уникальный инструмент для изучения и понимания принципов работы одного из ключевых алгоритмов в области сетевого моделирования и оптимизации. С помощью этой программы пользователи могут не только наблюдать за процессом построения минимального остовного дерева, но и активно участвовать в его создании, следуя за каждым шагом алгоритма.

В начале работы с программой пользователю предлагается выбрать способ создания графа: это может быть ручной ввод данных через графический интерфейс или загрузка предварительно подготовленных данных из файла. Такой подход обеспечивает гибкость и позволяет работать как с простыми, так и со сложными графами, адаптируя условия задачи под конкретные нужды исследования.

После создания графа алгоритм Краскала начинает свою работу, постепенно добавляя ребра в дерево, чтобы достичь минимального остовного дерева. Каждый шаг алгоритма визуализируется, что позволяет пользователю точно следить за процессом и понимать, как алгоритм выбирает определённые ребра для включения в остовное дерево. Эта визуализация не только упрощает понимание алгоритма, но и делает процесс обучения более интерактивным и увлекательным.

Требования к визуализации пользовательского интерфейса

Интерфейс представлен на рис. 2.

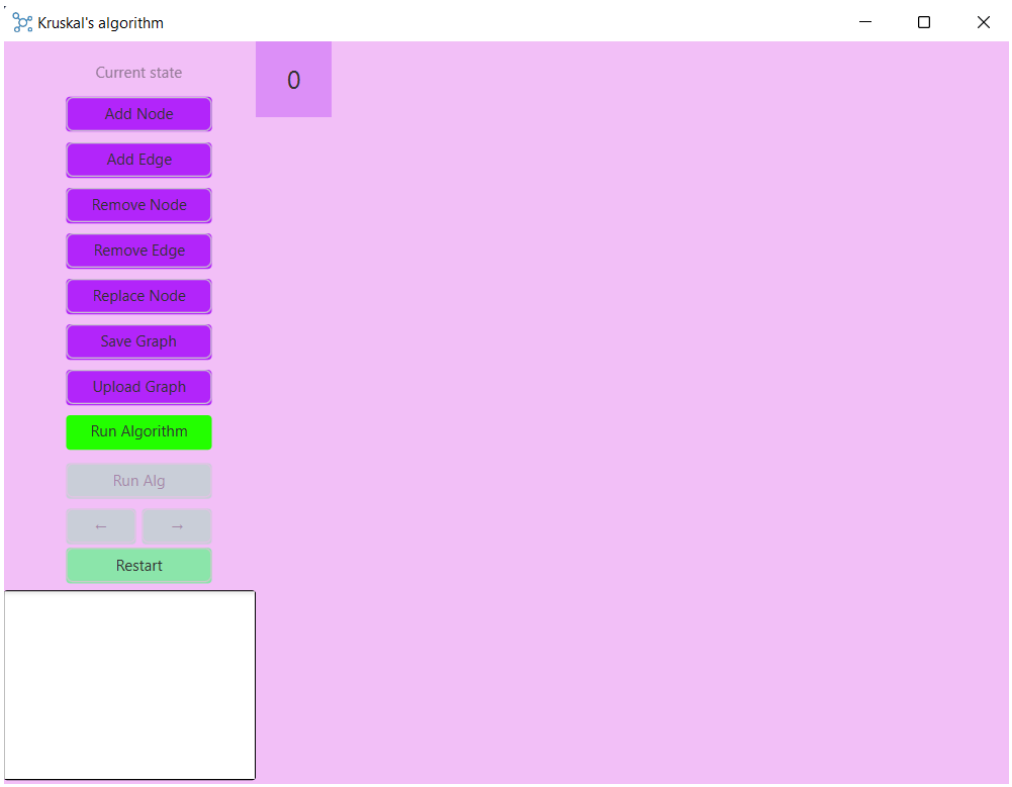

Рисунок 2 – Пользовательский интерфейс приложения в начале его работы

В начале запускается окно, в котором большую часть занимает зона для отрисовки, где будет находится граф, справа находится панель управления с кнопками, в правом нижнем углу находится окно для предоставления информационных сообщений при исключениях, а также при непосредственной работе алгоритма.

При создании графа, кнопка “Next Step” заблокирована, т.к. алгоритм ещё не начал своей работы. Кнопка “Save Graph” позволяет сохранить граф в файл текстовый файл. Кнопки “Add Node”, “Add Edge” переводят программу в режим добавления вершин либо рёбер соответственно, т.е. при нажатии на рабочее поле в режиме добавления вершин будет появляться вершина в указанной точке, а при нажатии на две вершины в режиме добавления рёбер будет появляться ребро между выбранными вершинами. Кнопки “Remove Node” и “Remove Edge” переводят программу в режим удаления вершин или рёбер. При нахождении в режиме удаления нажатие на объект соответствующий режиму приведёт к удалению этого объекта. При удалении вершины, все инцидентные ей рёбра тоже удаляются. Формат представления графа в файле будет представлен в п.1.1.3.

После того, как пользователь нажмёт кнопку “Run Algorithm”, кнопки связанные с созданием и сохранением графа будут заблокированы для пользователя, а кнопка “Next Step” станет доступной. Это можно увидеть на рис. 2.

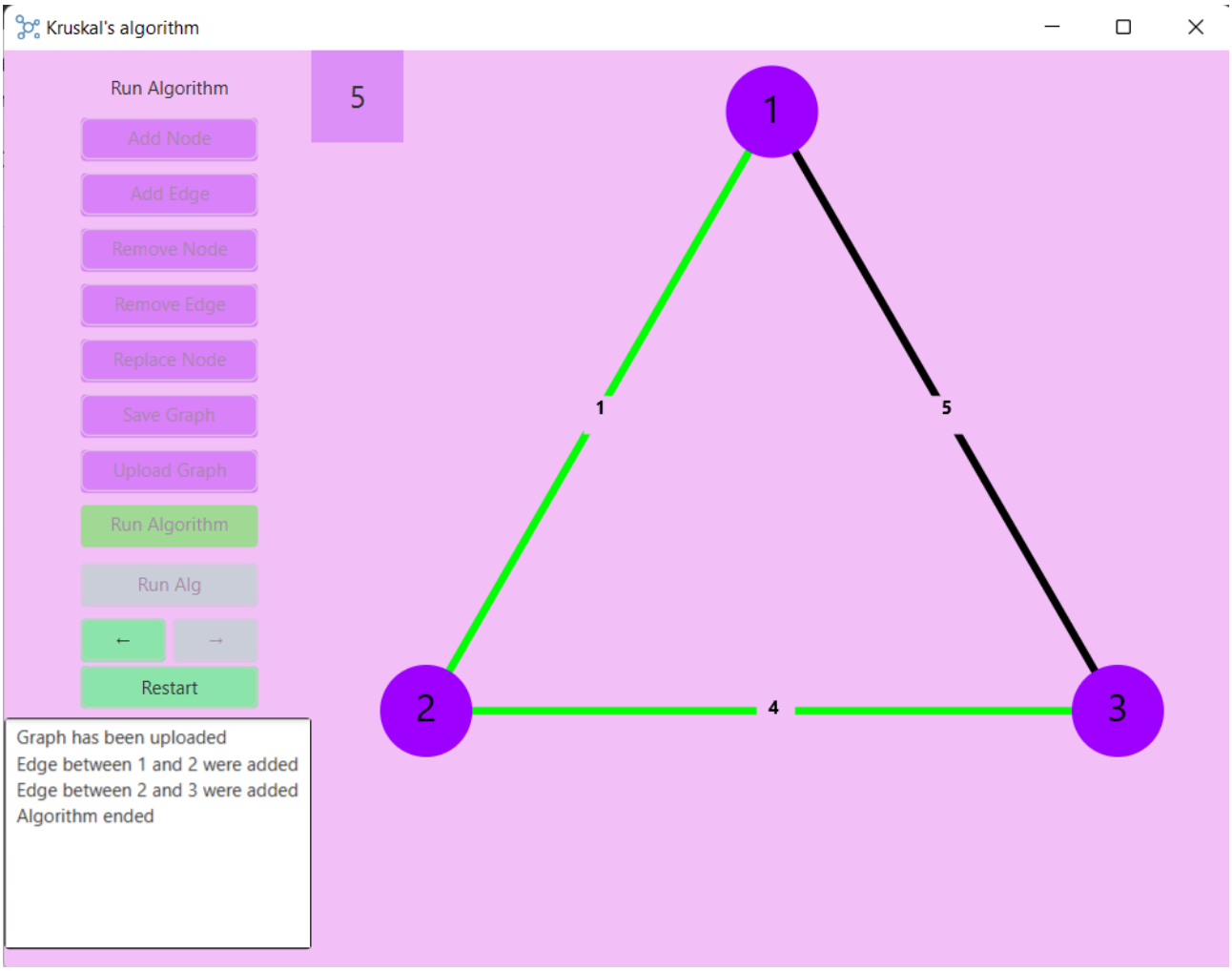

Рисунок 3 – Пользовательский интерфейс, после начала работы алгоритма

Кнопка “Restart” доступна всегда, она возвращает программу в начальное состояние как на рис. 2.

Как можно заметить на рис. 3 при работе алгоритма в окне сообщений выводится информация о пройденном шаге алгоритма. Вместо сообщения “Next step!” будет выводиться сообщение, поясняющее данный шаг алгоритма.

Требования к входным данным

Пользователи имеют возможность представить граф двумя разными методами, что обеспечивает гибкость и удобство в работе с различными типами данных и задачами. Первый способ — загрузка графа из текстового файла (.txt), который содержит информацию о числе вершин графа и матрице смежности. Для

этого достаточно нажать кнопку "Upload File" в графическом интерфейсе программы. Этот метод особенно удобен для работы с большими графами или когда данные уже существуют в текстовом формате. Он позволяет быстро и без ошибок импортировать граф в программу, минимизируя вероятность ввода ошибок при ручном вводе данных.

Второй способ создания графа предусматривает использование инструментов, доступных непосредственно в пользовательском интерфейсе программы. Это может включать в себя визуальное добавление вершин и ребер, установку весовых коэффициентов для ребер, а также другие параметры, связанные с структурой графа. Такой подход обеспечивает высокую степень контроля над процессом создания графа и позволяет пользователю сразу же видеть результаты своих действий в виде визуализации графа. Это особенно полезно для экспериментов, обучения или быстрой проверки концепций без необходимости предварительной подготовки файлов.

Оба этих метода интегрированы в единый интерфейс программы, что позволяет пользователю свободно переключаться между ними в зависимости от текущих потребностей и предпочтений. Такая архитектура обеспечивает удобство использования программы широким кругом пользователей, начиная от студентов и заканчивая профессионалами в области сетевого моделирования и алгоритмизации.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

План разработки и распределение ролей в бригаде

Составление спецификации. Исполнители: Валеева А.А., Перебейнова М.С., Синотова А.А. Скрок сдачи: 28.06

Составление плана разработки. Исполнители: Валеева А.А., Депрейс, А.С. Синотова А.А. Скрок сдачи: 28.06

Реализация графического интерфейса:

- Реализация области отображения графа. Исполнитель: Синотова А.А. Срок сдачи: 01.07
- Реализация графического интерфейса для редактирование графа и работы алгоритма. Исполнитель: Синотова А.А. Срок сдачи: 01.07

Реализация работы с файлом:

- Реализация сохранения графа в файл. Исполнитель: Валеева А.А. Срок сдачи: 08.07
- Реализация загрузки графа из файла. Исполнитель: Валеева А.А. Срок сдачи: 08.07

Реализация логики программы.

- Реализация алгоритма Краскала. Исполнитель: Перебейнова М.С. Срок сдачи: 05.07
- Реализация класса графа и его создания. Исполнитель: Перебейнова М.С. Срок сдачи: 01.07
- Реализация классов для вершины и дуги. Исполнитель: Перебейнова М.С. Срок сдачи: 01.07
- Создание объектов для передачи сообщений о работе алгоритма. Исполнитель: Перебейнова М.С. Срок сдачи: 05.07

Реализация связующих структур между графической и логической частью программы. Исполнители: Валеева А.А., Перебейнова М.С. Срок сдачи: 5.07

Реализация визуализации работы алгоритма:

Добавление возможности запуска алгоритма и просмотра каждого его шага. Исполнитель: Синотова А.А. Срок сдачи: 5.07

Реализация API для связи с логикой программы. Исполнитель: Синотова А.А. Скрок сдачи: 5.07

Внесение корректировок, исправление багов и тестирование программы:

Тестирование и исправление графической части программы. Исполнитель: Синотова А.А. Скрок сдачи: 8.07

Тестирование и исправление сохранения/загрузки графа. Исполнитель: Валеева А.А. Скрок сдачи: 08.07

Тестирование и исправление логики программы. Исполнитель: Перебейнова М.С. Скрок сдачи: 08.07

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

В данном проекте структуры данных можно разбить на 6 условных групп: реализация алгоритма Краскала, построение и хранение графа для внутренней логики, построение и хранение графа для отрисовки, работа с файлами, графический интерфейс, связь интерфейса с остальной программой.

Классы реализующие алгоритм Краскала:

Класс Kruska - предназначен для реализации алгоритма Краскала, который используется для построения минимального остовного дерева (MST) взвешенного связного неориентированного графа. Этот алгоритм позволяет найти подграф, который соединяет все вершины исходного графа и имеет минимальную суммарную длину ребер.

Как работает класс:

Инициализация: При создании экземпляра класса Kruskal, конструктор принимает объект Graph, представляющий исходный граф. Конструктор инициализирует необходимые структуры данных для хранения информации о шагах алгоритма, ребрах графа, узлах и их связи.

Сортировка ребер: Ребра графа сортируются по весу в порядке возрастания. Это позволяет алгоритму последовательно рассматривать ребра с наименьшим весом.

Построение MST: Алгоритм перебирает отсортированные ребра и на каждом шаге добавляет к MST ребро с наименьшим весом, которое не образует цикл с уже добавленными ребрами. Для проверки на наличие цикла используются множества, представляющие компоненты связности графа.

Отслеживание шагов: На каждом шаге алгоритма создается объект StepMessage, который содержит информацию о добавленном ребре, его весе и типе операции (добавлено или отклонено). Эти шаги сохраняются в списке steps, что позволяет отслеживать процесс построения MST.

Отмена шага: Метод stepBack позволяет отменить последний шаг алгоритма, что может быть полезно для визуализации процесса или для исправления ошибок в данных.

Получение веса MST: Метод getCurrentTreeWeight возвращает текущий общий вес MST, что позволяет оценить эффективность построенного остовного дерева.

Классы отвечающие за представление графа для логики программы:

Edge - класс предназначен для представления ребра в графе.

Как работает класс:

Конструктор: Конструктор класса принимает четыре параметра: две вершины (firstNode и secondNode), вес ребра (weight) и уникальный идентификатор ребра (id). Эти параметры инициализируют соответствующие поля класса.

Методы доступа: Класс предоставляет геттеры для получения веса ребра, идентификатора ребра, первой и второй вершин. Эти методы позволяют получить доступ к основной информации о ребре.

Метод equals: Переопределенный метод equals позволяет сравнить два объекта класса Edge на основе равенства их первых и вторых вершин.

Метод getAdjacentNode: Метод возвращает соседнюю вершину данного ребра относительно указанной вершины.

Метод toString: Переопределенный метод toString предоставляет строковое представление объекта Edge, включая идентификатор ребра, идентификаторы вершин и вес ребра. Это удобно для отладки и визуализации графа.

EdgeData – класс, хранящий информацию о ребре графа и предоставляющий её другим другим классам.

Как работает класс:

Конструктор: Конструктор класса принимает объект класса Edge в качестве параметра. Этот объект содержит всю необходимую информацию о ребре, включая идентификатор ребра, вес и идентификаторы связанных вершин.

Конструктор извлекает эту информацию и инициализирует соответствующие поля класса EdgeData.

Методы доступа: Класс предоставляет геттеры для получения идентификатора ребра, его веса и идентификаторов связанных вершин. Эти методы обеспечивают доступ к ключевой информации о ребре, которая может быть необходима для алгоритмов работы с графами или для визуализации графа.

Node – класс, хранящий информацию о вершине графа.

Как работает класс:

Конструктор: Конструктор принимает идентификатор вершины (id) в качестве параметра и инициализирует список смежных ребер (adjacentEdges), который будет содержать все ребра, связанные с данной вершиной.

Добавление ребра: Метод addEdge позволяет добавить новое ребро к вершине. При добавлении ребра оно автоматически добавляется в списки смежных ребер обеих вершин, участвующих в ребре.

Поиск ребра: Метод findEdge позволяет найти ребро в списке смежных ребер по его идентификатору. Это может быть полезно для удаления или модификации ребра.

Удаление ребра: Метод deleteEdge удаляет ребро из списка смежных ребер данной вершины и из списка смежных ребер другой вершины, к которой относится это ребро.

Доступ к смежным ребрам: Метод getAdjacentEdges возвращает список смежных ребер, связанных с данной вершиной.

Методы equals и toString: Переопределенные методы equals и toString обеспечивают корректное сравнение объектов класса Node и предоставляют строковое представление вершины соответственно.

Graph - класс, предназначенный для представления взвешенного связного неориентированного графа. Он предоставляет структуру для хранения вершин (Node) и ребер (Edge), а также методы для добавления и удаления вершин и ребер, поиска вершин и ребер по их идентификаторам, и для валидации графа.

Как работает класс:

Хранение вершин: Граф хранит список вершин (nodes), каждая из которых может иметь набор смежных ребер.

Поиск вершин: Метод findNode позволяет найти вершину по её идентификатору.

Добавление и удаление вершин: Методы addNode и deleteNode позволяют добавлять и удалять вершины из графа соответственно. При удалении вершины также удаляются все ребра, связанные с ней.

Добавление и удаление ребер: Методы addEdge и deleteEdge позволяют добавлять и удалять ребра между вершинами. При добавлении ребра проверяется наличие дубликатов ребер по идентификатору.

Получение списка всех ребер: Метод getEdgesList генерирует список всех ребер графа, обеспечивая возможность работы с графом вне зависимости от того, какие вершины были напрямую добавлены в граф.

Валидация графа: Метод isValid проверяет, является ли граф связным, путём просмотра всех вершин и их смежных ребер.

Печать графа: Метод printGraph позволяет визуально представить граф, печатая информацию о каждой вершине и её смежных ребрах.

Получение идентификаторов вершин: Метод getNodesId возвращает множество идентификаторов всех вершин графа.

Получение данных о ребрах: Метод getEdgesData возвращает список объектов EdgeData, содержащих упрощённые данные о ребрах графа, что может быть полезно для алгоритмов и визуализации.

GraphBuilder – класс, позволяющий строить граф и получать информацию о нём, для классов их других пакетов.

Как работает класс

Инициализация графа: В конструкторе класса GraphBuilder создаётся новый экземпляр класса Graph, который будет использоваться для хранения и манипуляции графом.

Добавление и удаление вершин и ребер: Методы addNode, deleteNode, addEdge, и deleteEdge предоставляют прямой интерфейс для добавления и

удаления вершин и ребер графа, делегируя вызовы соответствующим методам класса Graph.

Проверка валидности графа: Метод isValid позволяет проверить, является ли граф связным, делегируя вызов метода isValid класса Graph.

Печать графа: Метод printGraph позволяет визуально представить граф, делегируя вызов метода printGraph класса Graph.

Получение данных о ребрах: Метод getEdgesData возвращает список объектов EdgeData, содержащих упрощённые данные о ребрах графа, делегируя вызов метода getEdgesData класса Graph.

Сброс графа: Метод resetGraph позволяет полностью сбросить состояние графа, заменяя текущий граф новым пустым экземпляром класса Graph.

Получение графа: Метод getGraph возвращает текущий экземпляр графа, позволяя клиенту получить доступ к нему напрямую, если это необходимо.

GraphConnectednessExeption – класс исключение, которое выбрасывается при неподходящей структуре (несвязности) графа при запуске алгоритма.

Как работает класс:

Наследование: Класс GraphConnectednessException наследуется от класса RuntimeException, что делает его unchecked exception. Это означает, что исключение может быть выброшено во время выполнения программы без необходимости явного указания в сигнатуре метода.

Конструктор: Конструктор класса принимает строку сообщения (message), которая описывает причину возникновения исключения. Эта строка передаётся в суперкласс через вызов super(message).

Переопределение метода toString: Метод toString переопределён для предоставления строкового представления исключения, включающего в себя сообщение об ошибке. Это позволяет легче диагностировать причину возникновения исключения при отладке программы.

Классы отвечающие за представление графа для графического интерфейса:

EdgeView – класс, хранящий информацию, для отрисовки рёбер.

Как работает класс:

Конструкторы: Предоставляют различные способы создания объектов EdgeView. Один конструктор принимает координаты для рисования линии, а другой позволяет задать только идентификатор и вес ребра, автоматически генерируя остальные параметры.

Методы getIdNumber и setIdNumber: Геттер и сеттер для работы с идентификатором ребра.

Метод getWeightText: Геттер для получения текстового представления веса ребра.

Метод setAdjacentNodes: Позволяет установить узлы, между которыми проходит данное ребро.

Методы getStartNode и getEndNode: Геттеры для получения начального и конечного узлов соответственно.

NodeView – класс, хранящий информацию, для отрисовки вершин.

Как работает класс

Конструктор: Создает новый узел с указанными координатами, радиусом, цветом и идентификатором. Также устанавливает текстовое представление идентификатора внутри узла.

Методы getIdNumber и setIdNumber: Геттер и сеттер для работы с идентификатором узла. При изменении идентификатора также обновляется текстовое представление.

Метод getText: Геттер для получения текстового представления идентификатора узла.

Метод addIncidentEdgeId: Добавляет ребро в список инцидентных ребер данного узла.

Метод hasIncidentEdge: Проверяет, содержит ли узел указанное ребро среди своих инцидентных ребер.

Метод removeEdge: Удаляет указанное ребро из списка инцидентных ребер узла.

Метод isStartVertex: Проверяет, является ли данный узел начальной точкой указанного ребра.

ShapeContainer – класс, хранящий информацию о всех рёбрах и вершинах, которые надо отобразить, а так же предоставляющий методы для редактирования информации о графе.

Метод createNode: Добавляет новый узел в граф с заданными координатами, радиусом, цветом и идентификатором. После добавления узла вызывается метод redrawGraph для перерисовки графа.

Метод createEdge: Создает новое ребро между двумя узлами с заданным весом и идентификатором. Определяет начальный и конечный узлы ребра, обновляет их списки инцидентных ребер и перерисовывает граф.

Метод removeNode: Удаляет узел из графа по его идентификатору, удаляет все связанные с этим узлом ребра и перерисовывает граф.

Метод removeEdge: Удаляет ребро из графа по его идентификатору, обновляет списки инцидентных ребер для связанных узлов и перерисовывает граф.

Метод redrawGraph: Очищает панель и заново добавляет все узлы и рёбра, а также тексты весов рёбер.

Метод clear: Очищает все узлы, рёбра и панель.

Метод replaceNode: Заменяет положение узла с заданным идентификатором на новые координаты, обновляет позиции связанных с ним рёбер и перерисовывает граф.

Метод colorEdge: Изменяет цвет ребра по его идентификатору.

Классы отвечающие за работу с файлами:

FileReader – класс, реализущий считывание информации о графе из файла, проверку корректности формата файла и отправку сигналов, для построения графа, по считанной информации.

Как работает класс

Чтение файла: Метод read начинает процесс чтения файла, проверяет его тип и открывает файл для чтения. Если файл не является текстовым, генерируется исключение FileFormatException.

Парсинг первой строки: Первая строка файла должна содержать количество вершин в графе. Парсер проверяет, что строка содержит именно одно целое число, соответствующее количеству вершин, и что это число не превышает допустимые пределы.

Парсинг матрицы смежности: Для каждой вершины считывается строка, представляющая её связи с другими вершинами. Строка разбивается на отдельные элементы, каждый из которых представляет собой вес ребра между вершинами. Парсер проверяет, что матрица смежности является симметричной и что нет ненулевых значений на главной диагонали.

Преобразование в граф: После успешного парсинга данных о вершинах и ребрах, метод nodeMatrixToGraph использует медиатор для отправки запросов на добавление вершин и ребер в граф.

Расположение вершин: Метод nodeReader определяет положение вершин на экране, исходя из количества вершин и размеров окна. Расположение вершин рассчитывается таким образом, чтобы минимизировать переполнение окна и обеспечить равномерное распределение вершин.

FileWriter – класс, реализующий запись графа в файл, в корректном формате.

Как работает класс

Проверка типа файла: Перед началом записи в файл проверяется его тип. Если файл не является текстовым, генерируется исключение FileFormatException.

Создание файла: Если файл не существует, он создаётся. В случае возникновения ошибок при создании файла генерируется исключение FileFormatException.

Открытие файла для записи: Файл открывается для записи с использованием PrintWriter. В случае возникновения ошибок при открытии файла генерируется исключение FileFormatException.

Запись количества вершин: В начале файла записывается количество вершин графа.

Преобразование идентификаторов: Используется вспомогательный класс IdConverter для преобразования идентификаторов вершин из одного представления в другое. Это необходимо для обеспечения уникальности идентификаторов при сохранении и последующем восстановлении данных.

Запись ребер: Для каждой пары вершин записывается вес ребра между ними. Если ребро отсутствует, записывается значение 0. Все данные разделяются пробелами для удобства последующего чтения.

Закрытие файла: После завершения записи файл закрывается.

Вспомогательный класс *IdConverter*

Класс *IdConverter* используется для преобразования идентификаторов вершин между различными представлениями. Он хранит карту от старых идентификаторов к новым и обратно, что позволяет легко сопоставлять вершины при сохранении и восстановлении данных графа.

FileFormatExeption – класс исключение, которое выбрасывается классами FileWriter и FileReader, при проблемах с файлом или его содержимым.

Как работает класс

Наследование: Класс FileFormatException наследуется от RuntimeException, что делает его unchecked exception. Это означает, что исключение может быть выброшено во время выполнения программы без необходимости явного указания в сигнатуре метода.

Конструктор: Конструктор класса принимает два параметра: строку сообщения (message), описывающую общую информацию об ошибке, и номер строки файла (fileStringNumber), указывающий на место в файле, где произошла ошибка. Эти параметры передаются в суперкласс через вызов super(message), а номер строки файла сохраняется в приватном поле класса.

Переопределение метода toString: Метод toString переопределён для предоставления строкового представления исключения, включающего в себя сообщение об ошибке и номер строки файла. Это позволяет легче диагностировать причину возникновения исключения при отладке программы.

Классы реализующие графический интерфейс программы:

KruscalApplication – класс, запускающий программу и создающий основные объекты для её дальнейшей работы.

Как работает класс

Инициализация JavaFX приложения: Класс KruskalApplication наследуется от Application, что делает его основным классом для запуска JavaFX приложения.

Загрузка FXML: В методе start загружается FXML-файл, который определяет структуру и поведение GUI. Из FXML-файла извлекается контроллер действий (ActionController), который управляет логикой обработки пользовательских действий.

Настройка контроллеров: Создается экземпляр ShapeController, который отвечает за визуализацию графа и его элементов. Затем устанавливаются связи между ActionController, ShapeController и Mediator, который обеспечивает коммуникацию между этими компонентами.

Подготовка и отображение окна: Окну приложения задается иконка, заголовок и сцена, включающая корневой узел (AnchorPane). После настройки окно отображается пользователю.

Подключение посредника: Метод connectMediator устанавливает связи между mediator, actionController и shapeController, что позволяет эффективно обмениваться данными и командами между различными частями приложения.

ActionController – класс, отвечающий за работу кнопок управления и окна с информацией.

Как работает класс

Управление GUI: Класс содержит ссылки на различные элементы интерфейса, такие как кнопки, метки и поля ввода, которые используются для взаимодействия с пользователем. Эти элементы аннотированы с помощью

@FXML, что позволяет JavaFX автоматически связывать их с соответствующими полями в классе после загрузки FXML-файла.

Обработка действий пользователя: Методы, аннотированные @FXML, обрабатывают события, происходящие при взаимодействии пользователя с интерфейсом. Например, нажатие кнопки запускает алгоритм, а клик мышью на панели добавляет вершину или ребро в граф.

Коммуникация с бизнес-логикой: Через интерфейс Mediator класс ActionController отправляет запросы на выполнение операций, таких как добавление вершины или ребра, запуск алгоритма и т.д., и получает обратно результаты этих операций. Это позволяет отделить логику GUI от бизнес-логики приложения, упрощая тестирование и поддержку кода.

Обновление интерфейса: В зависимости от результатов выполнения операций, класс ActionController обновляет интерфейс, показывая текущее состояние графа, результаты работы алгоритма и сообщения об ошибках.

ActionMessage – класс, реализующий сообщения, с помощью которых компоненты программы передают информацию друг другу.

Как работает класс

Конструкторы: Класс предоставляет несколько конструкторов для создания объектов ActionMessage с различным набором параметров. Это позволяет создавать сообщения для различных типов действий, таких как добавление вершины или ребра, запуск алгоритма, загрузка или сохранение графа. Некоторые конструкторы позволяют указать только базовое состояние действия, в то время как другие могут включать дополнительные параметры, такие как координаты, идентификаторы объектов и вес ребра.

Геттеры и сеттеры: Для каждого свойства класса предусмотрены геттеры и сеттеры, что позволяет получить доступ к значениям свойств и изменить их после создания объекта ActionMessage.

Файловое имя: Добавление свойства fileName позволяет использовать один и тот же класс для сообщений, связанных с операциями загрузки и сохранения файлов, упрощая обработку таких операций.

ShapeController – класс, для управления информацией в ShapeContainer.

Как работает класс

Инициализация: Конструктор класса принимает объект Pane, который представляет собой контейнер для графических элементов. Внутри конструктора создается экземпляр ShapeContainer, который будет использоваться для управления графическим состоянием.

Добавление и удаление узлов и ребер: Методы addNode, removeNode, addEdge, и removeEdge позволяют добавлять и удалять узлы и ребра в графе соответственно. Эти методы используют ShapeContainer для непосредственного взаимодействия с графическим представлением.

Замена узла: Метод replaceNode позволяет заменить положение узла в графе, что может быть полезно для корректировки позиций узлов после их добавления.

Изменение цвета ребер: Методы paintEdge и paintEdgeDefault используются для изменения цвета ребер в соответствии с результатами работы алгоритма Краскала. Они принимают сообщения о шагах алгоритма (StepMessage), чтобы определить, следует ли окрашивать ребро зелёным цветом (если ребро было успешно добавлено) или красным (если ребро было отклонено).

State – перечисление, которое содержит типы состояний, в которых может находится программа.

Классы, реализующие связь компонентов программы:

Mediator – класс, который вызывает методы внутренней логики программы, в зависимости от состояния программы.

Как работает класс

Инициализация: В конструкторе создаются экземпляры FileReader и FileWriter, которые используются для чтения и записи данных графа в файлы.

Связывание компонентов: Методы setActionController и setShapeController позволяют установить связи с ActionController и ShapeController соответственно. При установлении ShapeController также создается ControllerSynchronizer,

который обеспечивает синхронизацию состояния графа между GUI и бизнес-логикой.

Обработка запросов: Метод sendRequest принимает объект ActionMessage, содержащий информацию о действии, которое должно быть выполнено. В зависимости от типа действия (добавление узла, ребра, удаление, загрузка/сохранение графа, запуск алгоритма и т.д.) вызываются соответствующие методы у controllerSynchronizer и shapeController, а также выполняется чтение/запись файлов при необходимости.

Выполнение алгоритма Краскала: Запросы на запуск алгоритма (RUNALGORITHM, RUNALG, NEXTSTEP, PREVIOUSSTEP) приводят к созданию экземпляра Kruskal и последовательному выполнению шагов алгоритма, с обновлением GUI на каждом шаге.

ControllerSynchronizer – класс, позволяющий синхронизировать обновления информации о графе в логической и графических частях, с помощью вызовов соответствующих методов у ShapeController и GraphBuilder.

Как работает класс

Инициализация: В конструкторе устанавливаются ссылки на ShapeController и GraphBuilder, а также инициализируются переменные для управления идентификаторами узлов и ребер.

Добавление узлов и ребер: Методы addNode и addEdge используют GraphBuilder для добавления узлов и ребер в граф соответственно. После успешного добавления узла или ребра в граф, они также добавляются в визуальное представление через ShapeController, и идентификаторы увеличиваются на единицу для предотвращения коллизий.

Удаление узлов и ребер: Методы deleteNode и deleteEdge используют GraphBuilder для удаления узлов и ребер из графа. После успешного удаления, соответствующие элементы удаляются из визуального представления через ShapeController.

Очистка графа: Метод eraseGraph сбрасывает все идентификаторы узлов и ребер до начальных значений и очищает граф и визуальное представление, используя GraphBuilder и ShapeController.

Получение графа: Метод getGraph позволяет получить текущее состояние графа, построенного с помощью GraphBuilder.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

На рисунках 4 - 10 в приложении В представлена демонстрация тестирования функционала модификации графа с помощью графического интерфейса. Были проверены все заявленные модификации.

4.2. Тестирование сохранения и загрузки графа

На рисунках 11 - 18 в приложении В представлена демонстрация тестирования сохранения и загрузки графа. Загрузка производилась в том числе из файлов с неверным форматом.

4.3 Тестирования визуализации алгоритма

На рисунках 19 - 21 в приложении В представлена демонстрация тестирования визуализации алгоритма на различных графах.

ЗАКЛЮЧЕНИЕ

Во время учебной практики студенты имели возможность углубленно изучить Java, строго типизированный объектно-ориентированный язык программирования, который известен своей надежностью и масштабируемостью. Изучение алгоритма Краскала было ключевым моментом практики, поскольку этот алгоритм является одним из основных методов поиска минимального остовного дерева в графе. Хотя алгоритм Прима часто рассматривается как альтернатива, его преимуществом является именно простота реализации. Однако, как отмечено, алгоритм Прима требует сортировки рёбер графа по возрастанию, что может стать узким местом при работе с большими графами.

В рамках практики особое внимание уделялось изучению JavaFX, библиотеки для создания графического интерфейса в Java. JavaFX позволяет разработчикам создавать богатые и интерактивные пользовательские интерфейсы, что является критически важным для современных приложений. Использование JavaFX позволило студентам реализовать функционал для визуализации алгоритма Краскала, делая процесс обучения более наглядным и интересным.

Результатом коллективного труда группы стало приложение, разработанное на Java, которое успешно визуализирует алгоритм Краскала для поиска остовного дерева в связном взешенном ненаправленном графе. Приложение предоставляет пользователю два варианта ввода графа: через интерфейс приложения или загрузку файла с расширением .txt. Это обеспечивает гибкость и удобство использования приложения различными категориями пользователей.

Интерфейс приложения разработан таким образом, чтобы обеспечивать пошаговую визуализацию алгоритма, что позволяет пользователю глубже понять процесс работы алгоритма Краскала. В ходе тестирования были

обнаружены и устранены некоторые неточности в работе приложения, что подтверждает стремление команды к качеству и точности.

В целом, полученный результат соответствует поставленным целям учебной практики и демонстрирует успешное применение знаний в области программирования на Java и алгоритмов поиска остовных деревьев. Команда продемонстрировала способность к teamwork, аналитическому мышлению и решению проблем, что является ценным опытом для будущей карьеры в IT.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Metanit. URL: https://metanit.com/java/javafx

Stackoverflow. URL: https://stackoverflow.com/

Wikipedia. URL: https://ru.wikipedia.org/wiki/Шаблон_проектирования

Habr. URL: https://habr.com/ru/articles/474982/

Рыбин С. В. Дискретная математика и информатика : учебник для вузов / С. В. Рыбин. — СанктПетербург : Лань, 2022. — 748 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: KruscalApplication.java

```
package com.kruskal.gui;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;


import java.io.IOException;

import com.kruskal.controlstructures.Mediator;

public class KruskalApplication extends Application {
    private Mediator mediator;
    private ActionController actionController;
    private ShapeController shapeController;
    @Override
    public void start(Stage stage) throws IOException
    {
        FXMLLoader fxmlLoader = new
FXMLLoader(KruskalApplication.class.getResource("application-view.fxml"));
        AnchorPane root = fxmlLoader.load();
        actionController = fxmlLoader.getController();
        actionController.setStage(stage);
        shapeController = new ShapeController((Pane)root.getChildren().get(1));
        connectMediator();
        Image icon = new
Image(getClass().getResourceAsStream("/icons/graph_icon.png"));
        Scene scene = new Scene(root);
        stage.getIcons().add(icon);
        stage.setTitle("Kruskal's algorithm");
        stage.setScene(scene);
        stage.show();
    }

    private void connectMediator() {
        mediator = new Mediator();
        mediator.setActionController(actionController);
        mediator.setShapeController(shapeController);
        actionController.setMediator(mediator);
    }
    public static void main(String[] args) {
        launch();
    }
}
```

Файл: ActionMessege.java

```
package com.kruskal.gui;


public class ActionMessage {
```

```
    private State state;
    private double x;
    private double y;
    private int objectId;
    private int secondObjectId;
    private int weight;

    public String getFileName() {
        return fileName;
    }

    private String fileName;

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public State getState() {
        return state;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public int getObjectId() {
        return objectId;
    }

    public int getSecondObjectId() {
        return secondObjectId;
    }

    public int getWeight() {
        return weight;
    }

    public ActionMessage(State state, double x, double y, int objectId, int
secondObjectId, int weight, String fileName) {
        this.state = state;
        this.x = x;
        this.y = y;
        this.objectId = objectId;
        this.secondObjectId = secondObjectId;
        this.weight = weight;
        this.fileName = fileName;
    }
    public ActionMessage(State state, double x, double y) {
        this(state, x, y, -1, -1, -1, null);
    }

    public ActionMessage(State state) {
        this(state, -1, -1, -1, -1, -1, null);
    }

    public ActionMessage(State state, int objectId, int secondObjectId, int
weight) {
        this(state, -1, -1, objectId, secondObjectId, weight, null);
    }
```

```
    public ActionMessage(State state, int objectId) {
        this(state, -1, -1, objectId, -1, -1, null);
    }

    public ActionMessage(State state, double x, double y, int objectId) {
        this(state, x, y, objectId, -1, -1, null);
    }

    public ActionMessage(State state, String fileName) {
        this(state, -1, -1, -1, -1, -1, fileName);
    }
    public ActionMessage(State state, double x, double y,String fileName) {
        this(state, x, y, -1, -1, -1, fileName);
    }
}
```

Файл: ActionConroller.java

```
package com.kruskal.gui;

import com.kruskal.controlstructures.Mediator;
import com.kruskal.fileparser.FileFormatException;
import com.kruskal.graph.GraphConnectednessExeption;
import com.kruskal.kruskalalgorithm.Kruskal;
import com.kruskal.kruskalalgorithm.StepMessage;
import com.kruskal.shapeview.EdgeView;
import com.kruskal.shapeview.NodeView;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class ActionController {
    @FXML
    private Button addEdgeButton;
    @FXML
    private Button addNodeButton;
    @FXML
    private Label currentState;
    @FXML
    private Button nextStepButton;
    @FXML
    private Button previousStepButton;
    @FXML
    private Button removeEdgeButton;
    @FXML
    private Button removeNodeButton;
    @FXML
```

```
    private Button replaceNodeButton;
    @FXML
    private Button runAlgorithmButton;
    @FXML
    private Button runAlgButton;
    @FXML
    private Button saveGraphButton;
    @FXML
    private Button uploadGraphButton;
    @FXML
    private TextArea messageSender;
    @FXML
    private Pane mainPane;
    @FXML
    private Label treeWeightInfo;
    private Mediator mediator;
    private NodeView startNode;
    private NodeView endNode;
    private Stage stage;
    private Kruskal algoritm;

    public void setStage(Stage stage) {
        this.stage = stage;
    }

    @FXML
    protected void onRunAlgorithmButtonClick() {
        try {
            mediator.sendRequest(new ActionMessage(State.RUNALGORITHM));
            currentState.setText("Run Algorithm");
            currentState.setOpacity(1);
            nextStepButton.setDisable(false);
            runAlgButton.setDisable(false);
            setDisability(true);
        } catch (GraphConnectednessExeption exception) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setHeaderText(exception.getMessage());
            alert.showAndWait();
        }
    }
    public void onRunAlgButtonClick() {
        try {
            mediator.sendRequest(new ActionMessage(State.PREVIOUSSTEP));
            // Обновляем UI соответствующим образом
            currentState.setText("Algorithm Finished");
            currentState.setOpacity(1);
            nextStepButton.setDisable(true); // Предполагается, что кнопка
следующего шага должна быть отключена после завершения алгоритма
            runAlgButton.setDisable(true); // Предполагается, что кнопка запуска
алгоритма должна быть отключена после его завершения
            previousStepButton.setDisable(false);
            setDisability(true); // Предполагается, что это метод для установки
всех кнопок в состояние "неактивно"
        } catch (GraphConnectednessExeption exception) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setHeaderText(exception.getMessage());
            alert.showAndWait();
            createErrorAlertMessage(exception.getMessage());
        }
    }
    @FXML
    protected void onUploadGraphButtonClick() {
        currentState.setText("Current state");
```

```
        currentState.setOpacity(0.5d);
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open File");
        File file = fileChooser.showOpenDialog(stage);
        if (file != null) {
            try {
                mediator.sendRequest(new ActionMessage(State.UPLOADGRAPH,
mainPane.getWidth(), mainPane.getHeight(), file.getAbsolutePath()));
                messageSender.appendText("Graph has been uploaded\n");
            } catch (FileFormatException exception) {
                createErrorAlertMessage(exception.toString());
            } catch (Exception exception) {
                createErrorAlertMessage(exception.getMessage());
            }
        } else {
            createErrorAlertMessage("Не удалось открыть файл");
        }
    }

    @FXML
    protected void onSaveGraphButtonClicked() {
        currentState.setText("Current state");
        currentState.setOpacity(0.5d);
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open File");
        File file = fileChooser.showSaveDialog(stage);
        if (file != null) {
            try {
                mediator.sendRequest(new ActionMessage(State.SAVEGRAPH,
file.getAbsolutePath()));
                messageSender.appendText("Graph has been saved\n");
            } catch (FileFormatException exception) {
                createErrorAlertMessage(exception.toString());
            }
        } else {
            createErrorAlertMessage("Не удалось открыть файл");
        }
    }

    @FXML
    protected void onNextStepButtonClick() {
        if (previousStepButton.isDisable()) {
            previousStepButton.setDisable(false);
            runAlgButton.setDisable(true);
        }
        try {
            mediator.sendRequest(new ActionMessage(State.NEXTSTEP));
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }

    }

    @FXML
    protected void onPreviousStepButtonClick() {
        if (nextStepButton.isDisable()) {
            nextStepButton.setDisable(false);
            runAlgButton.setDisable(true);
        }
        try {
            mediator.sendRequest(new ActionMessage(State.PREVIOUSSTEP));
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
```

```
        }
    }

    @FXML
    protected void onRestartButtonClick() {
        setDisability(false);
        treeWeightInfo.setText("0");
        nextStepButton.setDisable(true);
        previousStepButton.setDisable(true);
        messageSender.clear();
        currentState.setText("Current state");
        currentState.setOpacity(0.5d);
        mainPane.setOnMouseClicked(event -> {});
        mediator.sendRequest(new ActionMessage(State.RESTART));
    }

    private void setDisability(boolean disability) {
        addEdgeButton.setDisable(disability);
        addNodeButton.setDisable(disability);
        removeEdgeButton.setDisable(disability);
        removeNodeButton.setDisable(disability);
        saveGraphButton.setDisable(disability);
        uploadGraphButton.setDisable(disability);
        runAlgorithmButton.setDisable(disability);
        replaceNodeButton.setDisable(disability);
    }

    @FXML
    protected void onAddNodeButtonClick() {
        currentState.setText("Add Node");
        currentState.setOpacity(1);
        try {
            mainPane.setOnMouseClicked(event -> {
                if (mainPane.contains(event.getX() + 40, event.getY())) {
                    mediator.sendRequest(new ActionMessage(State.ADDNODE,
event.getX(), event.getY()));
                }
            });
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onAddEdgeButtonClick() {
        currentState.setText("Add Edge");
        currentState.setOpacity(1);
        try {
            mainPane.setOnMouseClicked(event -> {
                if (startNode == null) {
                    startNode = findNode(event.getX(), event.getY());
                    startNode.setStroke(Color.rgb(255,0,0));
                } else {
                    endNode = findNode(event.getX(), event.getY());
                    if (endNode != null && !endNode.equals(startNode)) {
                        String inputWeight = runAlert();
                        int weight = !inputWeight.equals("") ?
Integer.parseInt(inputWeight) : 1;
                        mediator.sendRequest(new ActionMessage(State.ADDEDGE,
startNode.getIdNumber(), endNode.getIdNumber(), weight));
                        startNode.setStroke(Color.rgb(157,0,253));
                        startNode = null;
                        endNode.setStroke(Color.rgb(157,0,253));
```

```
                        endNode = null;
                    }
                }
            });
        }
        catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onRemoveNodeButtonClick() {
        currentState.setText("Remove Node");
        currentState.setOpacity(1);
        try {
            mainPane.setOnMouseClicked(event -> {
                NodeView removingNode = findNode(event.getX(), event.getY());
                if (removingNode != null) {
                    mediator.sendRequest(new ActionMessage(State.REMOVENODE,
removingNode.getIdNumber()));
                }
            });
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onRemoveEdgeButtonClicked() {
        currentState.setText("Remove Edge");
        currentState.setOpacity(1);
        try {
            mainPane.setOnMouseClicked(event -> {
                for (int i = 0; i < mainPane.getChildren().size(); ++i) {
                    if (mainPane.getChildren().get(i) instanceof Line edge) {
                        if (edge.contains(event.getX(), event.getY())) {
                            mediator.sendRequest(new
ActionMessage(State.REMOVEEDGE, ((EdgeView) edge).getIdNumber()));
                        }
                    }
                }
            });
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
        }
    }

    @FXML
    protected void onReplaceNodeButtonClicked() {
        currentState.setText("Replace Node");
        currentState.setOpacity(1);
        startNode = null;
        try {
            mainPane.setOnMouseClicked(event -> {
                if (event.getButton() == MouseButton.PRIMARY) {
                    if (startNode == null) {
                        startNode = findNode(event.getX(), event.getY());
                        startNode.setStroke(Color.rgb(255,0,0));
                    } else {
                        mediator.sendRequest(new
ActionMessage(State.REPLACENODE, event.getX(), event.getY(),
startNode.getIdNumber()));
                        startNode.setStroke(Color.rgb(157,0,253));
```

```
                        startNode = null;
                    }
                } else if (event.getButton() == MouseButton.SECONDARY) {
                    startNode = null;
                }
            });
        } catch (Exception exception) {
            createErrorAlertMessage(exception.getMessage());
            startNode.setStroke(Color.rgb(157,0,253));
        }
    }

    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    private NodeView findNode(double x, double y) {
        for (int i = 0; i < mainPane.getChildren().size(); ++i) {
            if (mainPane.getChildren().get(i) instanceof Circle node) {
                if (node.contains(x, y)) return (NodeView) node;
            }
        }
        return null;
    }

    private String runAlert() {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Text field");
        alert.setHeaderText("Введите вес ребра");
        TextField inputField = new TextField();
        inputField.setPrefWidth(Integer.toString(Integer.MAX_VALUE).length());
        inputField.textProperty().addListener((observable, oldValue, newValue) -
> {
            if (newValue.startsWith("0") || !newValue.matches("\\d*") ||
newValue.equals("") ||
                    Integer.toString(Integer.MAX_VALUE).length() <
newValue.length() ||
                    Double.parseDouble(newValue) > Integer.MAX_VALUE) {
                inputField.setStyle("-fx-border-color: red");
                Button okButton = (Button)
alert.getDialogPane().lookupButton(ButtonType.OK);
                okButton.setDisable(true);
            } else {
                inputField.setStyle("-fx-border-color: blue");
                Button okButton = (Button)
alert.getDialogPane().lookupButton(ButtonType.OK);
                okButton.setDisable(false);
            }
        });
        VBox vBox = new VBox();
        vBox.getChildren().addAll(inputField);
        alert.getDialogPane().setContent(vBox);
        alert.showAndWait();
        return inputField.getText();
    }

    public void printMessage(StepMessage message) {
        switch (message.getType()) {
            case EDGEADDED -> messageSender.appendText("Edge between " +
message.getFirstNodeId()
                    + " and " + message.getSecondNodeId() + " were added\n");
            case EDGEDECLINED -> messageSender.appendText("Edge between " +
message.getFirstNodeId()
```

```
                    + " and " + message.getSecondNodeId() + " were declined\n");
            case ALGORITHMENDED -> {
                messageSender.appendText("Algorithm ended\n");
                nextStepButton.setDisable(true);
            }
        }
    }

    public void deleteLastMessage() {
        List<String> sentences = new
ArrayList<>(List.of(messageSender.getText().split("\n")));
        if (sentences.size() > 0 && sentences.contains("Algorithm ended")) {
            sentences.remove(sentences.size() - 1);
            if (sentences.size() > 0) {
                sentences.set(sentences.size() - 1, "");
            }
        } else {
            sentences.set(sentences.size() - 1, "");
        }
        if (!sentences.isEmpty()) {
            messageSender.setText(String.join("\n", sentences));
        } else {
            messageSender.setText("");
        }
    }

    public void blockPreviousStepButton() {
        previousStepButton.setDisable(true);
    }

    public void printTreeWeight(int treeWeight) {
        treeWeightInfo.setText(Integer.toString(treeWeight));
    }

    private void createErrorAlertMessage(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(message);
        alert.showAndWait();
        mediator.sendRequest(new ActionMessage(State.RESTART));
    }


}
```

Файл ShapeController.java

```
package com.kruskal.gui;

import com.kruskal.kruskalalgorithm.StepMessage;
import com.kruskal.shapeview.ShapeContainer;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;


public class ShapeController {
    private final ShapeContainer shapeContainer;


    public ShapeController(Pane pane) {
        this.shapeContainer = new ShapeContainer(pane);
    }
```

```
    public void clear() {
        shapeContainer.clear();
    }


    public void removeEdge(int edgeId) {
        shapeContainer.removeEdge(edgeId);
    }

    public void removeNode(int nodeId) {
        shapeContainer.removeNode(nodeId);
    }

    public void addEdge(int startNodeId, int endNodeId, int weight, int edgeId)
{
        shapeContainer.createEdge(startNodeId, endNodeId, weight, edgeId);
    }

    public void addNode(double x, double y, int nodeId) {
        shapeContainer.createNode(x, y,
                30, Color.rgb(157, 0, 255), nodeId);
    }

    public void replaceNode(double x, double y, int objectId) {
        shapeContainer.replaceNode(x, y, objectId);
    }
    public void paintEdge(StepMessage stepMessage) {
        switch (stepMessage.getType()) {
            case EDGEADDED -> shapeContainer.colorEdge(stepMessage.getEdgeId(),
Color.LIME);
            case EDGEDECLINED ->
shapeContainer.colorEdge(stepMessage.getEdgeId(), Color.RED);
        }
    }

    public void paintEdgeDefault(StepMessage stepMessage) {
        shapeContainer.colorEdge(stepMessage.getEdgeId(), Color.rgb(0, 0, 0));
    }
}
```

Файл: State.java

```
package com.kruskal.gui;

public enum State {
    ADDNODE,
    ADDEDGE,
    REMOVENODE,
    REMOVEEDGE,
    REPLACENODE,
    SAVEGRAPH,
    UPLOADGRAPH,
    RUNALGORITHM,
    NEXTSTEP,
    RESTART,
    PREVIOUSSTEP,
    RUNALG

}
```

Файл: Kruskal.java

```
package com.kruskal.kruskalalgorithm;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import com.kruskal.graph.EdgeData;
import com.kruskal.graph.Graph;

public class Kruskal {
    private final List<StepMessage> steps;
    private int currentStep;
    private final List<EdgeData> edges;
    private final List<Set<Integer>> connectedNodes;
    private int edgeNumber;
    private int edgesWeightSum;

    public Kruskal(Graph graph){
        steps = new ArrayList<StepMessage>();
        currentStep = 0;
        Set<Integer> nodes = graph.getNodesId();
        edges = graph.getEdgesData();

        Comparator<EdgeData> comparator = new Comparator<EdgeData>() {
            public int compare(EdgeData first, EdgeData second){
                return
Integer.valueOf(first.getWeight()).compareTo(Integer.valueOf(second.getWeight())
);
            }
        };
        edges.sort(comparator);

        connectedNodes = new ArrayList<Set<Integer>>();
        for (Integer node: nodes) {
            Set<Integer> newSet = new HashSet<Integer>();
            newSet.add(node);
            connectedNodes.add(newSet);
        }

        edgeNumber = 0;
        edgesWeightSum = 0;
    }

    public StepMessage makeStep(){
        if(currentStep < edgeNumber){
            currentStep++;
            edgesWeightSum += steps.get(currentStep - 1).getWeightShift();
            return steps.get(currentStep - 1);
        }

        if(edgeNumber == edges.size()){
            return new StepMessage(StepMessageType.ALGORITHMENDED);
        }

        if(connectedNodes.size() == 1){
            return new StepMessage(StepMessageType.ALGORITHMENDED);
        }

        EdgeData currentEdge = edges.get(edgeNumber);
        edgeNumber++;

        Set<Integer> firstNodeSet = null;
```

```
        Set<Integer> secondNodeSet = null;

        for (Set<Integer> set : connectedNodes) {
            if(firstNodeSet == null){
                if(set.contains(currentEdge.getFirstNodeId())){
                    firstNodeSet = set;
                }
            }

            if(secondNodeSet == null){
                if(set.contains(currentEdge.getSecondNodeId())){
                    secondNodeSet = set;
                }
            }
        }

        if(firstNodeSet.equals(secondNodeSet)){
            steps.add(new StepMessage(currentEdge, StepMessageType.EDGEDECLINED,
0));
            currentStep++;
            return steps.get(currentStep - 1);
        }

        firstNodeSet.addAll(secondNodeSet);
        connectedNodes.remove(secondNodeSet);
        edgesWeightSum += currentEdge.getWeight();

        steps.add(new StepMessage(currentEdge, StepMessageType.EDGEADDED,
currentEdge.getWeight()));
        currentStep++;
        return steps.get(currentStep - 1);
    }

    public StepMessage stepBack(){
        if(currentStep == 0){
            return null;
        }

        currentStep--;
        edgesWeightSum -= steps.get(currentStep).getWeightShift();
        return steps.get(currentStep);
    }

    public int getCurrentTreeWeight(){
        return edgesWeightSum;
    }


    public boolean isFinish() {
        // Алгоритм завершен, если все ребра обработаны и нет новых ребер для
добавления
        return connectedNodes.size() != 1 && !edges.isEmpty();
    }
}
```

Файл: StepMessage.java

```
package com.kruskal.kruskalalgorithm;
import com.kruskal.graph.EdgeData;

public class StepMessage {
    private final StepMessageType messageType;
```

```
    private final int firstNodeId;
    private final int secondNodeId;
    private final int edgeId;
    private final int weightShift;

    public StepMessage(EdgeData edgeData, StepMessageType messageType, int
weightShift){
        this.weightShift = weightShift;
        this.messageType = messageType;
        firstNodeId = edgeData.getFirstNodeId();
        secondNodeId = edgeData.getSecondNodeId();
        edgeId = edgeData.getId();
    }

    public StepMessage(StepMessageType messageType){
        this.messageType = messageType;
        firstNodeId = -1;
        secondNodeId = -1;
        edgeId = -1;
        weightShift = -1;
    }

    public StepMessageType getType(){
        return messageType;
    }

    public int getFirstNodeId(){
        return firstNodeId;
    }

    public int getEdgeId() {
        return edgeId;
    }

    public int getSecondNodeId() {
        return secondNodeId;
    }

    int getWeightShift(){
        return weightShift;
    }
}
```

Файл: StepMessageType.java

```
package com.kruskal.kruskalalgorithm;

public enum StepMessageType {
    EDGEADDED,
    EDGEDECLINED,
    ALGORITHMENDED
}
```

Файл: EdgeView.java

```
package com.kruskal.shapeview;

import javafx.scene.shape.Line;
import javafx.scene.text.Text;
```

```
import javafx.scene.text.TextAlignment;
import javafx.scene.text.TextFlow;
import javafx.util.Pair;

public class EdgeView extends Line {
    private int idNumber;
    private final int weight;
    private Text weightText;
    private final TextFlow textFlow = new TextFlow();

    public TextFlow getTextFlow() {
        return textFlow;
    }

    private Pair<NodeView, NodeView> adjacentNodes;

    public int getIdNumber() {
        return idNumber;
    }

    public void setIdNumber(int idNumber) {
        this.idNumber = idNumber;
    }

    public Text getWeightText() {
        return weightText;
    }

    public EdgeView(double v, double v1, double v2, double v3, int idNumber, int
weight) {
        super(v, v1, v2, v3);
        this.idNumber = idNumber;
        this.weight = weight;
        this.weightText = new Text(Integer.toString(weight));
    }

    public EdgeView(int idNumber, int weight) {
        super();
        this.idNumber = idNumber;
        this.weight = weight;
        this.weightText = new Text(Integer.toString(weight));
        this.weightText.setStyle("-fx-fill: #000000; -fx-font-weight: bold");
        this.textFlow.setStyle("-fx-background-color: #F2BFF7FF;");
        this.textFlow.setPrefWidth(25);
        this.textFlow.setPrefHeight(25);
        this.textFlow.setTextAlignment(TextAlignment.CENTER);
    }
    public void setAdjacentNodes(NodeView startNode, NodeView endNode) {
        adjacentNodes = new Pair<>(startNode, endNode);
    }

    public NodeView getStartNode() {
        return adjacentNodes.getKey();
    }
    public NodeView getEndNode() {
        return adjacentNodes.getValue();
    }
}
```

Файл: NodeView.java

```
package com.kruskal.shapeview;

import javafx.scene.paint.Paint;
import javafx.scene.shape.Circle;
import javafx.scene.text.Text;
import javafx.util.Pair;

import java.util.ArrayList;
import java.util.List;

public class NodeView extends Circle {
    private int idNumber; // Удален модификатор final
    private final Text text;
    private final List<EdgeView> incidentEdgeIdList = new ArrayList<>();

    public int getIdNumber() {
        return idNumber;
    }

    public NodeView(double x, double y, double radius, Paint paint, int
idNumber) {
        super(x, y, radius, paint);
        this.idNumber = idNumber;
        this.text = new Text(x-7, y+7, Integer.toString(idNumber));
        this.text.setStyle("-fx-font-size: 25");
    }

    public Text getText() {
        return text;
    }

    public void addIncidentEdgeId(EdgeView edge) {
        incidentEdgeIdList.add(edge);
    }

    public boolean hasIncidentEdge(EdgeView edge) {
        return incidentEdgeIdList.contains(edge);
    }

    public void removeEdge(EdgeView edge) {
        incidentEdgeIdList.remove(edge);
    }

    public boolean isStartVertex(EdgeView edge) {
        return edge.getStartNode().equals(this);
    }

    public void setIdNumber(int idNumber) {
        this.idNumber = idNumber;
        this.text.setText(String.valueOf(idNumber)); // Обновляем текстовое
представление идентификатора
    }
}
```

Файл: ShapeContainer.java

```
package com.kruskal.shapeview;

import com.kruskal.graph.Edge;
import com.kruskal.graph.Node;
import javafx.scene.layout.Pane;
```

```
import javafx.scene.paint.Color;

import java.util.ArrayList;
import java.util.List;

public class ShapeContainer {
    private final List<NodeView> nodeViewList = new ArrayList<>();
    private final List<EdgeView> edgeViewList = new ArrayList<>();
    private final Pane pane;


    public ShapeContainer(Pane pane) {
        this.pane = pane;
    }

    public void createNode(double xCoordinate, double yCoordinate, double
radius, Color color, int nodeId) {
        NodeView node = new NodeView(xCoordinate, yCoordinate, radius, color,
nodeId);
        nodeViewList.add(node);
        pane.getChildren().addAll(node, node.getText());
        redrawGraph(); // Перерисовываем граф после добавления узла

    }

    public void createEdge(int startNodeId, int endNodeId, int weight, int
edgeId) {
        EdgeView edgeView = new EdgeView(edgeId, weight);
        NodeView startNode = null, endNode = null;
        for (NodeView node : nodeViewList) {
            if (node.getIdNumber() == startNodeId) {
                startNode = node;
                edgeView.setStartX(node.getCenterX());
                edgeView.setStartY(node.getCenterY());
                node.addIncidentEdgeId(edgeView);
            } else if (node.getIdNumber() == endNodeId) {
                endNode = node;
                edgeView.setEndX(node.getCenterX());
                edgeView.setEndY(node.getCenterY());
                node.addIncidentEdgeId(edgeView);
            }
        }
        edgeView.setAdjacentNodes(startNode, endNode);
        edgeView.setStroke(Color.BLACK);
        edgeView.setStrokeWidth(5d);
        edgeView.getTextFlow().setLayoutX((edgeView.getEndX() +
edgeView.getStartX()) / 2 - 10);
        edgeView.getTextFlow().setLayoutY((edgeView.getEndY() +
edgeView.getStartY()) / 2 - 10);
        edgeView.getWeightText().setX((edgeView.getEndX() +
edgeView.getStartX()) / 2-2.5); // Позиция X для текста веса
        edgeView.getWeightText().setY((edgeView.getEndY() +
edgeView.getStartY()) / 2+2); // Позиция Y для текста веса

        // Добавление ребра, текста веса и текстового потока на панель
        pane.getChildren().addAll(edgeView, edgeView.getTextFlow(),
edgeView.getWeightText());

        edgeViewList.add(edgeView);
        redrawGraph(); // Перерисовываем граф после добавления ребра
    }

    public void removeNode(int nodeId) {
```

```
    for (NodeView node : nodeViewList) {
        if (nodeId == node.getIdNumber()) {
            List<EdgeView> needToBeDeletedEdge = new ArrayList<>();
            for (EdgeView edge : edgeViewList) {
                if (node.hasIncidentEdge(edge)) {
                    if (!node.equals(edge.getStartNode())) {
                        edge.getStartNode().removeEdge(edge);
                    } else {
                        edge.getEndNode().removeEdge(edge);
                    }
                    pane.getChildren().remove(edge.getTextFlow());
                    pane.getChildren().remove(edge);
                    needToBeDeletedEdge.add(edge);
                    node.removeEdge(edge);
                    // Удаление текста числа ребра
                    pane.getChildren().remove(edge.getWeightText());
                }
            }
            for (EdgeView edge : needToBeDeletedEdge) {
                if (edgeViewList.contains(edge)) {
                    edgeViewList.remove(edge);
                }
            }
            nodeViewList.remove(node);
            pane.getChildren().remove(node.getText());
            pane.getChildren().remove(node);
            break;
        }
    }
}


public void removeEdge(int edgeId) {
    for (EdgeView edge : edgeViewList) {
        if (edge.getIdNumber() == edgeId) {
            edge.getStartNode().removeEdge(edge);
            edge.getEndNode().removeEdge(edge);
            edgeViewList.remove(edge);
            pane.getChildren().remove(edge.getTextFlow());
            pane.getChildren().remove(edge);
            // Удаление текста числа ребра
            pane.getChildren().remove(edge.getWeightText());
            break;
        }
    }
}
public void redrawGraph() {
    // Очистка панели
    pane.getChildren().clear();

    // Добавляем все рёбра
    for (EdgeView edge : edgeViewList) {
        pane.getChildren().addAll(edge, edge.getTextFlow());
    }

    // Добавляем все узлы
    for (NodeView node : nodeViewList) {
        pane.getChildren().addAll(node, node.getText());
    }

    // Добавляем веса рёбер после отрисовки всех рёбер и узлов
    for (EdgeView edge : edgeViewList) {
        pane.getChildren().add(edge.getWeightText());
```

```
        }
    }


    public void clear() {
        nodeViewList.clear();
        edgeViewList.clear();
        pane.getChildren().clear();
    }

    public void replaceNode(double x, double y, int objectId) {
        for (NodeView node : nodeViewList) {
            if (node.getIdNumber() == objectId) {
                node.setCenterX(x);
                node.setCenterY(y);
                node.getText().setX(x);
                node.getText().setY(y);
                for (EdgeView edge : edgeViewList) {
                    if (node.hasIncidentEdge(edge) && node.isStartVertex(edge))
{
                        edge.setStartX(x);
                        edge.setStartY(y);
                        edge.getTextFlow().setLayoutX((edge.getEndX() +
edge.getStartX()) / 2 - 10);
                        edge.getTextFlow().setLayoutY((edge.getEndY() +
edge.getStartY()) / 2 - 10);
                        edge.getWeightText().setX((edge.getEndX() +
edge.getStartX()) / 2); // Обновление X для текста веса
                        edge.getWeightText().setY((edge.getEndY() +
edge.getStartY()) / 2); // Обновление Y для текста веса
                    } else if (node.hasIncidentEdge(edge)) {
                        edge.setEndX(x);
                        edge.setEndY(y);
                        edge.getTextFlow().setLayoutX((edge.getEndX() +
edge.getStartX()) / 2 - 10);
                        edge.getTextFlow().setLayoutY((edge.getEndY() +
edge.getStartY()) / 2 - 10);
                        edge.getWeightText().setX((edge.getEndX() +
edge.getStartX()) / 2); // Обновление X для текста веса
                        edge.getWeightText().setY((edge.getEndY() +
edge.getStartY()) / 2); // Обновление Y для текста веса
                    }
                }
                break;
            }
        }
    }

    public void colorEdge(int edgeId, Color color) {
        for (EdgeView edge : edgeViewList) {
            if (edge.getIdNumber() == edgeId) {
                edge.setStroke(color);
            }
        }
    }
}
```

Файл: Edge.java

```
package com.kruskal.graph;
```

```
public class Edge {
    private final Node firstNode;
    private final Node secondNode;
    private final int weight;
    private final int id;

    Edge(Node first, Node second, int weight, int id){
        this.firstNode = first;
        this.secondNode = second;
        this.weight = weight;
        this.id = id;
    }

    @Override
    public boolean equals(Object obj){
        if(this == obj){
            return true;
        }
        if(obj instanceof Edge){
            if(this.firstNode.equals(((Edge)obj).firstNode)){
                if(this.secondNode.equals(((Edge)obj).secondNode)){
                    return true;
                }
            }

            if(this.firstNode.equals(((Edge)obj).secondNode)){
                if(this.secondNode.equals(((Edge)obj).firstNode)){
                    return true;
                }
            }
        }
        return false;
    }

    int getWeight() {
        return weight;
    }

    Node getFirstNode() {
        return firstNode;
    }

    Node getSecondNode() {
        return secondNode;
    }

    int getId() {
        return id;
    }

    Node getAdjacentNode(Node start){
        if(firstNode.equals(start)){
            return secondNode;
        }
        return firstNode;
    }

    @Override
    public String toString(){
        return "(EdgeId:"+ id +" FirstNodeId:"+ firstNode.getId()+"
SecondNodeId:"+secondNode.getId() +" Weight"+ weight + ")";
    }
}
```

Файл: EdgeData.java

```
package com.kruskal.graph;

public class EdgeData {
    private final int id;
    private final int weight;
    private final int firstNodeId;
    private final int secondNodeId;

    EdgeData(Edge edge){
        this.id = edge.getId();
        this.weight = edge.getWeight();
        this.firstNodeId = edge.getFirstNode().getId();
        this.secondNodeId = edge.getSecondNode().getId();
    }

    public int getId(){
        return id;
    }

    public int getWeight() {
        return weight;
    }

    public int getFirstNodeId() {
        return firstNodeId;
    }

    public int getSecondNodeId() {
        return secondNodeId;
    }
}
```

Файл: Graph.java

```
package com.kruskal.graph;

import java.util.List;
import java.util.Set;
import java.util.ArrayList;
import java.util.HashSet;

public class Graph {
    private final List<Node> nodes;

    Graph(){
        nodes = new ArrayList<Node>();
    };

    private Node findNode(int nodeId){
        for(Node currentNode: nodes){
            if(currentNode.getId() == nodeId){
                return currentNode;
            }
        }
        return null;
    }

    boolean deleteNode(int nodeId){
```

```
    Node currentNode = findNode(nodeId);
    if(currentNode != null){

        List<Edge> edges = currentNode.getAdjacentEdges();
        while (!edges.isEmpty()) {
            Edge target = edges.get(0);
            currentNode.deleteEdge(target.getId());
        }

        nodes.remove(currentNode);
        return true;
    }
    return false;
}

boolean addNode(int newNodeId){
    Node newNode = new Node(newNodeId);
    if(findNode(newNode.getId()) == null){
        nodes.add(newNode);
        return true;
    }
    return false;
}

boolean addEdge(int firstNodeId, int secondNodeId, int weight, int edgeId){
    List<Edge> allEdges = getEdgesList();
    for(Edge edge: allEdges){
        if(edge.getId() == edgeId){
            return false;
        }
    }

    Node firstNode = findNode(firstNodeId);
    if(firstNode == null){
        return false;
    }

    Node secondNode = findNode(secondNodeId);
    if(secondNode == null){
        return false;
    }

    Edge newEdge = new Edge(firstNode, secondNode, weight, edgeId);

    if(newEdge.getFirstNode().getAdjacentEdges().contains(newEdge)){
        return false;
    }
    newEdge.getFirstNode().addEdge(newEdge);
    return true;
}

boolean deleteEdge(int edgeId){
    List<Edge> allEdges = getEdgesList();
    for(Edge edge: allEdges){
        if(edge.getId() == edgeId){
            edge.getFirstNode().deleteEdge(edgeId);
            return true;
        }
    }
    return false;
}

private List<Edge> getEdgesList(){
```

```
    List<Edge> allEdges = new ArrayList<Edge>();
    Set<Node> viewedNodes = new HashSet<Node>();
    List<Node> nodesToView = new ArrayList<Node>();

    if(nodes.isEmpty()){
        return allEdges;
    }

    nodesToView.add(nodes.get(0));
    while(!nodesToView.isEmpty()){
        Node currentNode = nodesToView.get(0);
        nodesToView.remove(0);

        if(viewedNodes.contains(currentNode)){
            continue;
        }

        viewedNodes.add(currentNode);

        for (Edge adjacent: currentNode.getAdjacentEdges()){
            if(!allEdges.contains(adjacent)){
                allEdges.add(adjacent);
            }

            nodesToView.add(adjacent.getAdjacentNode(currentNode));

        }
    }

    return allEdges;
}

public void printGraph(){
    for (Node node: nodes) {
        System.out.println(node);
    }
}

public boolean isValid(){
    Set<Node> viewedNodes = new HashSet<Node>();
    List<Node> nodesToView = new ArrayList<Node>();

    if(nodes.isEmpty()){
        return true;
    }

    nodesToView.add(nodes.get(0));
    while(!nodesToView.isEmpty()){
        Node currentNode = nodesToView.get(0);
        nodesToView.remove(0);

        if(viewedNodes.contains(currentNode)){
            continue;
        }

        viewedNodes.add(currentNode);

        for (Edge adjacent: currentNode.getAdjacentEdges()){
            nodesToView.add(adjacent.getAdjacentNode(currentNode));
        }
    }

    for (Node node : nodes){
```

```
            if(!viewedNodes.contains(node)){
                return false;
            }
        }

        return true;
    }

    public Set<Integer> getNodesId(){
        Set<Integer> set = new HashSet<>();
        for (Node node : nodes) {
            set.add(node.getId());
        }
        return set;
    }

    public List<EdgeData> getEdgesData(){
        List<EdgeData> data = new ArrayList<EdgeData>();
        for (Edge edge : getEdgesList()) {
            data.add(new EdgeData(edge));
        }
        return data;
    }
}
```

Файл: GraphBuilder.java

```
package com.kruskal.graph;

import java.util.List;


public class GraphBuilder {

    private Graph graph;
    public GraphBuilder(){
            graph = new Graph();
    };

    public boolean deleteNode(int nodeId){
        return graph.deleteNode(nodeId);
    }

    public boolean deleteEdge(int edgeId){
        return graph.deleteEdge(edgeId);
    }

    public boolean addNode(int newNodeId){
        return graph.addNode(newNodeId);
    }
    public boolean addEdge(int firstNodeId, int secondNodeId, int weight, int
edgeId){
        if(weight > 0){
            return graph.addEdge(firstNodeId, secondNodeId, weight, edgeId);
        }
        return false;
    }

    public boolean isValid(){
        return graph.isValid();
    }
```

```
    public void printGraph(){
        graph.printGraph();
    }

    public List<EdgeData> getEdgesData(){
        return graph.getEdgesData();
    }

    public void resetGraph(){
        graph = new Graph();
    }

    public Graph getGraph() {
        return graph;
    }
}
```

Файл: GraphConnectednessExeption.java

```
package com.kruskal.graph;

public class GraphConnectednessExeption extends RuntimeException{
    public GraphConnectednessExeption(String message){
        super(message);
    }

    @Override
    public String toString() {
        return "FileFormatException{" + getMessage() + "}";
    }
}
```

Файл: Node.java

```
package com.kruskal.graph;

import java.util.ArrayList;
import java.util.List;


public class Node {
    private final int id;
    private final List<Edge> adjacentEdges;
    Node(int id){
        this.id = id;
        adjacentEdges = new ArrayList<Edge>();
    };

    public int getId() {
        return id;
    }

    void addEdge(Edge newEdge){
        adjacentEdges.add(newEdge);
        newEdge.getAdjacentNode(this).adjacentEdges.add(newEdge);
    }

    private Edge findEdge(int edgeId){
        for(Edge currentEdge: adjacentEdges){
```

```
            if(currentEdge.getId() == edgeId){
                return currentEdge;
            }
        }
        return null;
    }

    void deleteEdge(int edgeId){
        Edge currentEdge = findEdge(edgeId);
        if(currentEdge != null){
            currentEdge.getFirstNode().adjacentEdges.remove(currentEdge);
            currentEdge.getSecondNode().adjacentEdges.remove(currentEdge);
        }
    }

    List<Edge> getAdjacentEdges() {
        return adjacentEdges;
    }

    @Override
    public boolean equals(Object obj){
        if(this == obj){
            return true;
        }
        if(obj instanceof Node){
            if (this.id == ((Node)obj).id){
                return true;
            }
        }
        return false;
    }

    @Override
    public String toString(){
        StringBuilder builder = new StringBuilder();
        builder.append("NodeId:" + id + ",  Edges:\n");
        for (Edge edge : adjacentEdges) {
            builder.append("\t" + edge + "\n");
        }
        return builder.toString();
    }
}
```

Файл: FileFormatException.java

```
package com.kruskal.fileparser;
public class FileFormatException extends RuntimeException{

    private final int fileStringNumber;
    public FileFormatException(String message, int fileStringNumber){
        super(message);
        this.fileStringNumber = fileStringNumber;
    }

    @Override
    public String toString() {
        return "FileFormatException{" + getMessage() + " (Строка файла: " +
fileStringNumber + ")}";
    }
}
```

Файл: FileReader.java

```
package com.kruskal.fileparser;

import com.kruskal.controlstructures.Mediator;
import com.kruskal.gui.ActionMessage;
import com.kruskal.gui.State;

import java.net.URLConnection;
import java.nio.file.Paths;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class FileReader {

    private enum ReadSymbol{
        SPACE,
        NUMBER
    }

    public void read(String inputFileName, double screenWidth, double
screenHeight, Mediator mediator){

        Path path = Paths.get(inputFileName);
        Scanner scanner;
        int nodeNumber;
        List<List<Integer>> nodeMatrix = new ArrayList<>();
        String type = URLConnection.guessContentTypeFromName(path.toString());
        if(!type.equals("text/plain")){
            throw new FileFormatException("Неверный тип файла", 0);
        }
        try{
            scanner = new Scanner(path);
        } catch (java.io.IOException e){
            throw new FileFormatException("Не удалось открыть файл для чтения",
0);
        }

        String line = scanner.nextLine();
        Scanner lineScanner = new Scanner(line);
        if(lineScanner.hasNextInt()){
            nodeNumber = lineScanner.nextInt();
            if (lineScanner.hasNext()){
                throw new FileFormatException("Неверно задана строка с размером
графа", 1);
            }
        } else{
            throw new FileFormatException("Неверно задана строка с размером
графа", 1);
        }
        if(nodeNumber > 100){
            throw new FileFormatException("Размер графа превышает 80 вершин",
1);
        }
        nodeReader(nodeNumber, screenWidth, screenHeight, mediator, nodeMatrix);
        scanMatrix(nodeMatrix, scanner, nodeNumber);
        nodeMatrixToGraph(nodeMatrix, mediator);
    }
```

```
    private void scanMatrix(List<List<Integer>> nodeMatrix, Scanner scanner, int
nodeNumber){
        for(int i = 0; i < nodeNumber; i++){
            if(!scanner.hasNextLine()) {
                throw new FileFormatException("Задана матрица меньшего размера,
чем указано в строке с размером", i+2);
            }
            String line = scanner.nextLine();
            Scanner lineScanner = new Scanner(line).useDelimiter("");
            ReadSymbol readSymbol = ReadSymbol.NUMBER;
            int numberCount = 0;
            for(int j = 1; j <= line.length(); j++) {
                if((lineScanner.hasNextInt() && (readSymbol ==
ReadSymbol.SPACE)) ||
                        (!lineScanner.hasNextInt() && (readSymbol ==
ReadSymbol.NUMBER))){
                    throw new FileFormatException("Символ вместо ожидаемого
числа", i+2);
                }
                if(readSymbol == ReadSymbol.NUMBER) {
                    int weight = 0;
                    while (lineScanner.hasNextInt()) {
                        weight = weight * 10 + lineScanner.nextInt();
                        if(weight >= 10){
                            j++;
                        }
                    }

                    if(i == numberCount && weight != 0){
                        throw new FileFormatException("Значение на диалогнали
матрицы отличное от 0", i+2);
                    }

                    nodeMatrix.get(i).add(weight);
                    if(numberCount < i &&
!(nodeMatrix.get(i).get(numberCount).equals(nodeMatrix.get(numberCount).get(i)))
){
                        throw new FileFormatException("Матрица не симметричная",
i+2);
                    }
                    numberCount++;
                } else{
                    String next = lineScanner.next();
                    if(!next.equals(" ")) {
                        throw new FileFormatException("Присутствуют символы
кроме разделительного пробела", i+2);
                    }
                }
                if(readSymbol == ReadSymbol.NUMBER){
                    readSymbol = ReadSymbol.SPACE;
                } else {
                    readSymbol = ReadSymbol.NUMBER;
                }
            }
            if(numberCount != nodeNumber){
                throw new FileFormatException("В строке не то количество чисел,
которое ожидалось", i+2);
            }
        }
        if(scanner.hasNextLine()){
            throw new FileFormatException("В файле больше строк, чем ожидалось",
nodeNumber+2);
        }
```

```
        scanner.close();
    }

    private void nodeMatrixToGraph(List<List<Integer>> nodeMatrix, Mediator
mediator){
        for(int raw = 1; raw <= nodeMatrix.size(); raw++){
            for(int column = 1; column <= nodeMatrix.size(); column++){
                if(nodeMatrix.get(raw-1).get(column-1) != 0){
                    mediator.sendRequest(new ActionMessage(State.ADDEDGE, raw,
column, nodeMatrix.get(raw-1).get(column-1)));
                }
            }
        }
    }

    private void nodeReader(int nodeNumber, double screenWidth, double
screenHeight, Mediator mediator, List<List<Integer>> nodeMatrix){
        double radius = Double.min((screenHeight - 80)/2, (screenWidth - 80)/2);
        double centreY = screenHeight/2;
        double centreX = screenWidth/2;
        int layerNodeNumber = 40;
        if (nodeNumber < layerNodeNumber){
            layerNodeNumber = nodeNumber;
        }
        double degreeStep = Math.toRadians((double) 360/(layerNodeNumber));
        int layerNodeIterator = 0;
        for(int i = 1; i <= nodeNumber; i++){
            mediator.sendRequest(new ActionMessage(State.ADDNODE, centreX -
Math.sin(degreeStep*layerNodeIterator)*radius,
                    centreY - Math.cos(degreeStep*layerNodeIterator)*radius));
            nodeMatrix.add(new ArrayList<>());
            layerNodeIterator++;
            if(layerNodeIterator == layerNodeNumber){
                layerNodeNumber -= 10;
                layerNodeIterator = 0;
                if (nodeNumber - i < layerNodeNumber){
                    layerNodeNumber = nodeNumber - i;
                }
                degreeStep =  Math.toRadians((double) 360/(layerNodeNumber));
                radius = radius - 60;

            }
        }
    }
}
```

Файл: FileWriter.java

```
package com.kruskal.fileparser;

import com.kruskal.graph.EdgeData;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.net.URLConnection;
import java.util.*;

public class FileWriter {

    public void write(String outFileName, List<EdgeData> edgesData, Set<Integer>
nodesId){
```

```
        int nodesNumber = nodesId.size();
        File file;
        String type = URLConnection.guessContentTypeFromName(outFileName);
        if(!type.equals("text/plain")){
            throw new FileFormatException("Неверный тип файла", 0);
        }
        try {
            file = new File(outFileName);
            file.createNewFile();
        }
        catch (Exception e) {
            throw new FileFormatException("Неудалось создать/открыть файл для
записи", 0);
        }
        PrintWriter printWriter;
        try {
            printWriter = new PrintWriter(file);
        } catch (FileNotFoundException e) {
            throw new FileFormatException("Неудалось найти файл для записи", 0);
        }

        printWriter.println(nodesNumber);
        IdConverter idConverter = new IdConverter();
        Iterator<Integer> iterator = nodesId.iterator();
        for(int i = 1; i <= nodesNumber; i++){
            idConverter.addId(iterator.next(), i);
        }
        for(int firstNodeIndex = 1; firstNodeIndex <= nodesNumber;
firstNodeIndex++){
            for(int secondNodeIndex = 1; secondNodeIndex <= nodesNumber;
secondNodeIndex++){
                boolean edgeExists = false;
                for (EdgeData edgeData : edgesData) {
                    if (!((idConverter.getId(firstNodeIndex) ==
edgeData.getFirstNodeId() && idConverter.getId(secondNodeIndex) ==
edgeData.getSecondNodeId())
                            || (idConverter.getId(secondNodeIndex) ==
edgeData.getFirstNodeId() && idConverter.getId(firstNodeIndex) ==
edgeData.getSecondNodeId()))) {
                        continue;
                    }
                    printWriter.print(edgeData.getWeight());
                    edgeExists = true;
                    if (secondNodeIndex != nodesNumber) {
                        printWriter.print(" ");
                    }
                }
                if(!edgeExists) {
                    printWriter.print(0);
                    if (secondNodeIndex != nodesNumber) {
                        printWriter.print(" ");
                    }
                }
            }
            printWriter.println();
        }
        printWriter.close();
    }
}

class IdConverter{
    private Map<Integer, Integer> idMap;
```

```
    public IdConverter(){
        idMap = new HashMap<>();
    }

    public void addId(int id, int value){
        idMap.put(value, id);
    }

    public int getId(int value){
        return idMap.get(value);
    }
}
```

Файл: ControllerSynchronizer.java

```
package com.kruskal.controlstructures;

import com.kruskal.gui.ShapeController;
import com.kruskal.graph.Graph;
import com.kruskal.graph.GraphBuilder;

public class ControllerSynchronizer {
    private final ShapeController shapeController;
    private final GraphBuilder graphBuilder;
    private int freeNodeId;
    private int freeEdgeId;

    public ControllerSynchronizer(ShapeController shapeController, GraphBuilder
graphBuilder){
        this.graphBuilder = graphBuilder;
        this.shapeController = shapeController;
        freeNodeId = 1;
        freeEdgeId = 1;
    }

    public boolean addNode(double xCoordinate, double yCoordinate){
        if(graphBuilder.addNode(freeNodeId)){
            shapeController.addNode(xCoordinate, yCoordinate, freeNodeId);
            freeNodeId++;
            return true;
        }
        return false;
    }

    public boolean addEdge(int firstNodeId, int secondNodeId, int weight){
        if(graphBuilder.addEdge(firstNodeId, secondNodeId, weight, freeEdgeId)){
            shapeController.addEdge(firstNodeId, secondNodeId, weight,
freeEdgeId);
            freeEdgeId++;
            return true;
        }
        return false;
    }

    public boolean deleteNode(int nodeId){
        if(graphBuilder.deleteNode(nodeId)){
            shapeController.removeNode(nodeId);
            return true;
        }
        return false;
    }
```

```
    public boolean deleteEdge(int edgeId){
        if(graphBuilder.deleteEdge(edgeId)){
            shapeController.removeEdge(edgeId);
            return true;
        }
        return false;
    }

    public void eraseGraph(){
        freeEdgeId = 1;
        freeNodeId = 1;
        graphBuilder.resetGraph();
        shapeController.clear();
    }

    public Graph getGraph(){
        return graphBuilder.getGraph();
    }
}
```

Файл: Mediator.java

```
package com.kruskal.controlstructures;

import com.kruskal.fileparser.FileReader;
import com.kruskal.fileparser.FileWriter;
import com.kruskal.graph.Graph;
import com.kruskal.graph.GraphBuilder;
import com.kruskal.graph.GraphConnectednessExeption;
import com.kruskal.gui.ActionController;
import com.kruskal.gui.ActionMessage;
import com.kruskal.gui.ShapeController;
import com.kruskal.kruskalalgorithm.Kruskal;
import com.kruskal.kruskalalgorithm.StepMessage;

public class Mediator {
    private ActionController actionController;
    private ControllerSynchronizer controllerSynchronizer;
    private ShapeController shapeController;
    private Kruskal algorithm;
    private final FileReader fileReader;
    private final FileWriter fileWriter;

    public Mediator() {
        fileWriter = new FileWriter();
        fileReader = new FileReader();
    }

    public void setActionController(ActionController controller) {
        this.actionController = controller;
    }

    public void setShapeController(ShapeController shapeController){
        this.shapeController = shapeController;
        this.controllerSynchronizer = new
ControllerSynchronizer(shapeController, new GraphBuilder());
    }

    public void sendRequest(ActionMessage actionMessage) {
        switch (actionMessage.getState()) {
            case ADDNODE -> controllerSynchronizer.addNode(actionMessage.getX(),
actionMessage.getY());
```

```
            case ADDEDGE ->
controllerSynchronizer.addEdge(actionMessage.getObjectId(),
actionMessage.getSecondObjectId(), actionMessage.getWeight());
            case REMOVEEDGE ->
controllerSynchronizer.deleteEdge(actionMessage.getObjectId());
            case REMOVENODE ->
controllerSynchronizer.deleteNode(actionMessage.getObjectId());
            case RESTART -> controllerSynchronizer.eraseGraph();
            case REPLACENODE ->
shapeController.replaceNode(actionMessage.getX(), actionMessage.getY(),
actionMessage.getObjectId());
            case UPLOADGRAPH -> {
                controllerSynchronizer.eraseGraph();
                fileReader.read(actionMessage.getFileName(),
actionMessage.getX(), actionMessage.getY(), this);
            }
            case SAVEGRAPH -> fileWriter.write(actionMessage.getFileName(),
controllerSynchronizer.getGraph().getEdgesData(),
controllerSynchronizer.getGraph().getNodesId());
            case RUNALGORITHM -> {
                Graph graph = controllerSynchronizer.getGraph();
                if(graph.isValid()){
                    algorithm = new Kruskal(graph);
                }else{
                    throw new GraphConnectednessExeption("Graph is not
connected");
                }
            }
            case RUNALG -> {
                while (algorithm.isFinish()) {
                    StepMessage stepMessage = algorithm.makeStep();
                    actionController.printMessage(stepMessage);
                    shapeController.paintEdge(stepMessage);
                }

actionController.printTreeWeight(algorithm.getCurrentTreeWeight());
                //actionController.disableFinishAlgorithmButton(); //
Подразумевается, что есть метод для отключения кнопки
            }
            case NEXTSTEP -> {
                StepMessage stepMessage = algorithm.makeStep();
                actionController.printMessage(stepMessage);
                shapeController.paintEdge(stepMessage);

actionController.printTreeWeight(algorithm.getCurrentTreeWeight());
            }
            case PREVIOUSSTEP -> {
                StepMessage stepMessage = algorithm.stepBack();
                if (stepMessage != null) {
                    actionController.deleteLastMessage();
                    shapeController.paintEdgeDefault(stepMessage);

actionController.printTreeWeight(algorithm.getCurrentTreeWeight());
                } else {
                    actionController.blockPreviousStepButton();
                    actionController.deleteLastMessage();
                }
            }
        }
    }
    public void startAlgorithm() {
        while (algorithm.isFinish()) {
            StepMessage stepMessage = algorithm.makeStep();
```

```
                actionController.printMessage(stepMessage);
                shapeController.paintEdge(stepMessage);
            }
            actionController.printTreeWeight(algorithm.getCurrentTreeWeight());
        }
    }
```

ПРИЛОЖЕНИЕ В

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

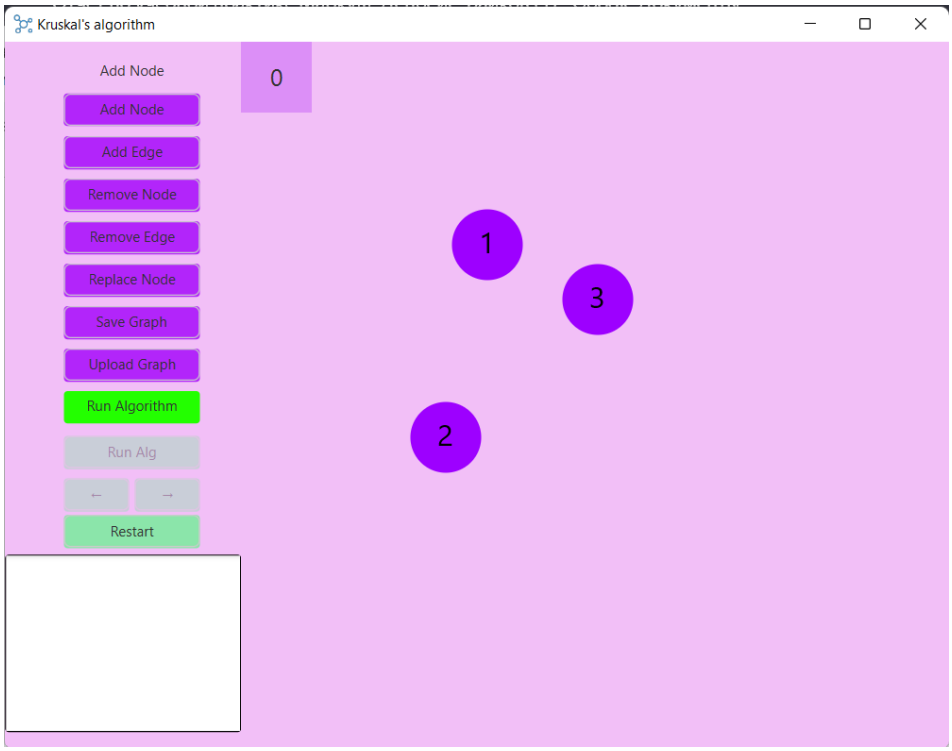

Рисунок 4 – Создание вершины графа

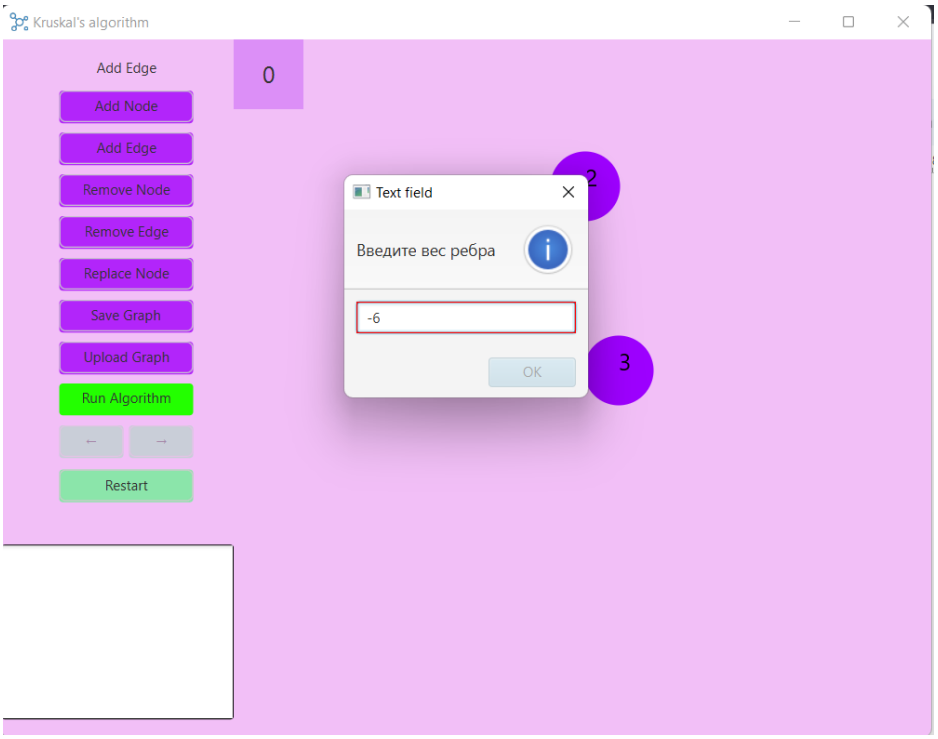

Рисунок 5 – Попытка создать ребро с отрицательным весом

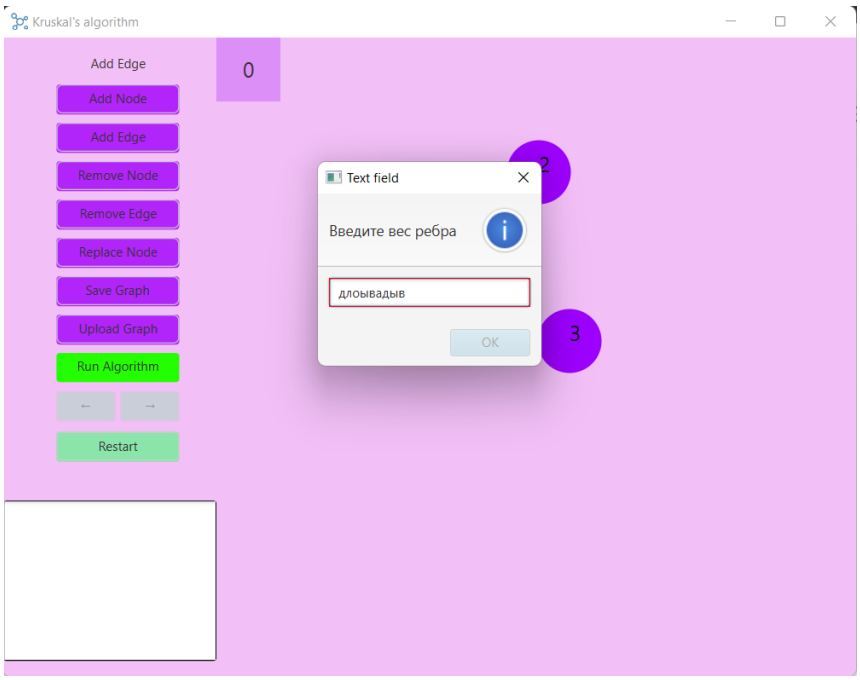

Рисунок 6 – Попытка создать ребро с некорректными данными вместо веса

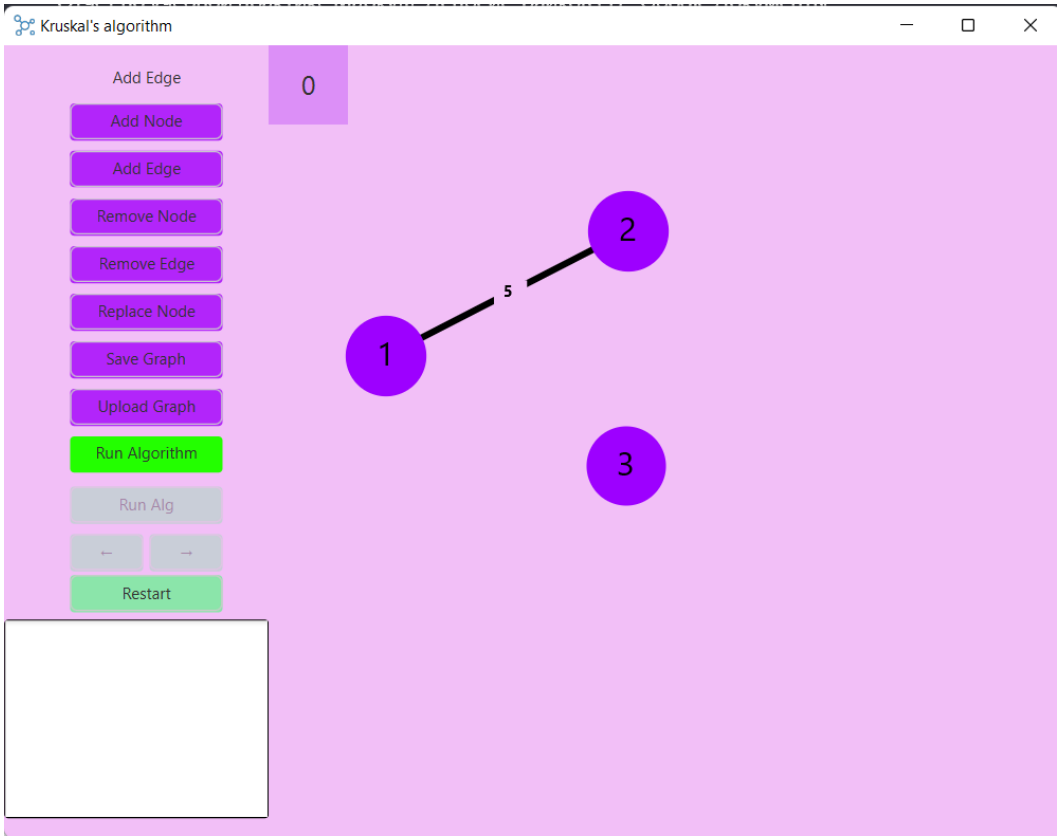

Рисунок 7 – Успешное создание корректного ребра

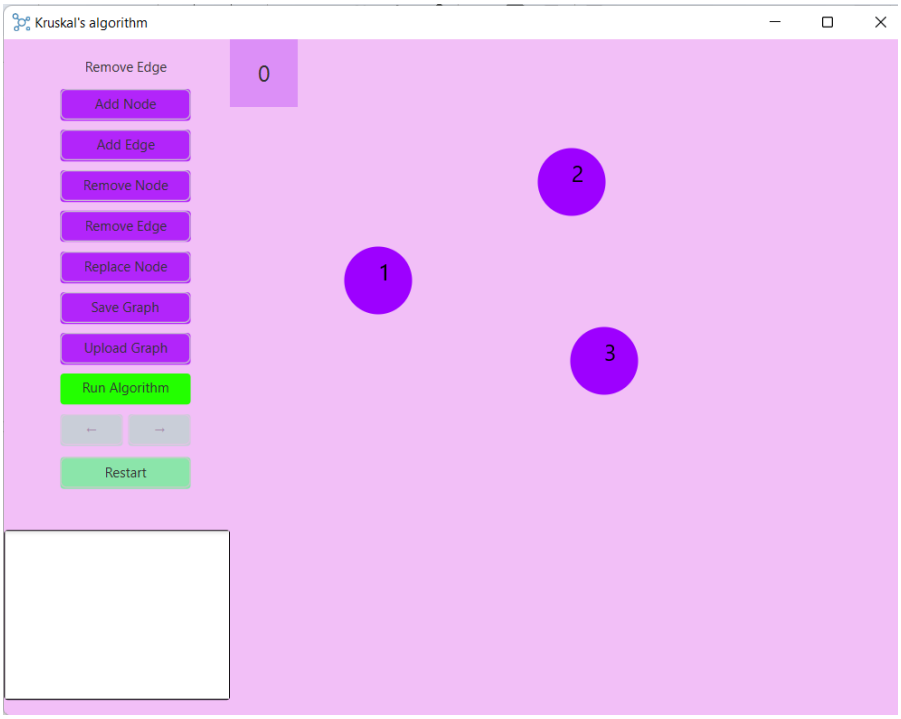

Рисунок 8 – Удаление ребра

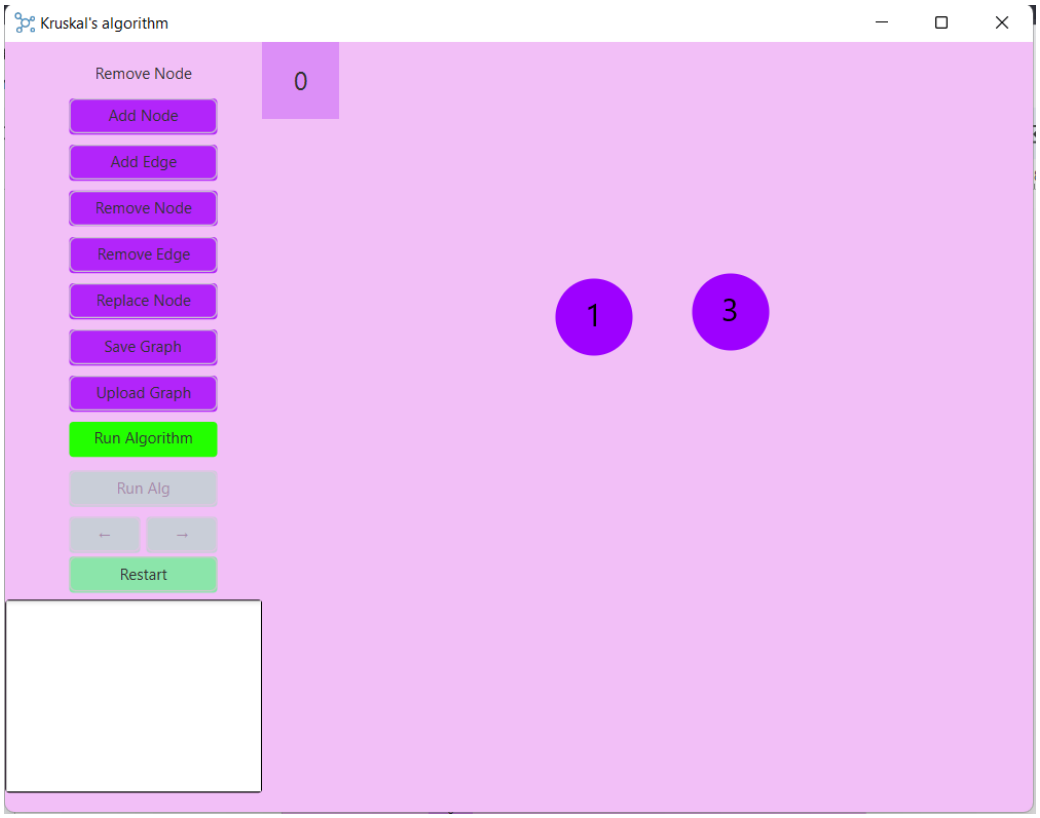

Рисунок 9 – Удаление вершины

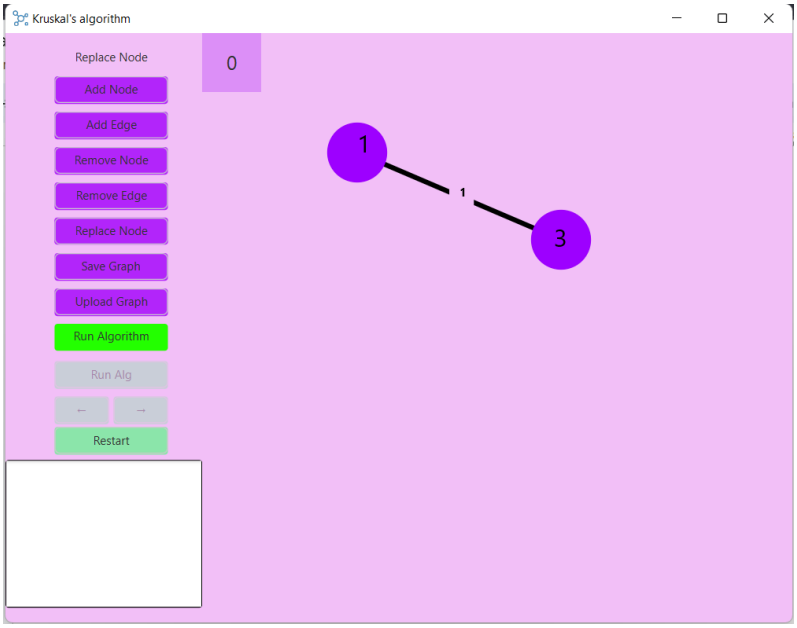

Рисунок 10 - Перемещение вершины

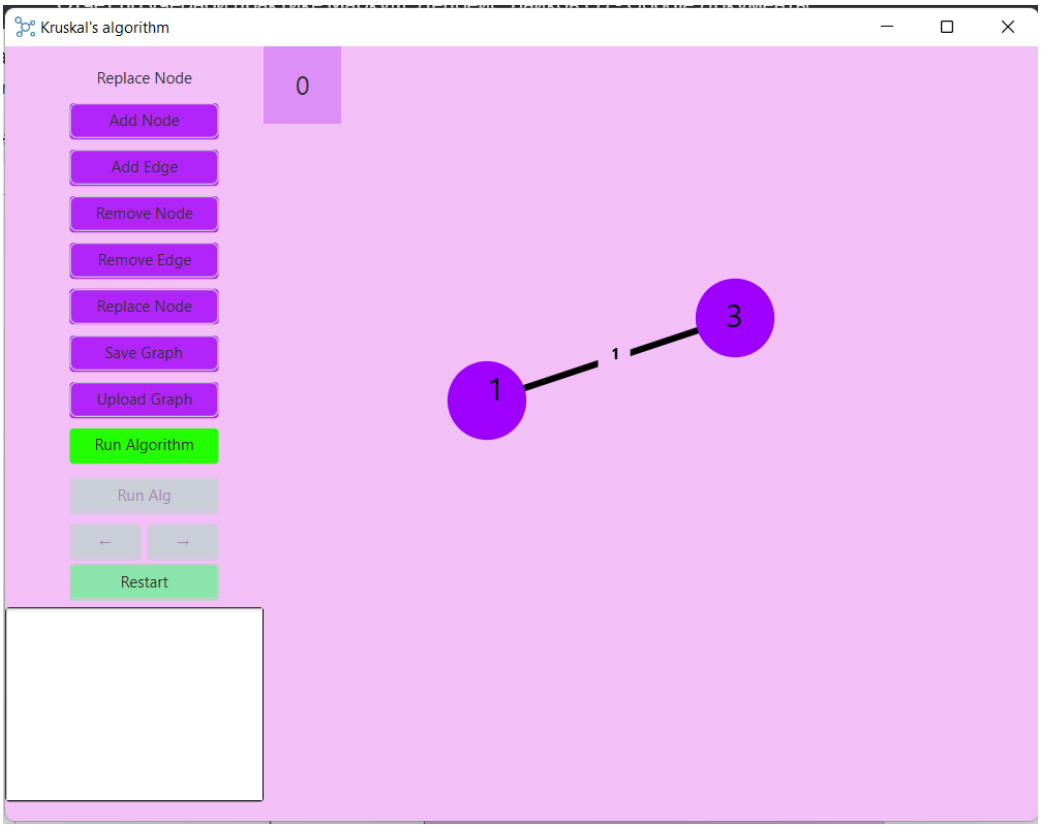

Рисунок 11 – Перемещение вершины

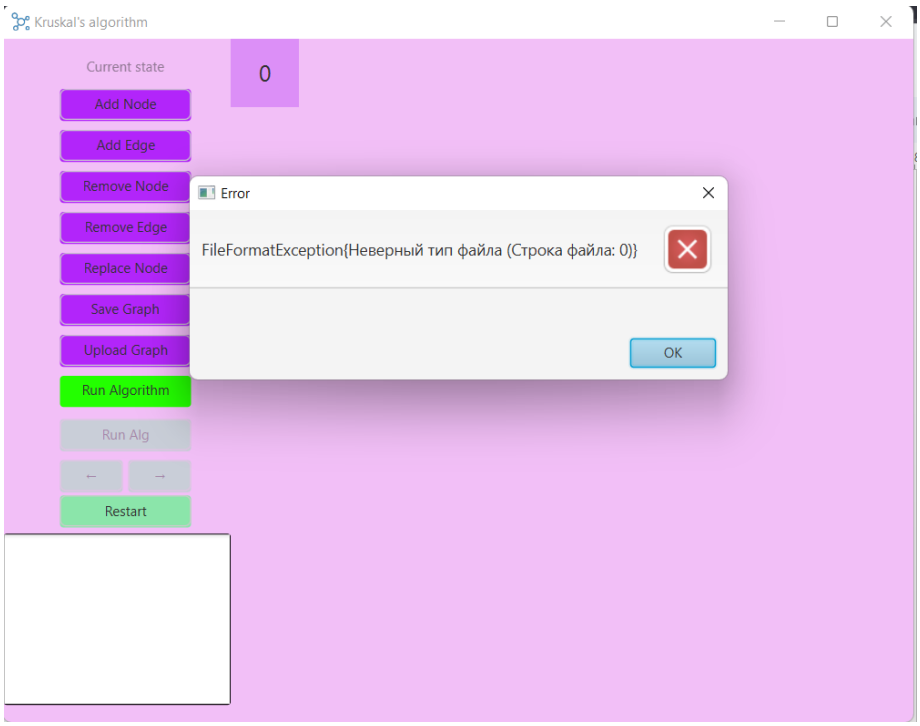

Рисунок 12 – Передан некорректный для считывания формат файла (не .txt)

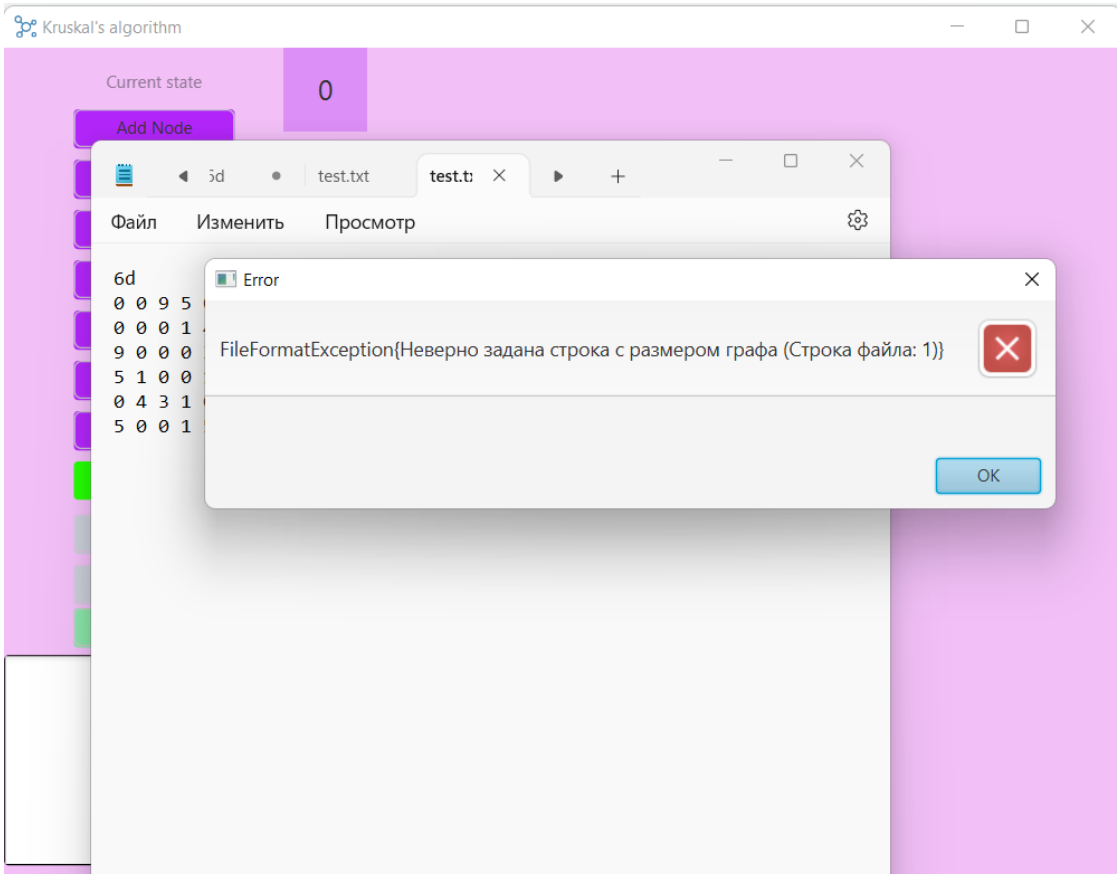

Рисунок 13 – Некорректная запись в строке с количеством вершин

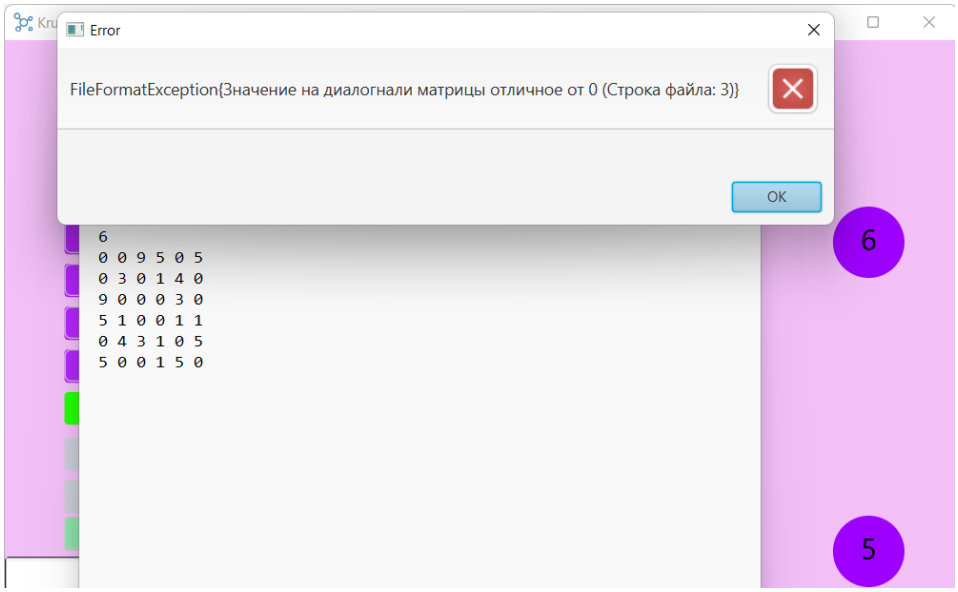

Рисунок 14 – На диагонали находится не 0 вес ребра, т.е. в гарфе есть петля

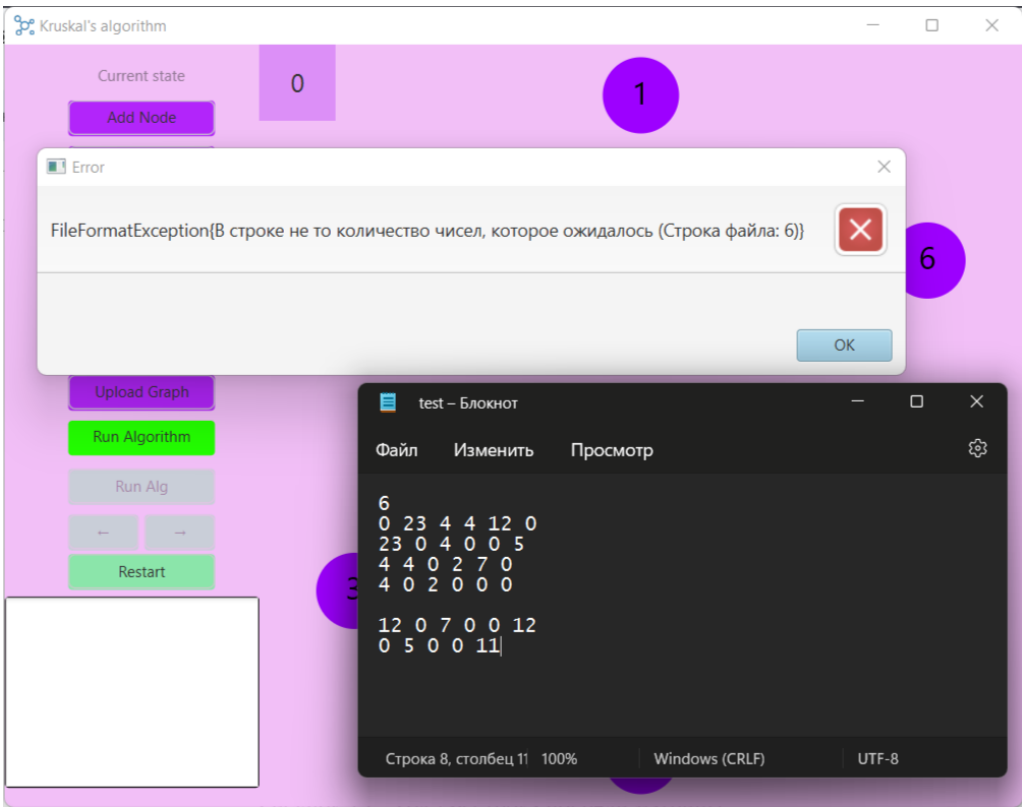

Рисунок 15 – Пустая строка посреди матрицы

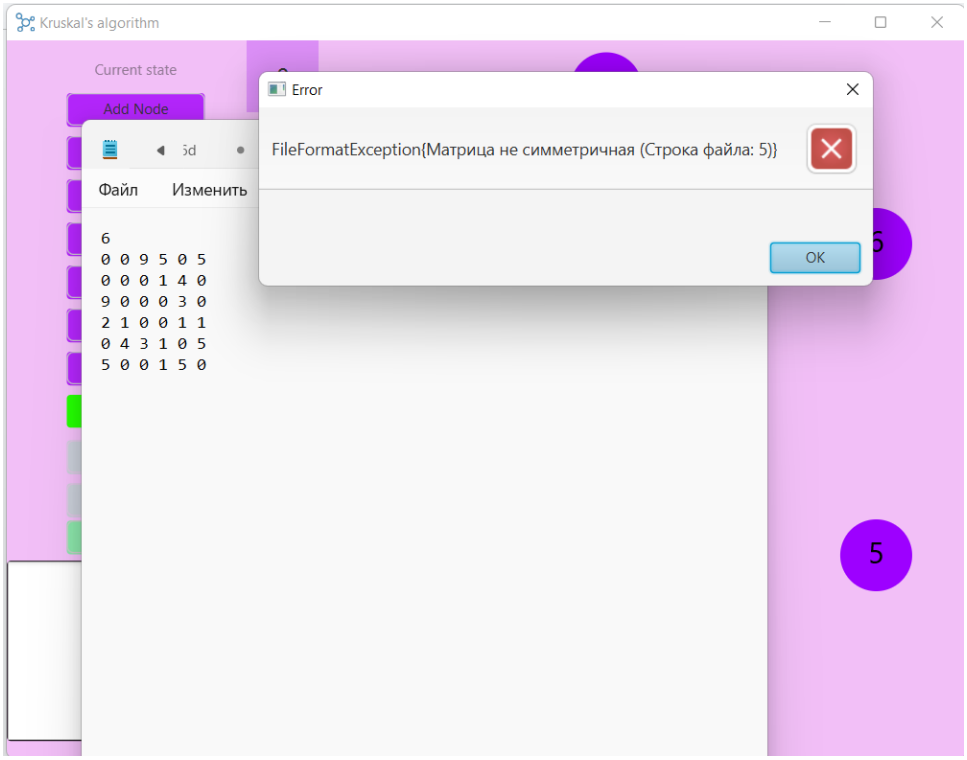

Рисунок 16 – Асимметрия матрицы рёбер

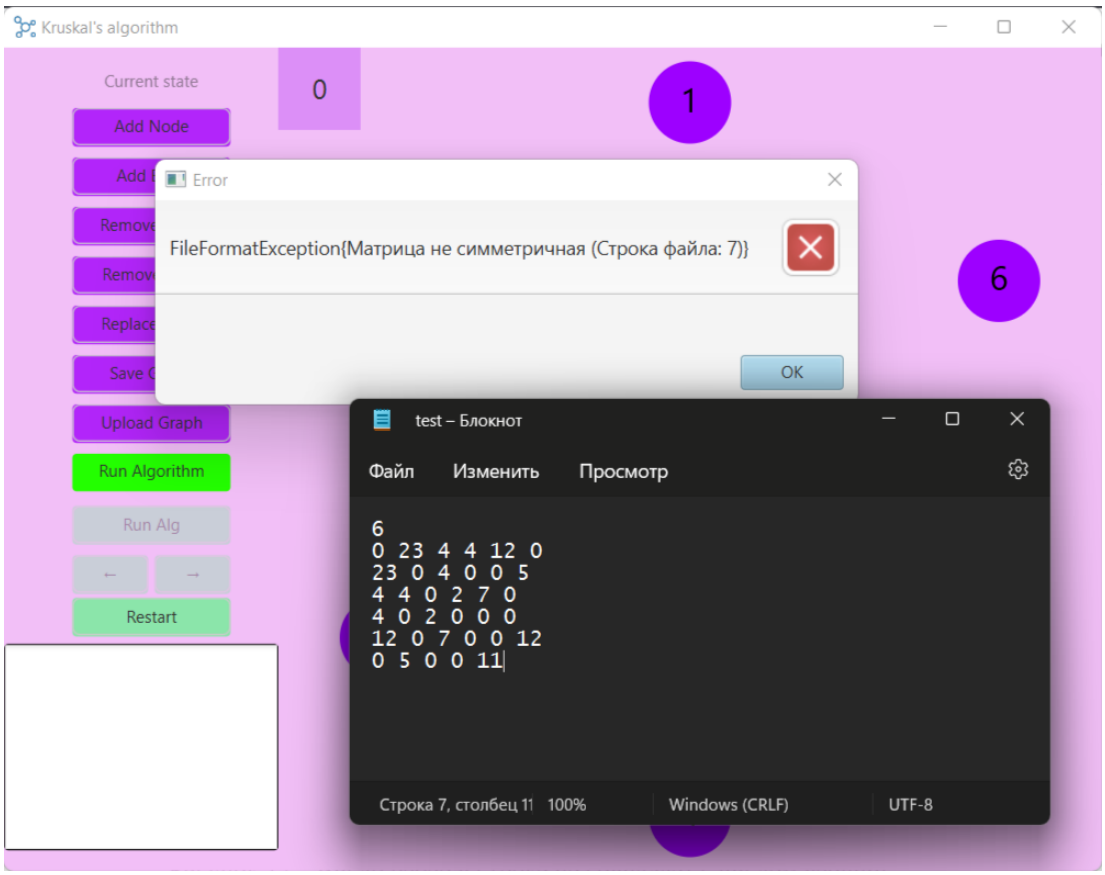

Рисунок 17 – Число рёбер в строке не совпадает с числом вершин

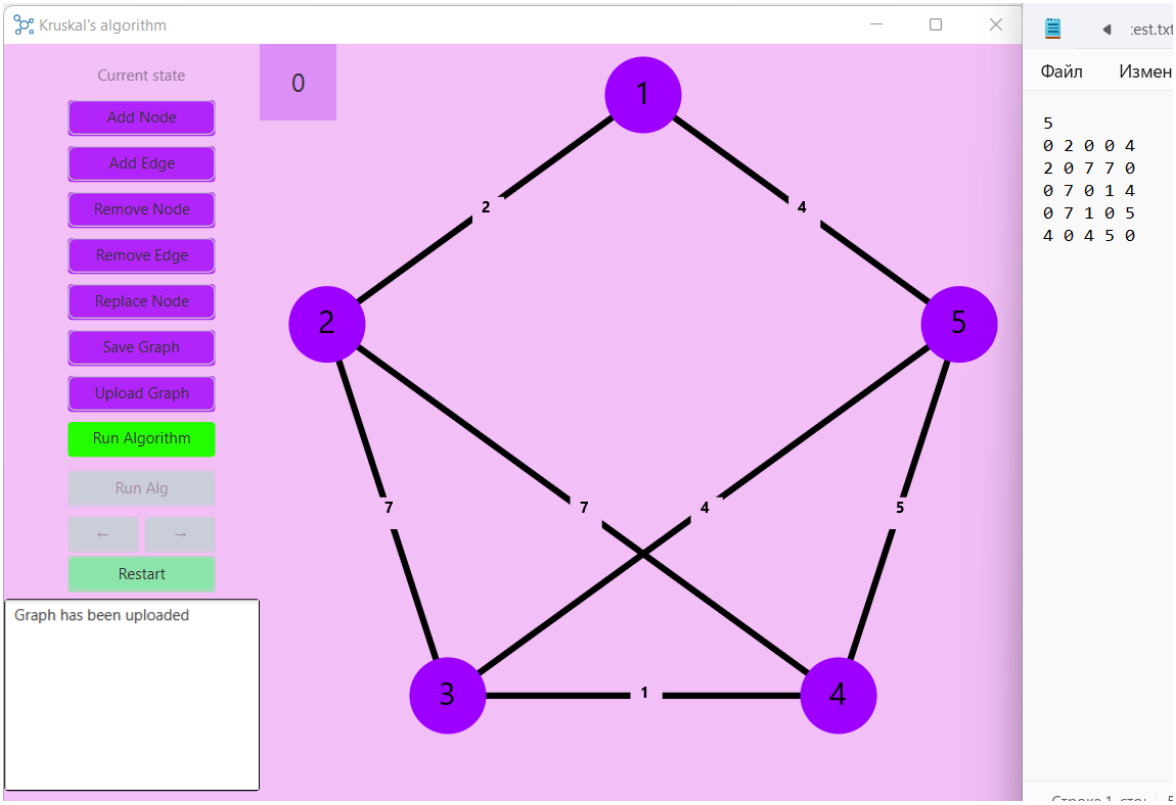

Рисунок 18 – Успешная загрузка графа из корректного файла

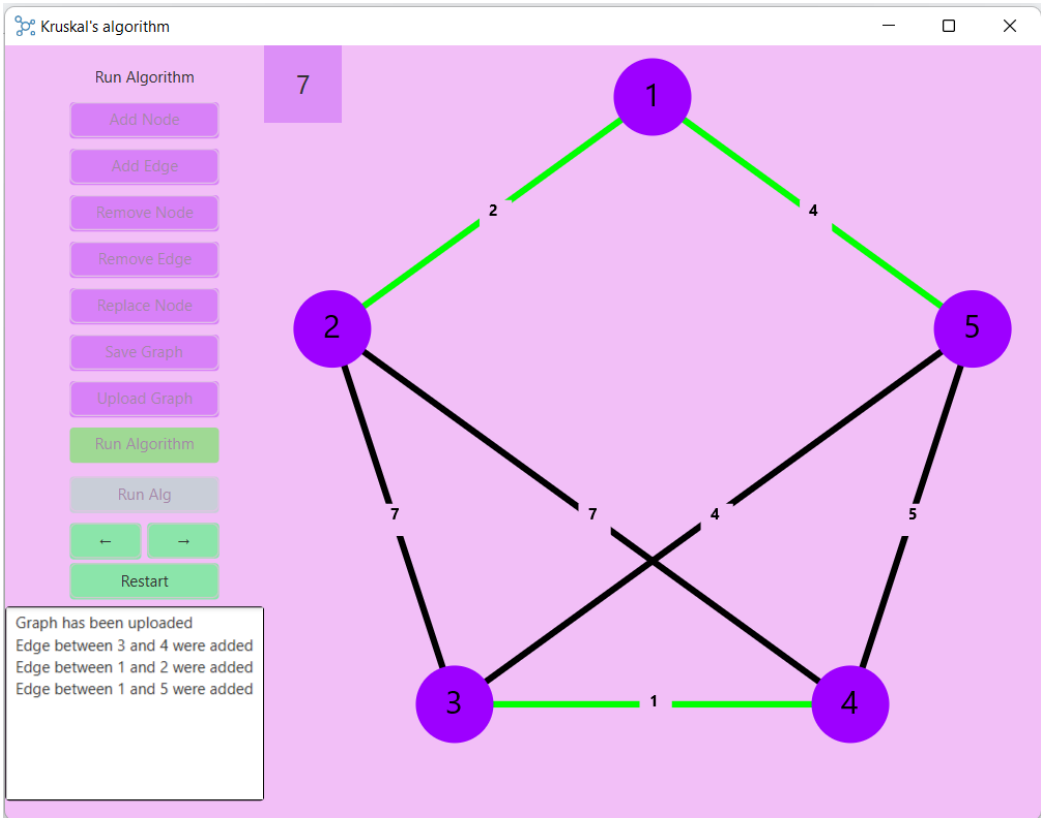

Рисунок 19 – Запуск алгоритма и прохождение нескольких шагов

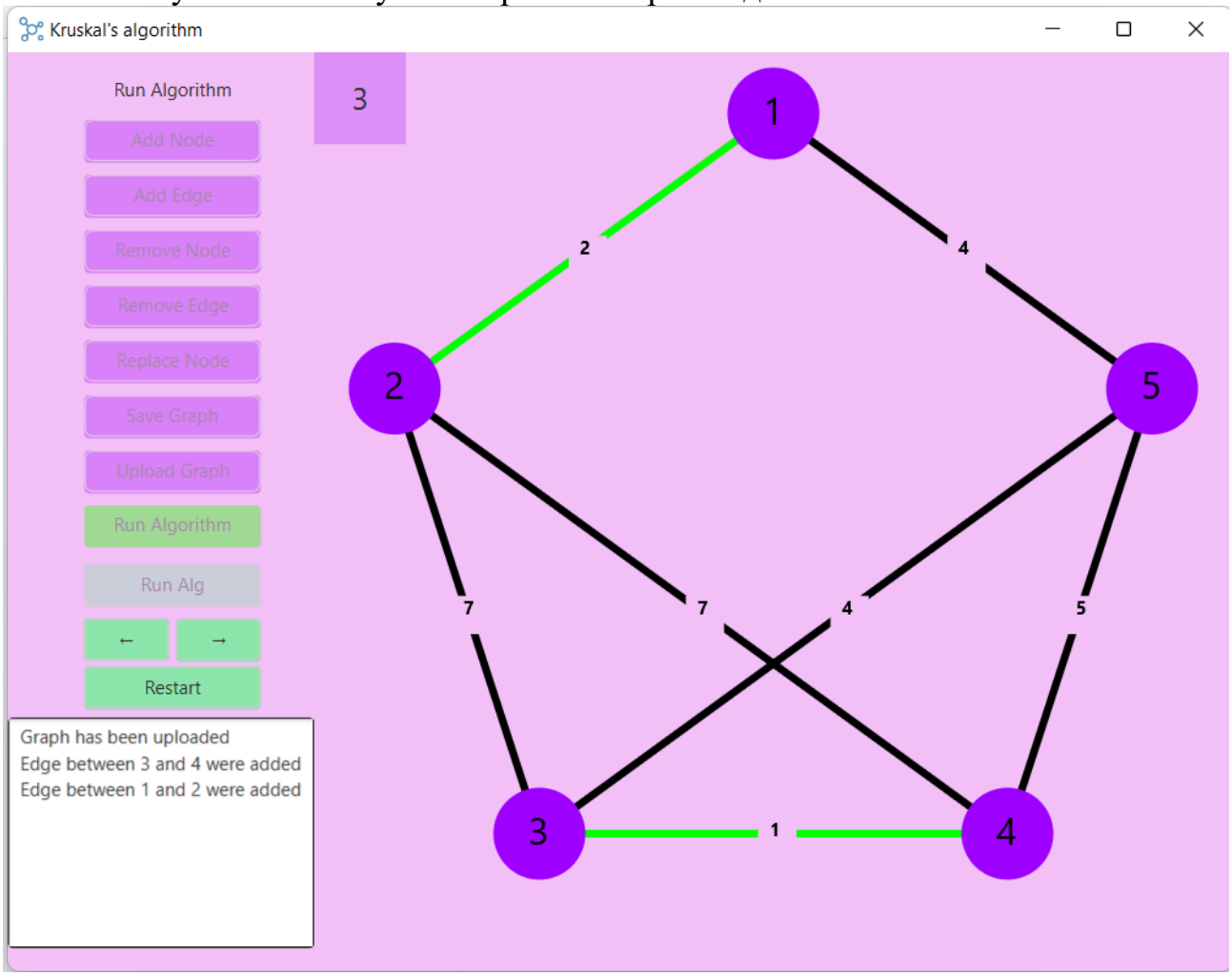

Рисунок 20 – Возвращение на шаг назад

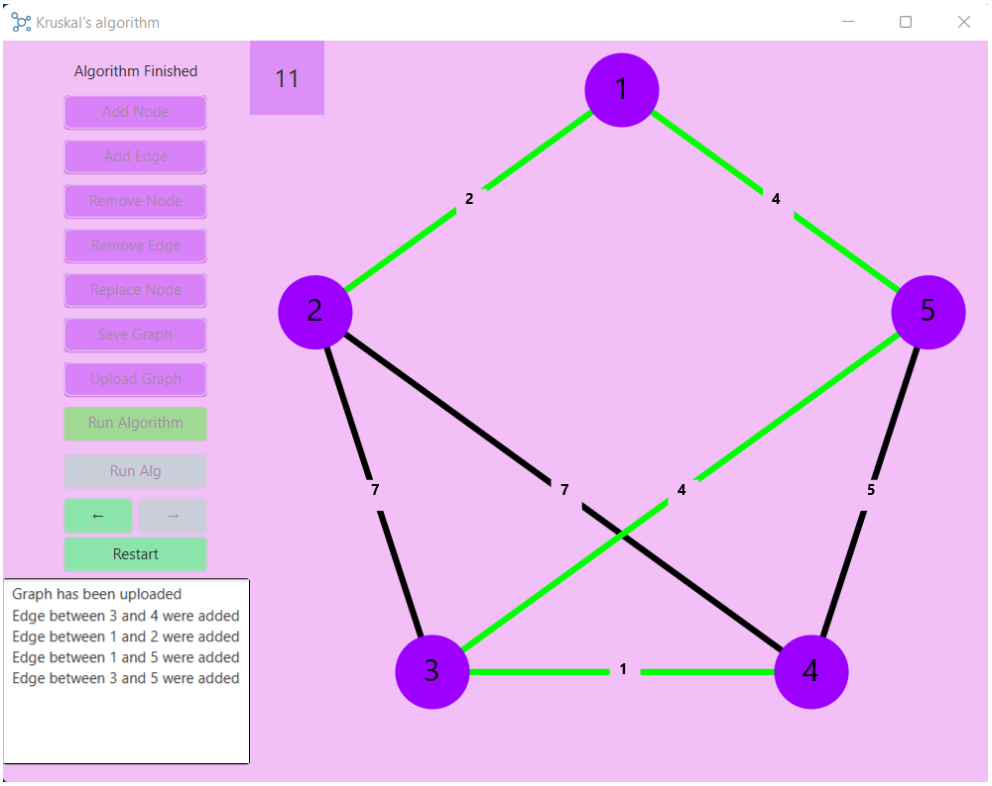

Рисунок 21 – Полностью отработавший алгоритм