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

ОТЧЕТ

по учебной практике

Тема: Кратчайшие пути в графе. Алгоритм Дейкстры

Студент гр. 1304 _____ Сулименко М. А.

Студент гр. 1304 _____ Клепнев Д. А.

Студент гр. 1381 _____ Таргонский М. А.

Руководитель _____ Шестопалов Р. П.

Санкт-Петербург
2023

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Сулименко М. А. группы 1304

Студент Клепнев Д. А. группы 1304

Студент Таргонский М. А. группы 1381

Тема практики: Кратчайшие пути в графе. Алгоритм Дейкстры

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Дейкстра.

Сроки прохождения практики: 30.06.2023 – 13.07.2023

Дата сдачи отчета: 13.07.2023

Дата защиты отчета: 13.07.2023

Студент		Сулименко М. А.
Студент		Клепнев Д. А.
Студент		Таргонский М. А.
Руководитель		Шестопалов Р. П.

АННОТАЦИЯ

Задача практики – реализовать графическое приложение, отображающее последовательную работу алгоритма Дейкстры. Перед выполнением были поставлены точные цели и сроки их реализации. После чего выполнялась работа по установленным срокам.

SUMMARY

The task of practice is to implement a graphical application that displays the sequential operation of Dijkstra's algorithm. Before implementation, precise goals and deadlines were set. After that, the work was carried out according to the established deadlines.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная практическая работа состоит в реализации графического отображения работы алгоритма Дейкстры. Результат работы – готовое приложение с графическим интерфейсом, позволяющее пользователю удобно настраивать параметры работы алгоритма и наблюдать за каждым этапом его работы.

Алгоритм Дейкстры – это алгоритм поиска кратчайшего пути между двумя вершинами на произвольном графе. Алгоритм проходит по всем ребрам графа, выбирая наиболее выгодный путь.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к функциональности

Программа должна выполнять ряд задач.

В первую очередь, программа должна уметь работать с графами любого типа: ориентированными и неориентированными. Будет лишь установлено ограничение в максимальном количестве ребер между любыми двумя вершинами.

В обоих видах графа будут запрещены «петли» - ребра, начало и конец которых совпадают, т.к. их наличие в графе не повлияет на результат работы алгоритма.

Также в ориентированных графах будет разрешено лишь одно ребро между двумя вершинами. Это также не повлияет на результат алгоритма, т.к., при наличии кратных ребер, алгоритм бы выбирал ребро с минимальной стоимостью, что может сделать пользователь вручную в процессе создания графа.

В неориентированных графах можно будет иметь максимум два ребра между любыми двумя вершинами, при условии, что эти ребра будут направлены в противоположные стороны. Опять же, такое сокращение не повлияет на результат алгоритма по тем же причинам, что и в ориентированных графах.

Во-вторых, она должна уметь реализовывать алгоритм Дейкстры. Алгоритм Дейкстры — это алгоритм поиска кратчайшего пути до всех вершин графа от некой фиксированной вершины S. В самом начале алгоритма считается, что расстояние до всех вершин графа бесконечно большое. Выбирается вершина с кратчайшим путём, которая помечается как проверенная с неким расстоянием. После чего, выбирается непроверенная вершина, смежная с любой из проверенных, которая имеет наименьший вес пути от вершины S. Вес считается по формуле $V_{рассматриваемая}=V_{ребра\_между\_вершинами}+V_{текущая}$. Если в процессе поиска обнаруживается более короткий путь, чем ранее найденный, то информация о найденном пути обновляется.

Приложение позволит наглядно продемонстрировать работу алгоритма Дейкстры. Для этого в процессе работы, алгоритм будет разделен на ряд шагов, которые будут обозначать какие-либо действия в алгоритме, поддающиеся несложной анимации. Переключаться между шагами можно будет с помощью

специальных кнопок управления на интерфейсе пользователя. Или же можно будет включить автоматическое воспроизведение шагов, когда шаги будут показываться друг за другом с определенной пользователем в настройках паузой. Также можно приостановить алгоритм с помощью кнопки паузы или отключить показ совсем на кнопку стоп. Пошаговое отображение позволит пользователю остановится в любой момент или же рассматривать алгоритм в обратном порядке.

В-третьих, приложение должно иметь интуитивно понятный интерфейс, позволяющий пользователю удобно пользоваться всеми возможность программы, позволяющий графически отображать граф и все его изменения, позволяющий реализовать некоторое количество инструментов по работе с графом, а именно: кнопки по постройке графа (режим перетаскивания вершин, режим добавления вершины, режим добавления ребра, режим очистки графа), кнопка настройки начальной и конечной точки, для работы алгоритма Дейкстры, кнопки запуска и остановки вычислений алгоритма, кнопки пошагового вывода следующего и предыдущего шага алгоритма Дейкстры, а также позволяющий наглядно видеть все шаги программы в соответствующей текстовой области. В отдельной области экрана должна выводится информация по происходящему на экране. Если пользователь взял инструмент, туда выведется информация о том, для чего предназначен этот инструмент. Во время работы алгоритма, каждый шаг будет в текстовом виде выведен в эту область.

Важным моментом является возможность сохранять граф. Для этого в отдельном месте интерфейса будут располагаться кнопки для обычного сохранения и так называемого «сохранения как», позволяющего настроить путь и название сохраняемого файла. При первом сохранении обе кнопки будут работать, как «сохранить как». Для сохранения графа будет использоваться технология JSON.

Также граф можно будет загружать из файла. Для этого также будет создана кнопка, при нажатии на которую, откроется интерфейс, сходный с проводником, для выбора загружаемого файла.

Помимо этого, должна быть возможность выводить граф на экран в виде ряда вершин (кругов), соединенных ребрами (отрезками). Нужно уметь выводить как ориентированный граф, так и неориентированный с соответствующими изменениями в виде ребер (отрезки со стрелкой или без). У

каждого элемента графа будет своя приписка, обозначающая какую-либо информацию об этом элементе. В случае вершины – это название вершины, в случае ребра – его вес. Кроме того, нужно иметь холст, на котором и будет рисоваться граф. При этом мы должны уметь увеличивать или уменьшать масштаб отображения холста.

В случае загрузки из файла, граф должен добавится на экран с использованием какого-либо правила расстановки вершин.

Также пользователь должен иметь возможность взаимодействовать с графом. Это включает в себя перемещение вершин графа. Нажимая и удерживая ЛКМ на вершине, можно будет переместить мышь, и вершина будет перемещаться с ней; создание ребер и вершин; стирание всего графа; выделение вершин для поиска кратчайшего пути.

Создание вершин подразумевает под собой нажатие по свободному участку экрана, где создастся вершина, под которой можно будет ввести ее название. Название не будет ограничено по размеру, но отображаться будут максимум лишь 4 символа с многоточием после них, если название не поместилось. Наведя на вершину, можно будет посмотреть ее полное название. Перемещение вершин в этом режиме будет разрешено.

Создание ребер будет выполнятся, как выбор пары вершин: начала и конца (в случае неориентированного графа это будет задавать направление). Выбор каждой вершины отображается в виде окантовки круга в определенный цвет. После чего будет нарисовано ребро, посередине которого нужно будет ввести его вес. Вес может быть только целочисленной переменной больше 0 и меньше 2*10^9.

Стирание графа – удаление из экрана всех построенных элементов графа.

Под выделением вершин для алгоритма поиска имеется в виду выбор двух вершин (с графическим подтверждением их выбора, как это было при создании ребра, но уже с другим цветом). Можно будет выбрать максимум 2 вершины. Нажатие на уже выбранную вершину будет снимать выбор.

Все элементы графа должны поддаваться стилистике вне классов самих элементов. Все стили должны указываться в специальном css файле.

1.1.2. Требования к интерфейсу

Интерфейс должен быть интуитивно понятен пользователю, т.е. назначение элементов интерфейса должно быть понятно без дополнительных надписей или с малым их количеством, и удобен в использовании.

Он будет состоять из двух режимов: режима настройки и режима воспроизведения. Режим настройки должен включать в себя набор инструментов, позволяющий настраивать граф перед запуском алгоритма (это включает в себя инструменты создания, перемещения и очистки графа, а также инструмент выбора начальной и конечной вершины последующего поиска кратчайшего пути). В то же время, режим воспроизведения будет содержать следующие инструменты: переход на шаг вперед/назад, запуск автоматического проигрывания шагов, пауза и стоп.

В самом верху программы будет меню с различными вкладками: File, Edit и Help. Вкладка File содержит в себе инструменты для создания нового графа с нуля, для загрузки графа из файла и для сохранения графа в файл, а также кнопку выхода из программы. Вкладка Edit состоит из продублированных инструментов из обоих режимов работы, из функции переключения этих самым режимов и из еще двух кнопок, одна из которых будет открывать окно с логами (куда будет выводится информация о любом взятом пользователем инструменте и информация по каждому проигранному шагу алгоритма), а другая – настройки, в которых можно настроить некоторые параметры отображения графа и стандартный путь сохранения всех графов (он будет автоматически открываться при первом сохранении). Остается вкладка Help, в которой будет кнопка, открывающая репозиторий с исходным кодом, и кнопка, открывающая окно с информацией о самом приложении.

Посмотрим на сам интерфейс:

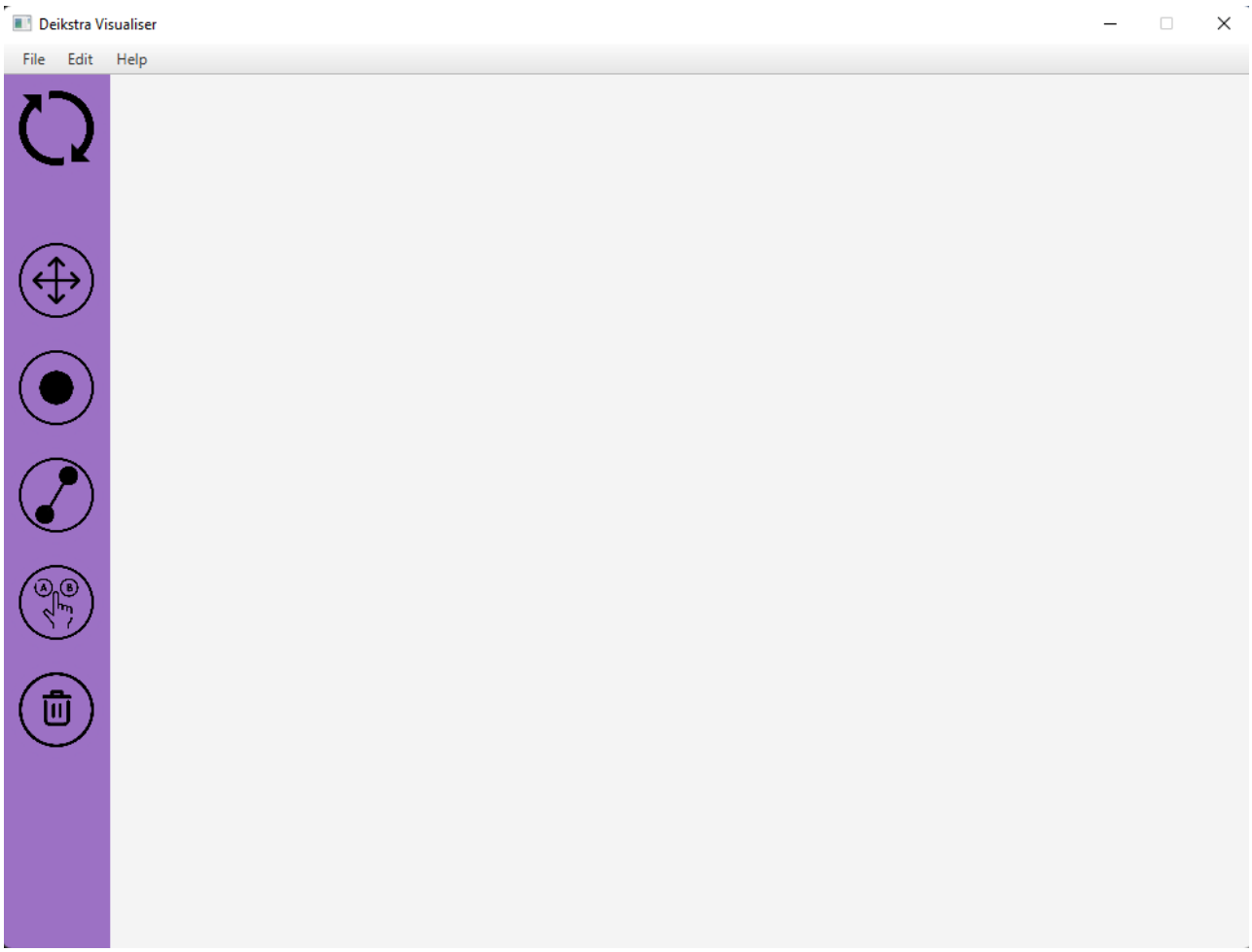


Рисунок 1 – Начальный экран программы.

Запустившись, приложение выводит то, что показано на рис. 1. Это окно можно менять в размерах. Пока что, ни один из инструментов не доступен для пользователя, т.к. для начала ему нужен граф. Он должен перейти во вкладку File в верхнем меню и выбрать подходящий для него метод создания или загрузки графа (см. рис. 2).

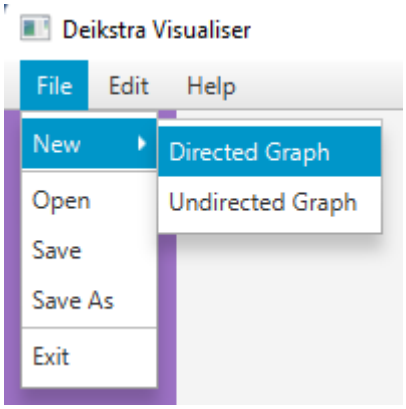


Рисунок 2 – Вкладка File из меню и ее содержимое.

Как видно из рисунка, существует два способа создать граф (создать ориентированный или неориентированный граф) и два способа его сохранить. При первом сохранении кнопка «Save» будет вести себя, как «Save As», т.е. будет открывать новое окно с проводником, спрашивая пользователя, куда он хочет сохранить свой граф. В той же вкладке меня можно выйти из приложения (стандартный способ для всех приложений также работает).

Создав или загрузив граф, пользователю открываются для использования инструменты его настройки. Рассмотрим каждый в отдельности:

Кнопка на рис. 3 отвечает за перемещение вершин графа. Пока она активна каждую вершину можно перетащить вышкой по холсту.

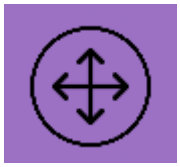


Рисунок 3 – Кнопка перемещения вершин графа

Следующая кнопка на рис. 4 отвечает за создание новых вершин. Активировав ее, можно будет создавать вершины, просто ткнув в любое свободное от элементов графа место на холсте.

Рисунок 4 – Кнопка создания новых вершин

Далее идет кнопка создания новых ребер. Она расположена на рис. 5. С ее помощью, можно будет выделить 2 вершины, между которыми образуется новое ребро.

Рисунок 5 – Кнопка создания новых ребер

Важной кнопкой является кнопка выбора двух вершин для последующей работы алгоритма. Это кнопка на рис. 6.

Рисунок 6 – Кнопка выбора вершин для алгоритма

И остается кнопка, которая полностью стирает весь граф с экрана. Она на рис. 7.

Рисунок 7 – Кнопка стирания всего графа

На самом деле осталась еще одна кнопка в самом верхнем левом углу (см. рис. 8). Но она занимается не настройкой графа, а переключением режимов. Такая кнопка есть в обоих режимах.

Рисунок 8 – Кнопка переключения режимов

Нажмем на нее и посмотрим, что находится в другом режиме (см. рис. 9). Как видно из рисунка, панель слева убралась и появилась панель сверху. Эта панель уже имеет другой окрас и другой набор инструментов. Это мы переключились в режим воспроизведения.

В этом режиме нам доступны инструменты управления воспроизведением, такие как: переход на шаг назад или вперед, старт воспроизведения, пауза и стоп.

Если в предыдущем режиме не были выбраны вершины для алгоритма, то все инструменты этого режима будут заблокированы. В ином же случае, после переключения будет доступна для нажатия кнопка запуска, как бы показывая, что можно начать показ шагов. При нажатии на эту кнопку, запустится автоматическое воспроизведение шагов с небольшой, но достаточной для принятия решения приостановки показа, паузой и разблокируются все остальные кнопки. Приостановку показа можно сделать, нажав на кнопку паузы. Если автоматическое воспроизведение приостановлено, то шагами можно управлять кнопками налево и направо, перемещаясь по одному шагу назад и вперед соответственно. Или же можно нажать на эти кнопки самостоятельно, и тогда

программа сама отключит автоматический показ шагов. Если дальнейший просмотр алгоритма не нужен или хочется быстро вернутся в его начало, то можно полностью остановить показ на кнопку стоп.

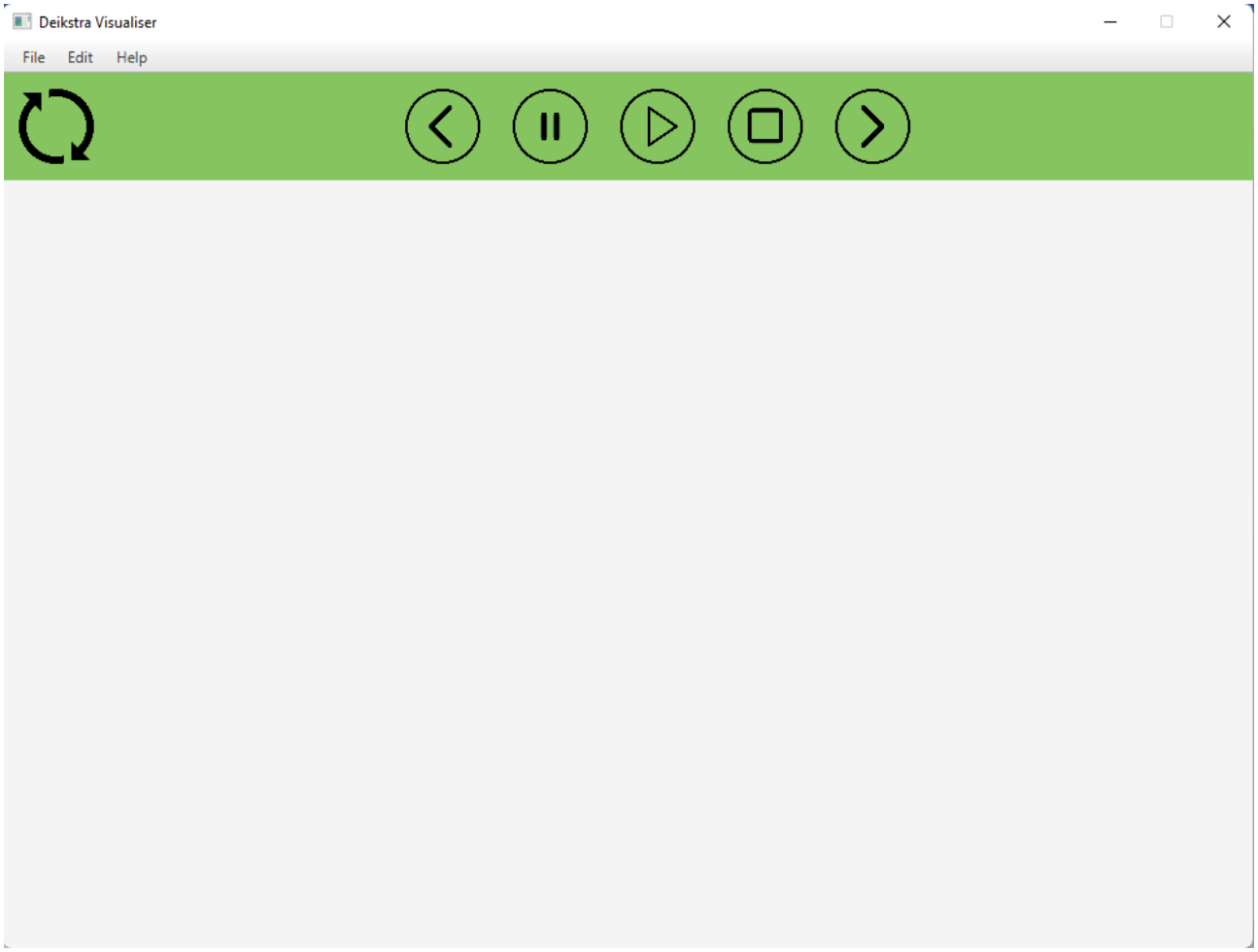


Рисунок 9 – Режим воспроизведения

Теперь вернемся ко вкладкам панели меню. Во вкладке Edit (см. рис. 10) располагаются кнопка переключения режимов, два списка с инструментами режимов (доступность этих инструментов соответствует описанному выше), кнопка, открывающая логи (см. рис. 11), и кнопка, открывающая настройки приложения (см. рис. 12), в которых можно изменить некоторые параметры отображения графа и стандартный путь сохранения.

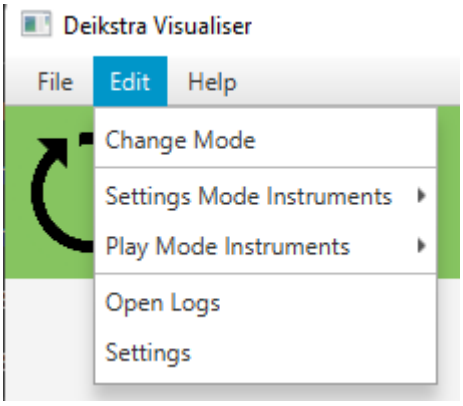


Рисунок 10 – Вкладка Edit

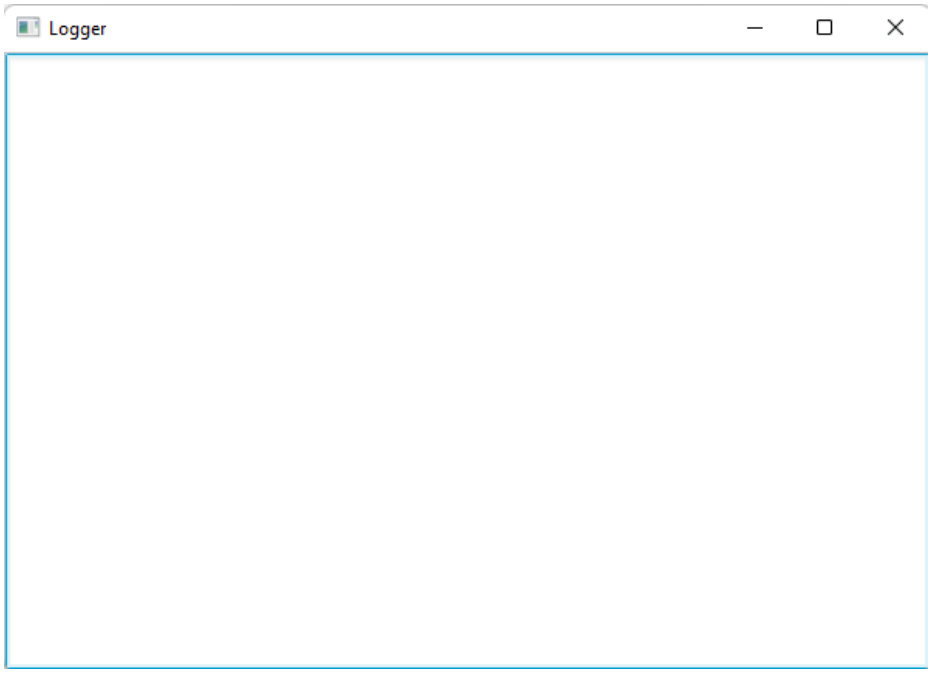


Рисунок 11 – Окно логгера

Рисунок 12 – Окно настроек

И остается последняя вкладка Help (см. рис. 13), в которой традиционно располагается кнопка, открывающая окно с информацией о программе (см. рис. 14), и кнопка, отсылающая на репозиторий проекта.

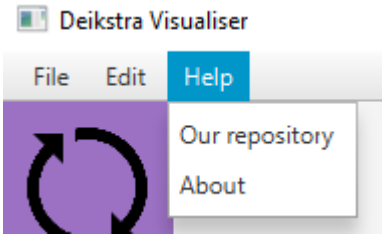


Рисунок 13 – Вкладка Help

1.1.3. Требования к визуализации алгоритма

Работа алгоритма должна разделятся на логические части, в каждой из которых будет сохранятся состояние алгоритма в отдельный список. Далее, используя этот список, визуализатор будет пошагово (управление переключением шагов остается за пользователем) выводить работу алгоритма.

Будут выделены следующие выводимые этапы алгоритма:

- Проверка ребёнка у узла:
  - Для каждого узла, исходящего из текущего, проверяется нахождение нового более дешёвого пути.
  - Визуализация: цветом выделяется окантовка проверяемой вершины и ребро до него.
- Обновление информации у ребёнка:
  - В случае нахождения более дешёвого пути до ребра в ней обновляется информация, такая как наименьший вес пути до неё и узел, из которого можно до неё добраться с наименьшим весом.
  - Визуализация: полная перекраска вершины определённым цветом.
- Переход к узлу:
  - Из вершин из очереди на обработку, выбирается новая текущая вершина с наименьшим весом
  - Визуализация: полная отрисовка пути с нуля.

Шагом в работе алгоритма будет считаться выполнение любого из перечисленного этапа.

При запуске алгоритма на каждом шаге будет сохраняться информация о текущем состоянии графа и очереди вершин. После завершения работы алгоритма класс пошагового отображения сможет получить информацию о пройденных шагах, чтобы отобразить их в графическом интерфейсе с

возможностью переключения между шагами, причём как вперёд, так и назад. Все пути будут анимированы (ребра будут перекрашиваться постепенно друг за другом с определенной задержкой).

1.1.4. Требования к архитектуре приложения

UML-схема базовых классов для хранения и работы с графами различного типа:

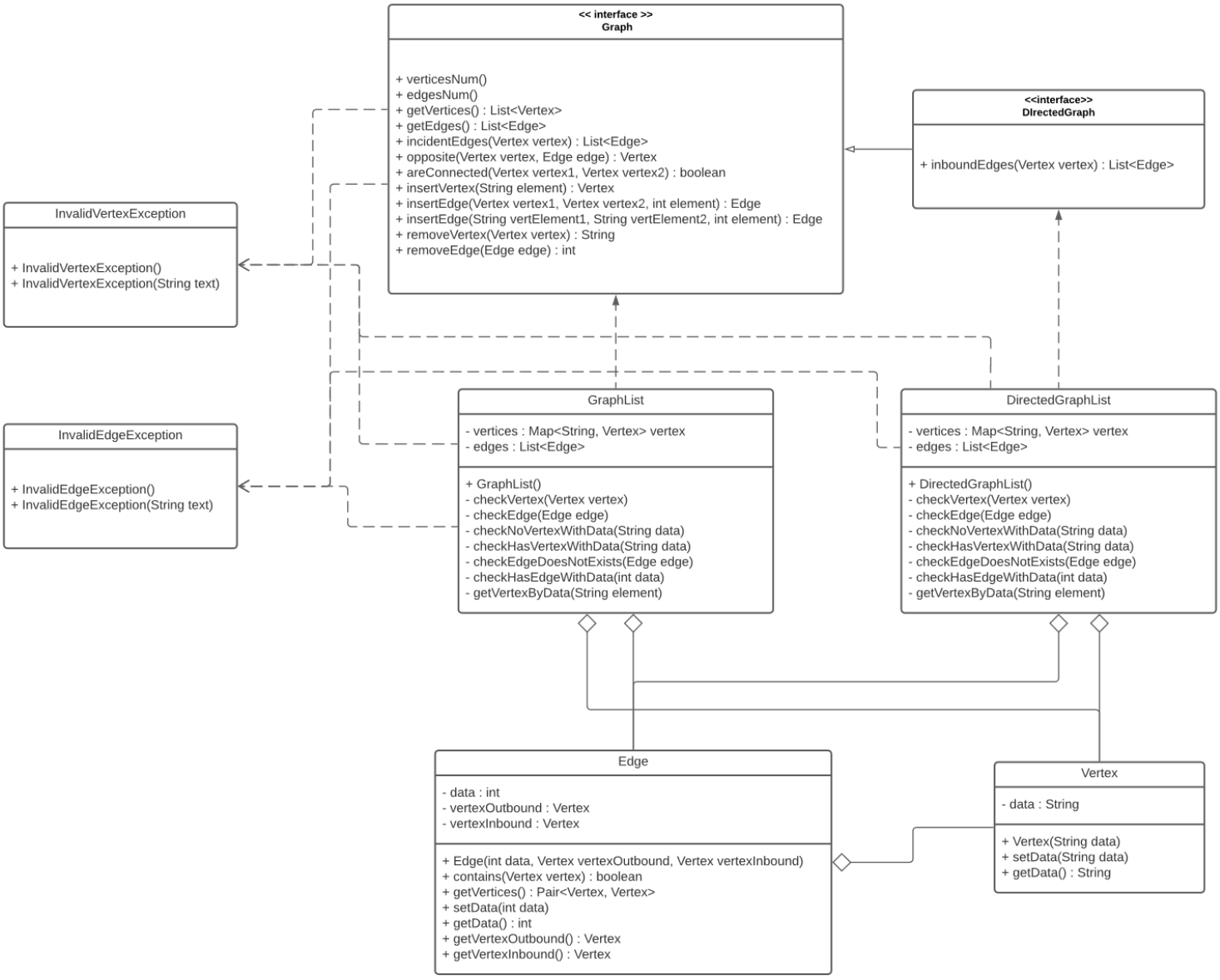


Рисунок 14 – UML-схема базовых классов для хранения и работы с графами

Базовые классы для работы с графической составляющей создания графа:

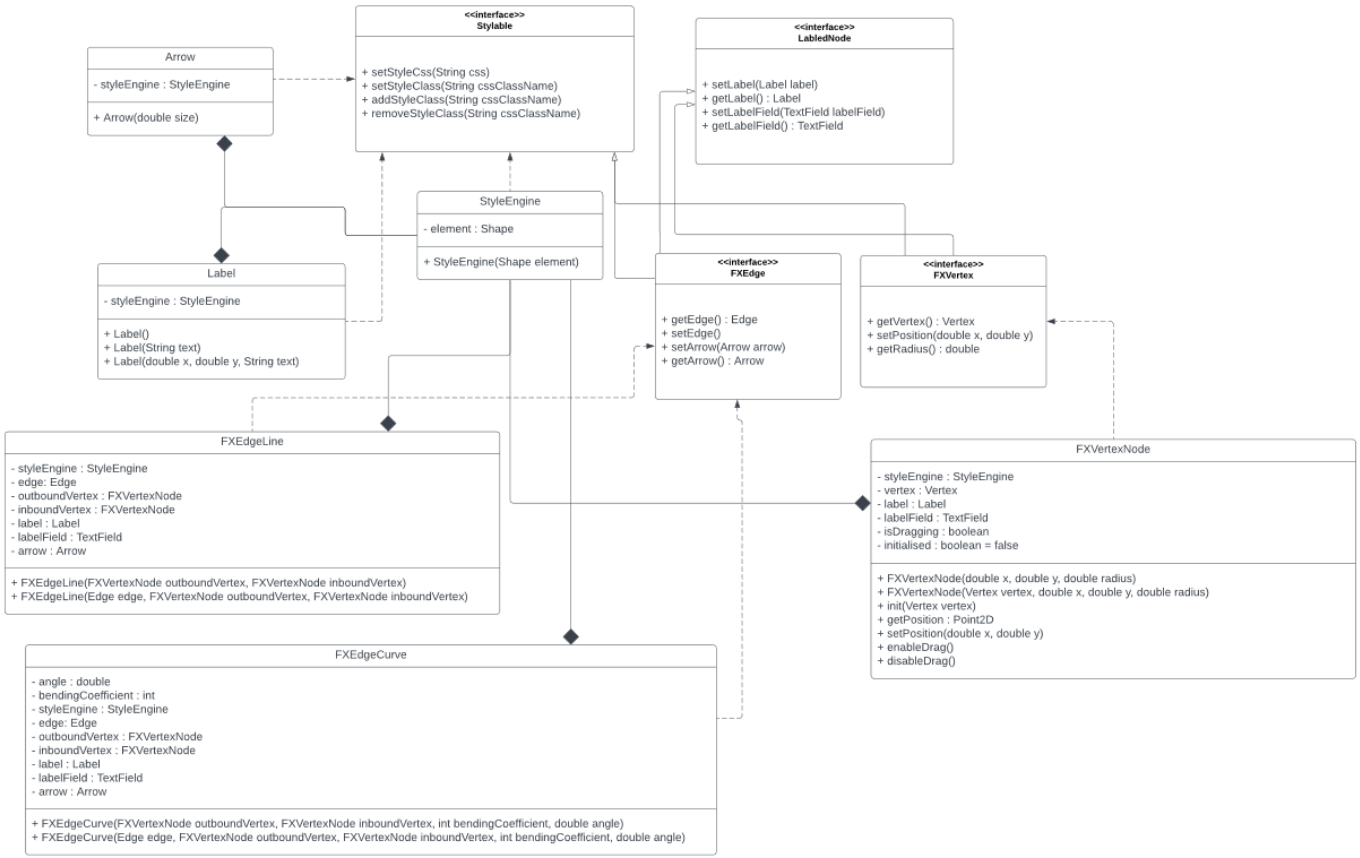


Рисунок 15 – Базовые классы для работы с графической составляющей создания графа

Классы отрисовки:

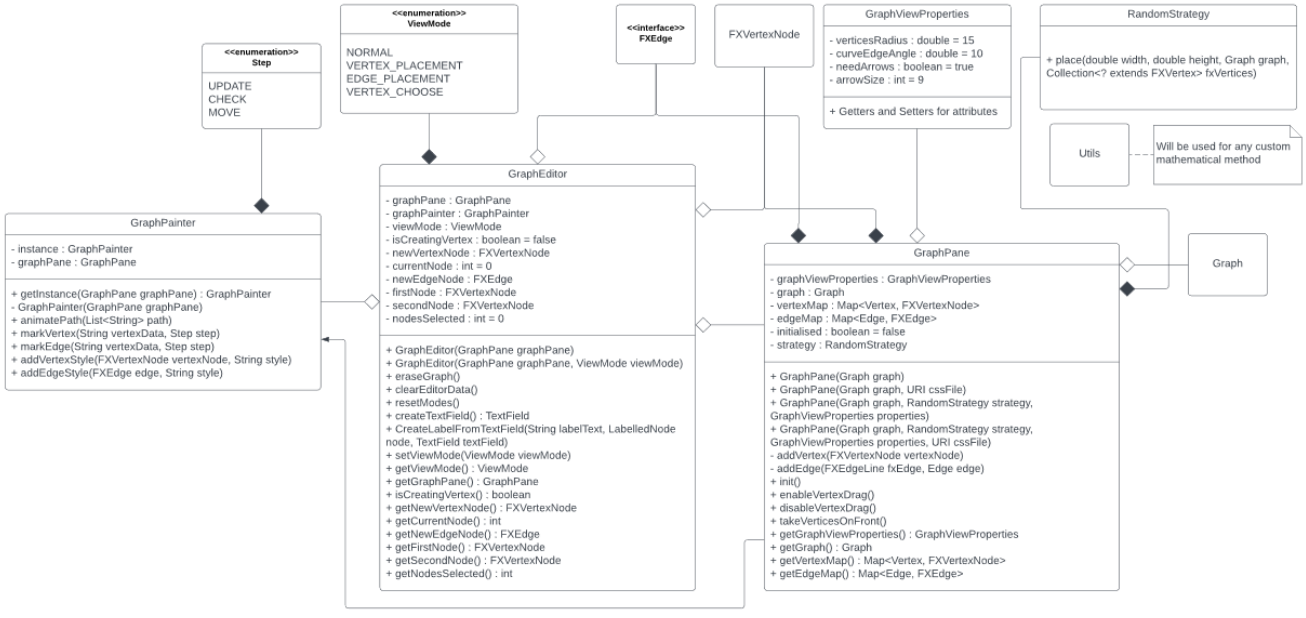


Рисунок 16 – Классы отрисовки

Классы управления алгоритмом Дейкстры:

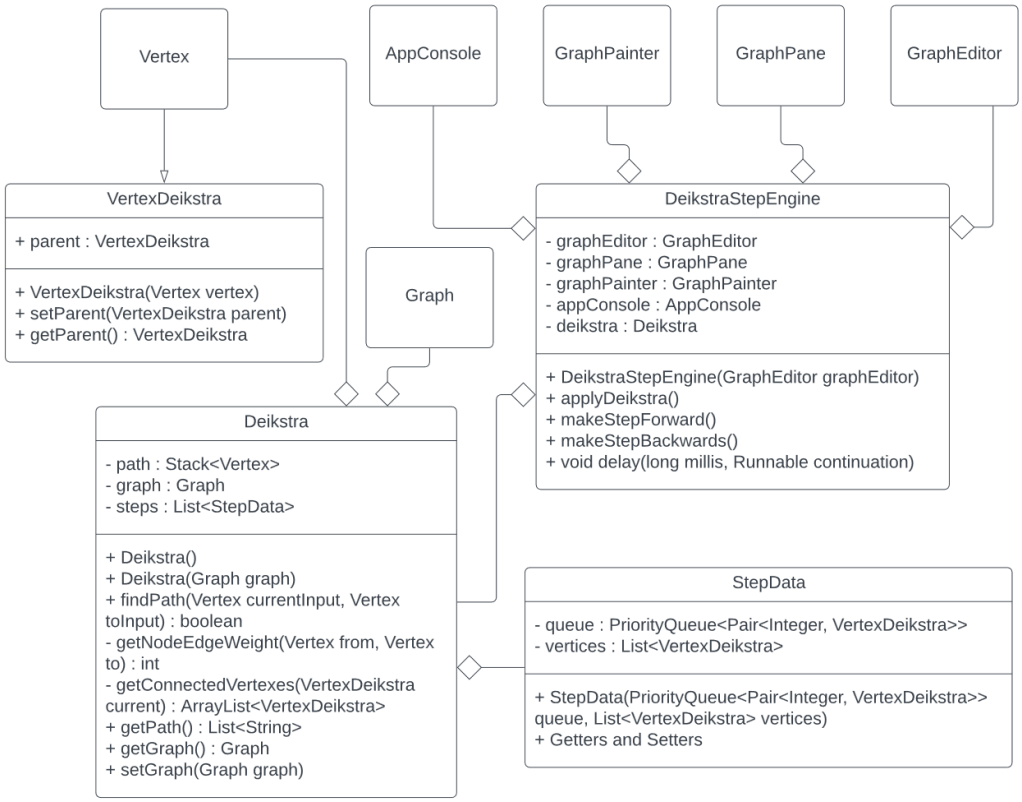


Рисунок 17 – Классы отрисовки

Классы для сохранения и загрузки графа:

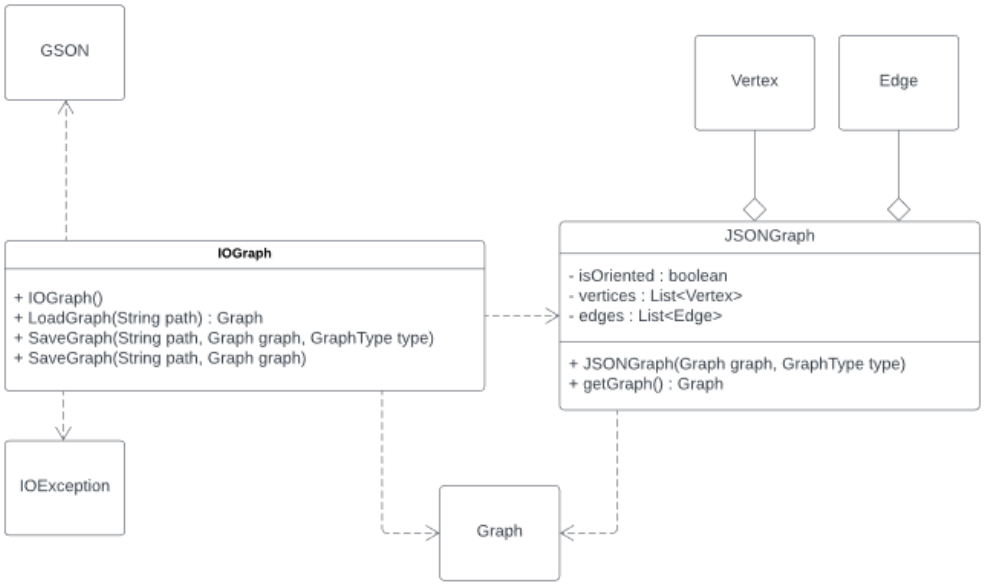
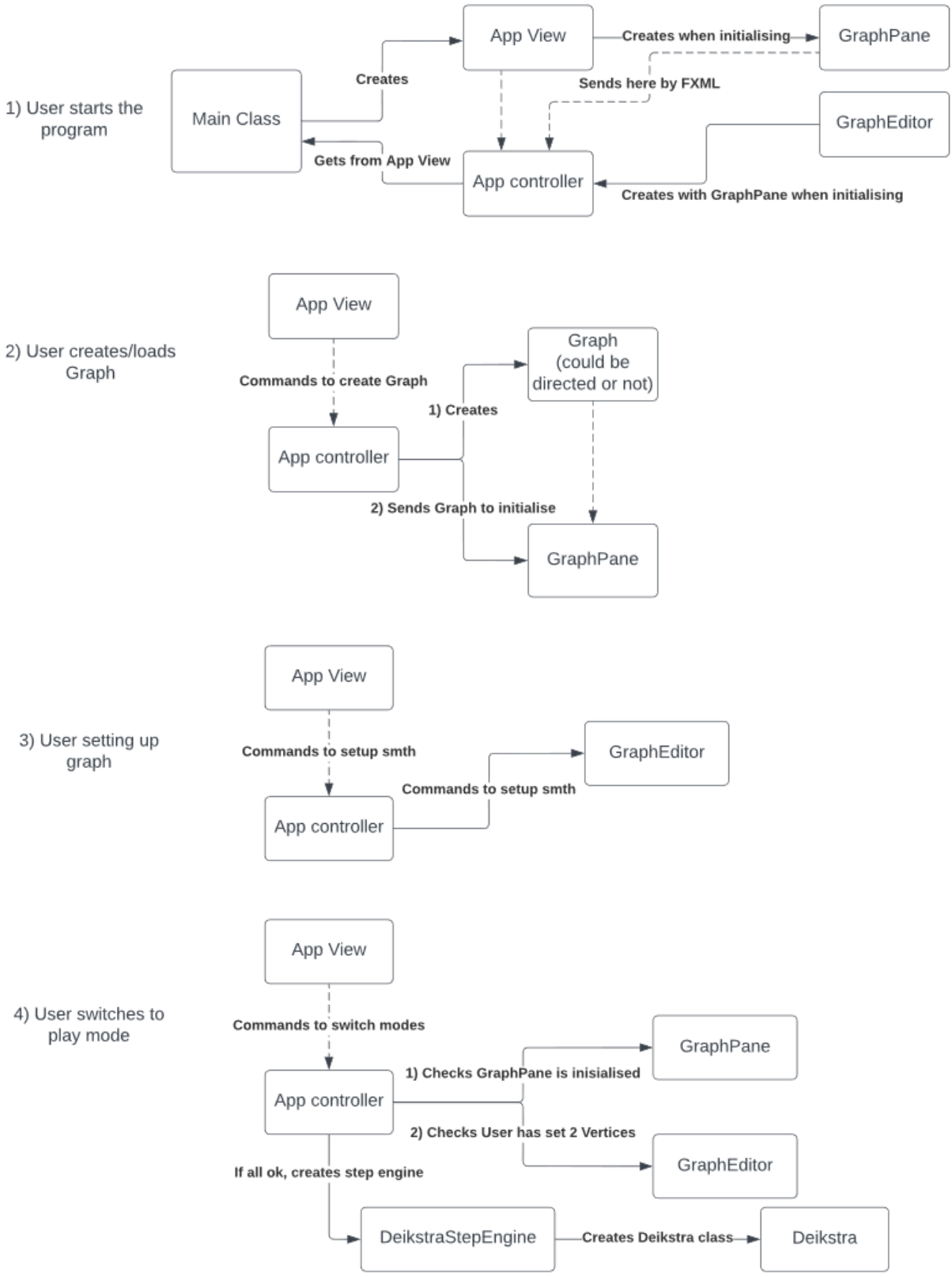


Рисунок 18 – Классы для сохранения и загрузки графа

Архитектура GUI:



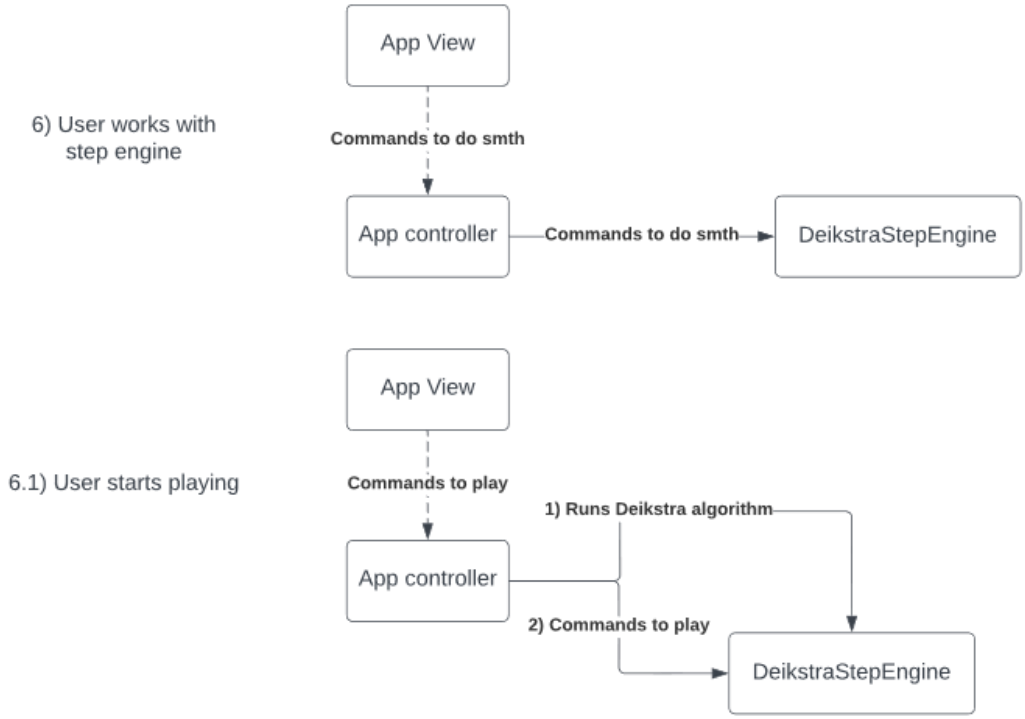


Рисунок 19 – Архитектура GUI

1.1.5. Требования к тестированию

Тестирование приложения будет разбито на 2 вида: автоматическое тестирование и ручное. Автоматическое тестирование (далее Unit-тесты) будет реализовано при помощи библиотеки JUnit и будет применятся к методам и классам, в которых подобное тестирование будет возможно. Это все классы, не связанные с реализацией графического интерфейса. В остальных случаях будет применено ручное тестирование.

Unit-тесты планируется написать до непосредственной реализации методов для последующего ускорения разработки. Ручное тестирование будет производится после создания элементов интерфейса.

1.1.6. Заключение

Подводя итоги вышенаписанного, приходим к тому, что программа должна уметь выполнять алгоритм Дейкстры, должна уметь выводить его пошагово на экран с определенными анимациями шагов на построенном ранее графе. Шагами можно управлять кнопками переключения шагов или кнопкой автоматического воспроизведения, паузы и полной остановки. Остановить показ можно кнопкой стоп.

Граф можно построить вручную с помощью инструментов создания вершин и ребер, очистки графа (предварительно создав новый граф) или

загрузить ранее сохраненный вариант. Граф можно будет увеличивать или уменьшать в размере. На построенном графе можно будет выбрать начальную и конечную точки алгоритма.

Пользователю будет доступно окно с различными настройками программы, окно с выводом текстовой информации о происходящем на экране (логгер) и окно с информацией о программе.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Предполагается следующий план разработки:

- Сдача вводного задания: до 05.07.2023
- Разработка плана и спец-ии: до 05.07.2023
- Разработка прототипа: до 05.07.2023
- Разработка 1-ой версии алгоритма: до 07.07.2023
- Реализация вывода графа на экран: до 07.07.2023
- Реализация системы вывода сообщений: до 10.07.2023
- Реализация вывода шагов алгоритма на экран: до 10.07.2023
- Реализация первой версии интерфейса: до 10.07.2023
- Реализация сохранения и загрузки графа: до 10.07.2023
- Реализация логики работы окна настроек: до 12.07.2023
- Исправление возникших проблем: до 13.07.2023

2.2. Распределение ролей в бригаде

Сулименко М. А.:

- Проектирование и реализация графического интерфейса пользователя

Таргонский М. А.

- Реализация алгоритма Дейкстры

Клепнев Д. А.:

- Реализация хранения ориентированного и неориентированного типов графа

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

3.1.1. Граф

Структура графа не представляет из себя ничего сложного. Это объект, который хранит в себе в отдельных списках вершины и ребра. Вершиной будет считаться объект, хранящий в себе только свои данные, а ребром – объект, хранящий данные и начальную и конечную вершины (в случае неориентированного графа эти вершины равнозначны). Граф предоставляет ряд методов по управлению им (см. п. 3.2.1).

3.1.2. Отображение графа

Для отображения графа требуются:

- Вершины. Их отображение реализовано через класс, который наследуется от класса круга. Дополнительно наш класс может хранить в себе ссылки на отображения своих данных и на текстовое поле ввода, которое используется при создании вершины пользователем.
- Ребра. Они бывают двух видов: прямые и кривые. Оба вида реализуются тем же способом, что и вершины, только в случае прямых мы наследуемся от прямой, а в случае кривой – от параметрической кривой с одной контрольной точкой. Также, если требуются стрелки, то они будут хранится здесь.
- Стрелки. Этот объект создан, как две наклонные линии по 45 градусов, соединенные таким образом, чтобы они образовывали стрелку, смотрящую направо. В дальнейшем с помощью функции atan2 ребра разворачивают стрелку так, как им надо.
- Подписи. Это просто класс, наследующийся от класса текста. Если в нем создать слишком большую надпись, то он создаст для себя ToolTip, который будет показываться при наведении.

Кроме того, все вышеописанные классы реализуют интерфейс, позволяющий более просто настраивать стили. Для всех классов реализация этого интерфейса одинаковая, т.ч. был создан класс StyleEngine, который и реализует нужные методы.

Для отображения всех вышеописанных элементов существует класс GraphPane, который наследуется от Pane и имплементирует интерфейс для стилей (StyleEngine в нем также хранится). После создания, элементы графа добавляются на GraphPane, как его дети. Также GraphPane хранит в себе сам граф и хранит все установленные на него вершины и ребра, соотнося их с реальными элементами графа.

Для редактирования графа существует класс GraphEditor, который работает с GraphPane и может манипулировать объектами в нем.

3.1.3. Сохранение графа

Для сохранения графа использовалась сторонняя библиотека GSON, т.ч. для сохранения требовалось только создать промежуточный класс для сохранения/загрузки графа и класс, который будет работать с этим промежуточным форматом графа и библиотекой.

3.1.4. Алгоритм и разбиение на шаги

Для работы с алгоритмом был создан класс Dijkstra, который хранит очередь с приоритетом для самого алгоритма, найденный путь и лист для хранения данных выделенных шагов в алгоритме.

Для алгоритма пришлось переопределить понятие вершины, чтобы она сохраняла своего предка. Также она сохраняет объект оригинальной вершины, из которой была образована.

Для хранения данных шагов также был создан класс. Он хранит тип шага, вершину из которой шаг начинался, вершину с которой шаг в итоге работал и вес пути.

3.1.5. Интерпретатор шагов

Интерпретатор шагов представляет из себя класс, который хранит в себе объект класса Dijkstra, может его вызвать и сформировать список действий по отображению созданных алгоритмом шагов. Эти действия интерпретатор хранит в специальном объекте, который предназначен для работы с двумя потоками. Также интерпретатор хранит множество списков с данными, которые использовали действия всех показанных шагов. Эти списки нужны для переключения шагов назад.

Исключительно для интерпретатора также был создан класс GraphPainter для графических взаимодействий с уже построенным графом.

3.1.6. Интерфейс

Интерфейс в программе нарисован с помощью программы Scene Builder и хранится в виде fxml файлов. Были нарисованы интерфейсы для окон настроек, логгера, окна about и основного окна приложения. Каждый интерфейс имеет свой контроллер, который хранит разные данные, зависящие от самого интерфейса. Но каждое из них хранит в себе некоторые объекты, расположенные на самом интерфейсе, чтобы в дальнейшем работать с ними.

Отдельного внимания требует логгер, т.к. ему помимо контроллера понадобился класс, реализующий singleton паттерн. Он хранит в себе объект контроллера окна логгера и позволяет любому классу получить себя для вывода сообщений в логгер.

3.2. Основные методы

3.2.1. Граф

В структуре графа существует множество методов, задача которых позволить работать с самим графом и его элементами. Так, в нем есть методы, позволяющие:

- Получить все вершины
- Получить все ребра
- Получить все ребра, смежные с определенной вершиной
- Получить ребро, соединяющее одну вершину с другой
- Добавить вершину
- Добавить ребро

И так далее…

3.2.2. Отображение графа

Основные элементы отображения графа содержат методы по настройке их вида и методы получения и установки для тех или иных параметров, которые можно хранить в этих объектах. Так, вершине можно указать или получить ее радиус и объект вершины из графа. Ребру можно установить или получить стрелку и сам объект ребра графа. На оба этих объекта можно добавить тестовые подписи через соответствующие методы.

GraphPane содержит методы по загрузке готового графа, по очистке всего поля, по добавлению и удалению графических объектов и еще несколько вспомогательных методов.

У GraphEditor важно отметить метод setViewMode, который позволяет задать то, как программа будет вести себя при нажатии левой кнопкой мыши по GraphPane. Также он реализует метод очистки графа, который свои очищает внутренние данные и GraphPane.

3.2.3. Сохранение графа

Промежуточный класс графа имеет функцию получения нормального класса графа из своих данных. Класс, который переводит промежуточный класс в json и обратно, имеет соответствующие методы.

3.2.4. Алгоритм и разбиение на шаги

Класс Dijkstra имеет метод поиска кратчайшего пути, который выдает true или false в зависимости от того, нашел он путь или нет. Найденный путь можно получить отдельным методом. Также можно получить список выделенных алгоритмом шагов.

3.2.5. Интерпретатор шагов

Среди методов интерпретатора важно выделить метод applyDijkstra(), который выполняет поиск пути, забирает полученные шаги, а затем формирует список действий, требующихся для графического отображения каждого шага.

Также важны методы:

- Авто воспроизведения, когда 2 потока по очереди с определенной паузой выполняют шаги.
- Переход на шаг вперед, когда очередной шаг вручную выполняется.
- Переход назад, когда восстанавливается предыдущий шаг.

У graphPainter важен метод последовательной отрисовки пути до вершины, метод окрашивания вершины и метод окрашивания ребра (оба имеют разные параметры окраски).

3.2.6. Интерфейс

Интерфейс программы, как было сказано ранее, собран из 4 различных окон, каждое из которых имеет свой контроллер со своими методами.

Так, контроллер настроек имеет методы по добавлению в него данных и по отправке их в контроллер главного экрана. Контроллер логгера имеет метод, записывающий определенную информацию в логгер. Контроллер окна about имеет метод, позволяющий копировать текст в буфер обмена.

Самый важный контролер – это контроллер главного экрана. В нем расположены все управляющие методы всех кнопок интерфейса. В нем же

реализуется эффект увеличения и уменьшения. Через него работает окно настроек (метод применения настроек).

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Было произведено ручное тестирование на двух операционных системах (Linux и Windows) всех элементов интерфейса, а именно:

- Режима изменения графа
- Режима воспроизведения алгоритма Дейкстры
- Панели инструментов (меню File, Edit, Help)
- Инструментов сохранения и загрузки
- Окна настроек
- Окна логирования

Во время тестирования **режима изменения графа** была проверена корректная работоспособность каждого инструмента: перетаскивание вершин графа, создание вершины, создание ребра, выбор начальной и конечной вершины для Алгоритма Дейкстры, очистка графа. Инструменты работают исправно, конфликта не возникает. При выборе инструмента, который отвечает редактирование графа, графически подсвечивается какая кнопка была выбрана пользователем, при этом невозможно выбрать одновременно два режима, только один.

Во время тестирования **режима воспроизведения** также была проверена работоспособность каждого инструмента: следующий/предыдущий шаг, пауза, старт, стоп. Если пользователь не выбрал начальную и/или конченые вершину – инструменты по воспроизведению остаются недоступными для пользователя, аналогичный запрет распространяется на меню Edit в верхней панели инструментов. Были протестированы ситуации, когда пользователь, во время проигрывания алгоритма Дейкстры, начинает менять настройки или как-либо взаимодействовать с инструментами, которые ему доступны, никаких конфликтов не возникает, в случае переключения на режим редактирования графа или изменения каких-либо настроек – воспроизведение останавливается, пользователю необходимо заново его начать, так как он изменил какие-либо

настройки. В случае, когда настройки не были изменены (окно настроек было открыто, однако изменений не последовало) – алгоритм продолжает воспроизводиться. Открытие окна логирования никак не влияет на проигрывание, в нём наоборот отражается каждый шаг и что на нём произошло.

Каждое меню на **панели инструментов** вверху программы было протестировано на работоспособность. В любой момент времени пользователю доступны только необходимые инструменты. Например, если пользователь ещё не выбрал тип графа, с которым он хочет работать, ему не доступны никакие инструменты редактирования или воспроизведения. Окно настроек и окно логирования может быть открыто, даже если тип графа ещё не был выбран, однако никаких конфликтных ситуаций не будет, изменение любых параметров будет учитываться в созданном графе.

Инструменты сохранения и загрузки работают корректно. При первом сохранении графа (по сути, безымянного) с помощью инструмента “Save” – будет открыт проводник, такой же, как при нажатии на “Save as”. Последующее использование инструмента “Save” будет производить сохранение без вызова проводника. Расширение у сохраняемых файлов будет .json. Инструмент загрузки файлов будет загружать только целые файлы формата .json, то есть, если файл был повреждён каким-либо образом – приложение не сможет его загрузить, конфликта не возникнет.

Окно настроек тестировалось многократно, таким образом, в любой момент времени пользователь может безопасно изменить необходимый ему параметр, что не вызовет никаких конфликтов. Окно настроек не может быть одновременно открыто дважды, если пользователь решил выбрать инструмент открытия окна настроек при уже запущенном окне – фокус просто перейдёт на уже открытое окно.

Окно логирования не доступно для редактирования пользователем, что было протестировано. Любое действие пользователя записывается в окно логирования. Аналогично окну настроек – пользователь не сможет открыть

несколько окон логирования, при возникновении такой ситуации фокус просто перейдёт на уже открытое окно логирования.

4.2. Тестирования кода графа

Тестирования реализации графа подразумевает под собой создание двух групп автоматического тестирования для каждого вида графа, т.к. реализация некоторых функций у разных типов разная.

Для тестирования использовалась библиотека JUnit, которая сильно упростила этот процесс. Удобной возможностью оказалась функция, которая вызывается перед каждым тестом, в которой производился сброс графа.

Каждая функция в обоих видах тестировалась минимум тремя тестами: тестом на нормальную работу, на получение ошибки при передаче null значения или на получения ошибки при других обстоятельствах.

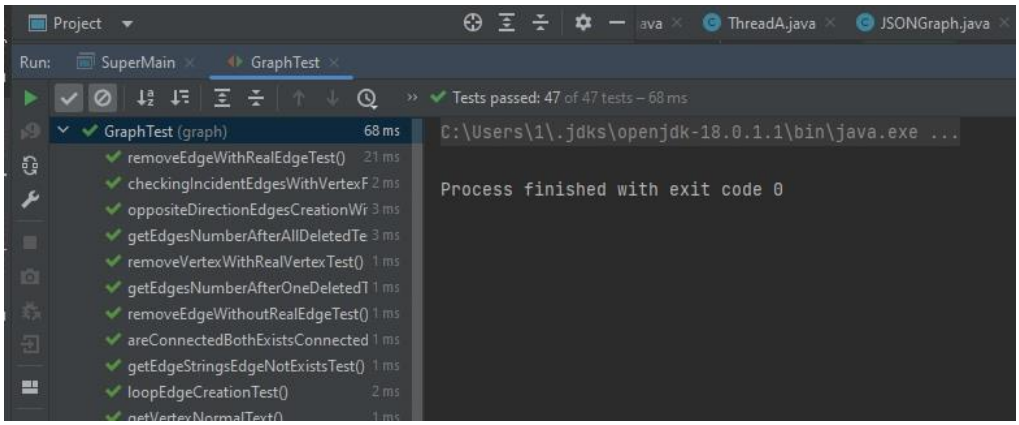


Рисунок 20 – Тесты для неориентированного графа

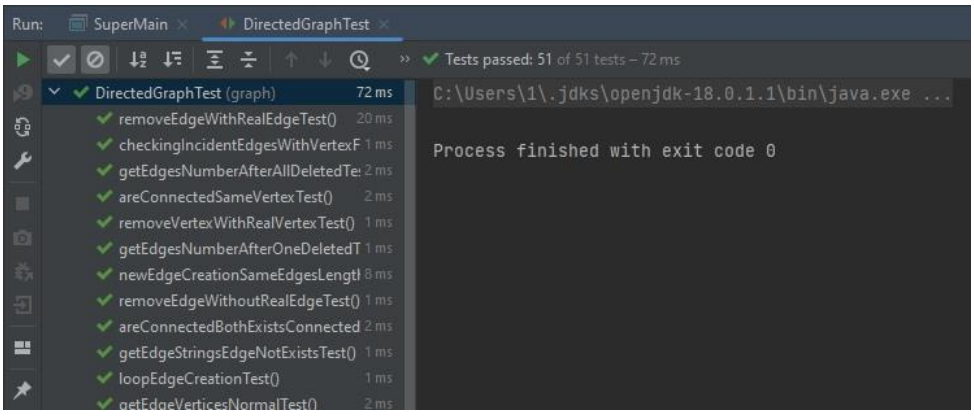


Рисунок 21 – Тесты для ориентированного графа

4.3. Тестирование кода алгоритма

Для тестирования кода алгоритма были созданы несколько автоматических тестов с помощью библиотеки JUnit. С её помощью

тестировался сам алгоритм на ориентированном и неориентированном графе, а также ситуации, когда путь не существует, либо же когда существуют несколько кратчайших путей. Сами тесты подбирались, чтобы охватывать все описанные случаи.

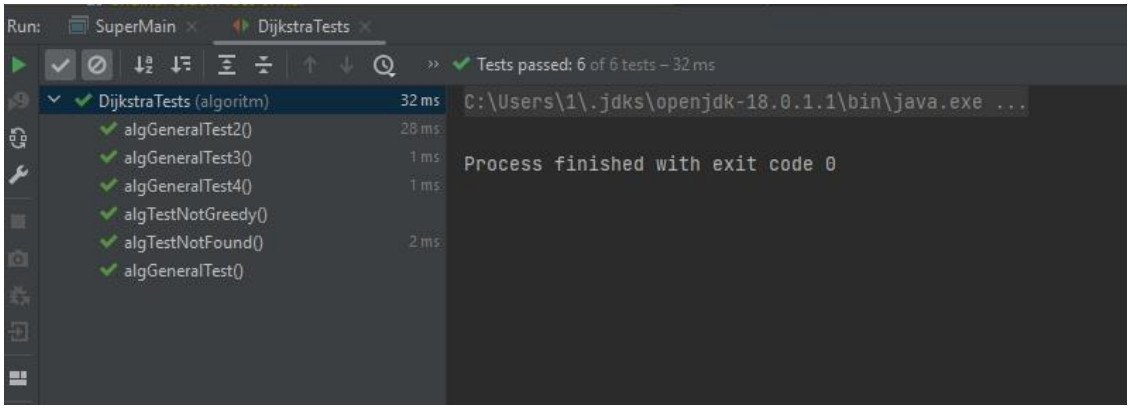


Рисунок 22 – Тестирование алгоритма на неориентированном графе

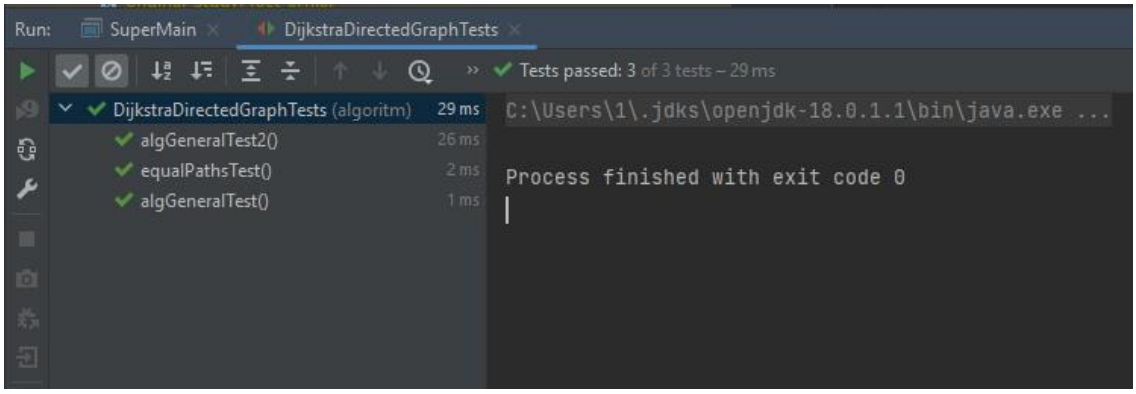


Рисунок 23 – Тестирование алгоритма на ориентированном графе

4.4. Тестирование кода сохранения графа

Для тестирования кода сохранения графа были созданы несколько автоматических тестов с помощью библиотеки JUnit. С её помощью тестировалась как загрузка, так и записи графа в текстовый файл формата JSON, причём сохранение тестировалось в разных директориях, как относительно запущенной программы, так и абсолютно начиная с дисков, а так-же ситуации, если невозможно открыть файл. Сами тесты были подобраны таким образом, чтобы охватить все описанные случаи, причём над ними была проведена дополнительная настройка, чтобы порядок выполнения тестов соответствовал их порядку в коде, так как некоторые из них взаимосвязаны. Например, сначала идут тесты сохранения графов в файлы в разных директориях, после чего идут тесты на загрузку графов из сохранённых ранее директорий. К тому же, после

выполнения тестов все созданные в тестах файлы с графами автоматически удаляются к компьютера.

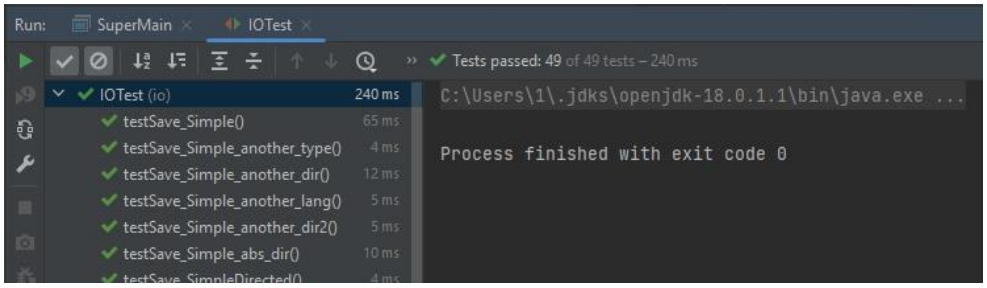


Рисунок 24 – Тесты сохранения графов разлиных типов

ЗАКЛЮЧЕНИЕ

Подводя итоги проделанной работы, выполнение всех поставленных задач было выполнено в короткие сроки с минимальным количеством проблем. Некоторые концепции и способы реализации были подкорректированы в процессе разработки, опираясь на реальность их реализации за поставленное время. Тем не менее, в конечном итоге получился качественный продукт.

Конечно, в программе могло остаться некоторое количество багов, которые не были обнаружены при тестировании. На их поиск и исправление требуется дополнительное время, которое нельзя уместить в заложенные сроки практики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Форум программистов, где можно получить помощь по любому интересующему вопросу // Stack overflow. URL: https://stackoverflow.com
2. Документация по JavaFX // Oracle: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm
3. Курс обучения Java // Stepik: https://stepik.org/course/187/promo
4. Сайт с бесплатными иконками для приложений // Flaticon: https://www.flaticon.com
5. Статья по работе с JavaFX // Habr: https://habr.com/ru/post/474292/
6. Репозиторий бригады: https://github.com/mishatar/StudyPractice