

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

ОТЧЕТ
по учебной практике
Тема: Визуализация алгоритмов на языке Kotlin

Студент гр. 8381	_____	Сосновский Д.Н.
Студент гр. 8381	_____	Киреев К.А.
Студент гр. 8381	_____	Муковский Д.В.
Руководитель	_____	Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Сосновский Д.Н. группы 8381

Студент Киреев К.А. группы 8381

Студент Муковский Д.В. группы 8381

Тема практики: визуализация алгоритмов на языке Kotlin

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Kotlin с графическим интерфейсом.

Алгоритм: Дейкстры.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 10.07.2020

Дата защиты отчета: 10.07.2020

Студент	_____	Сосновский Д.Н.
Студент	_____	Киреев К.А.
Студент	_____	Муковский Д.В.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

В ходе выполнения задания учебной практики была реализована программа, предназначенная для нахождения кратчайшего пути в графе. Поиск кратчайшего пути осуществляется с использованием алгоритма Дейкстры. Разработанная программа детально показывает этапы работы алгоритма при построении кратчайшего пути. Программа была написана на языке программирования Kotlin, в среде разработки IntelliJ Idea.

SUMMARY

In the course of the assignment, a program was developed designed to find the shortest paths in a graph. Search using the Dijkstra algorithm. The developed program shows in detail the stages of the algorithm when building the shortest path. Kotlin, in the IntelliJ Idea development environment.

СОДЕРЖАНИЕ

1.	Постановка задачи	6
1.1.	Формулировка задания	6
1.2.	Формальное определение и сложность алгоритма	6
1.3.	Реализуемый алгоритм	6
1.4	Псевдокод	7
2.	Спецификация	8
2.1.	Требования к интерфейсу пользователя	8
2.2.	Требования к вводу исходных данных	10
2.3.	Требования к визуализации	10
2.4.	Уточнение требований после сдачи первой версии	11
2.5.	Уточнение требований после сдачи второй версии	11
3.	План разработки и распределение ролей в бригаде	12
3.1.	План разработки	12
3.2.	Распределение ролей в бригаде	13
4.	Особенности реализации	14
4.1	Архитектура программы	14
4.2	Структуры данных	14
4.3	Основные методы	15
5.	Тестирование	20
5.1.	Тестирование графического интерфейса	20
5.2.	Тестирование алгоритма	25
5.3.	Unit тестирование	30
	Заключение	34
	Список использованных источников	35
	Исходный код	36

ВВЕДЕНИЕ

Разработка программы будет вестись с использованием языка программирования Kotlin и его возможностей по созданию GUI. Для написания программы используется интегрированная среда разработки IntelliJ IDEA.

Цели выполнения учебной практики:

1. Освоение среды программирования Kotlin, особенностей и синтаксиса данного языка.
2. Изучение средств для реализации графического интерфейса, а именно – библиотеки Swing.
3. Получение таких навыков как:
 - разработка программ на языке Kotlin;
 - работа с системой контроля версий, репозиториями;
 - командная работа.

Поставленные цели будут достигнуты во время выполнения проекта (визуализации алгоритма Дейкстры), выбранного бригадой.

Алгоритм Дейкстры - алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Данный алгоритм находит кратчайшие пути от одной из вершин графа до всех остальных вершин. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например, при планировании автомобильных и авиамаршрутов, при разводке электронных плат, в протоколах маршрутизации.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Формулировка задания

Разработать программу, которая визуализирует алгоритм Дейкстры для поиска кратчайших путей в графе на языке программирования Kotlin.

1.2. Формальное определение и сложность алгоритма

Дан взвешенный ориентированный граф $G(V, E)$ без дуг отрицательного веса. Найти кратчайшие пути от некоторой вершины a графа G до всех остальных вершин этого графа.

Сложность алгоритма Дейкстры зависит от способа нахождения вершины v , а также способа хранения множества непосещённых вершин и способа обновления меток. В простейшем случае, когда для поиска вершины с минимальным $d[v]$ просматривается всё множество вершин, а для хранения величин d используется массив, время работы алгоритма есть $O(n^2)$.

1.3. Реализуемый алгоритм

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a . Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация.

Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Шаг алгоритма.

Если все вершины посещены, алгоритм завершается.

В противном случае, из ещё не посещённых вершин выбирается вершина u , имеющая минимальную метку.

Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u , назовём

соседями этой вершины. Для каждого соседа вершины u , кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.

1.4. Псевдокод

```
func dijkstra(s):
  for  $v \in V$ 
     $d[v] = \infty$ 
     $used[v] = false$ 
   $d[s] = 0$ 
  for  $i \in V$ 
     $v = null$ 
    for  $j \in V$  // найдём вершину с минимальным расстоянием
      if ! $used[j]$  and ( $v == null$  or  $d[j] < d[v]$ )
         $v = j$ 
    if  $d[v] == \infty$ 
      break
     $used[v] = true$ 
    for  $e$  : исходящие из  $v$  рёбра // произведём релаксацию по всем рёбрам, исходящим из  $v$ 
      if  $d[v] + e.len < d[e.to]$ 
         $d[e.to] = d[v] + e.len$ 
```

2. СПЕЦИФИКАЦИЯ

2.1. Требования к интерфейсу пользователя

Пользовательский интерфейс должен представлять собой диалоговое окно, содержащее набор кнопок, предназначенных для управления состоянием программы.

Диалоговое окно должно состоять из:

- Рабочей области для построения графа.
- Кнопки «Сохранить», которая должна позволять пользователю сохранить созданный в рабочей области граф в виде .txt файла.
- Кнопки «Загрузить», которая должна позволять пользователю загрузить граф, для которого необходимо применить алгоритм Дейкстры, в виде .txt файла.
- Кнопки «Сгенерировать граф», которая должна генерировать случайный граф в рабочей области.
- Кнопки «Вперед», которая должна отобразить следующую итерацию алгоритма.
- Кнопки «Назад», которая должна отобразить предыдущую итерацию алгоритма

Первоначальный макет представлен на рис. 1.1

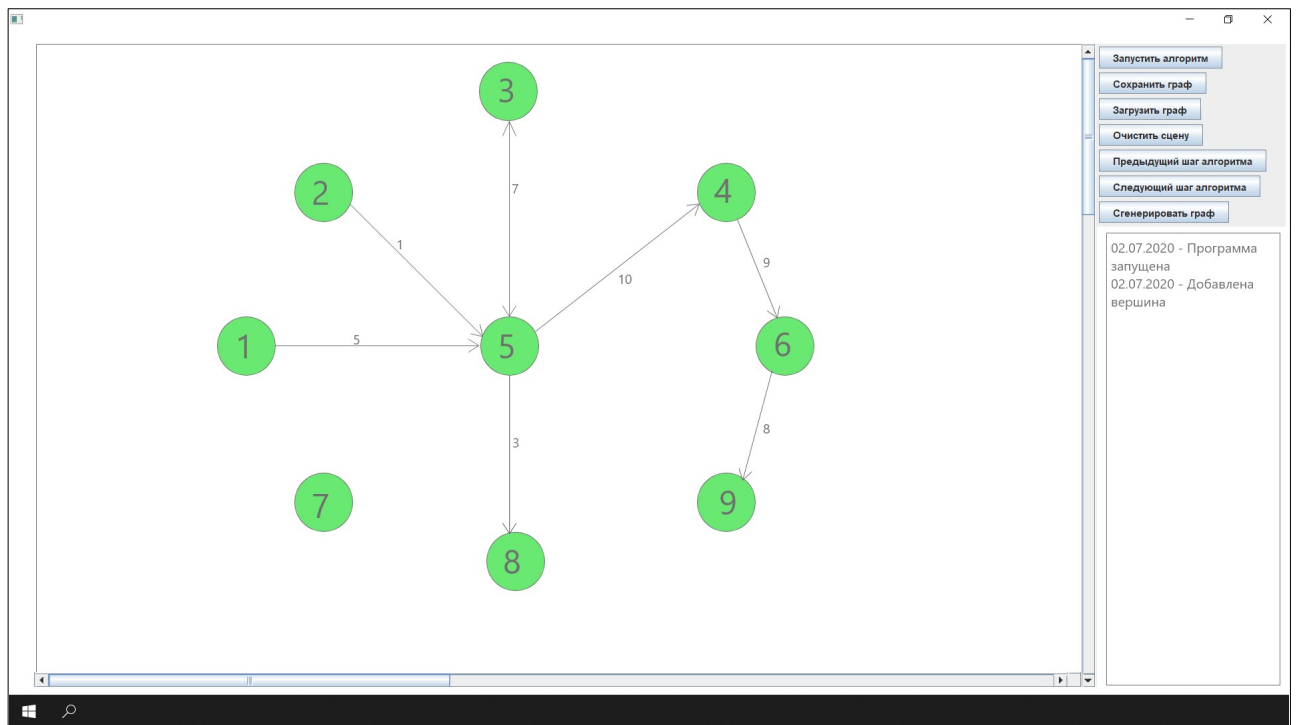


Рисунок 1.1 — Макет программы

Также представлена use case диаграмма на рисунке 1.2.

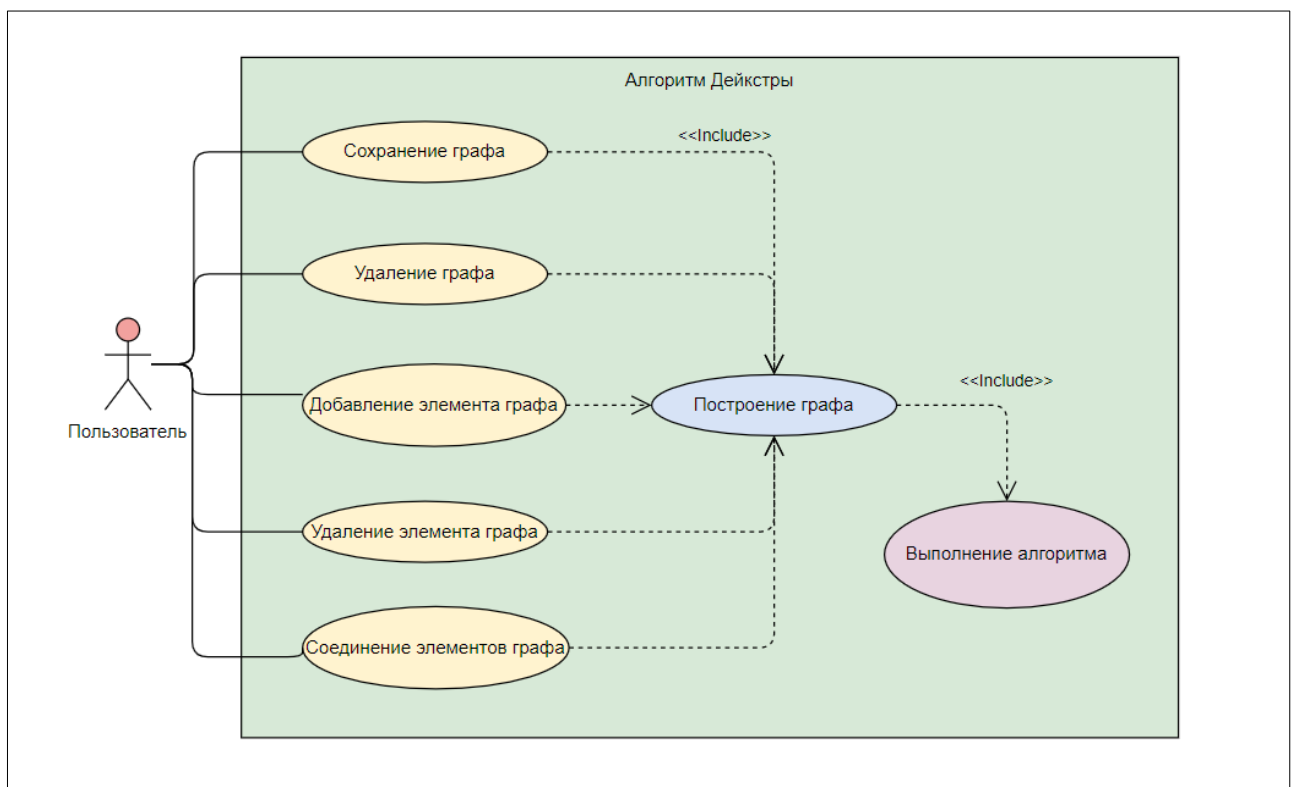


Рисунок 1.2 - Use case диаграмма

2.2. Требования к вводу исходных данных

На вход алгоритму должен подаваться взвешенный ориентированный граф. Именем вершины могут быть числа. Область создания графа предоставляет возможность ввести данные вручную, с файла или сгенерировать. При ручном вводе происходит взаимодействие с графическим представлением графа с помощью мыши. Двойным кликом по области создается вершина с указанным именем, а через контекстное меню вершины можно будет удалить ее или провести ребро к другой вершине. При считывании с файла надо нажать соответствующую кнопку. Формат данных для считывания из файла: на первой строке находятся координаты вершин а их именем является порядок, в котором они расположены в файле и далее на второй строке список попарных вершин с их весами, обозначающих ребра. При генерации нужно ввести необходимые данные в поля генерации.

2.3. Требования к визуализации

Перед началом работы алгоритма создается граф, одним из указанных выше способов. Также уже существующие вершины можно перемещать. При выделении вершины ее цвет меняется на специальный цвет. На каждом шаге будут выделяться все вершины, с которыми сейчас происходит действие алгоритма в специальный отличный цвет, будет выводиться дополнительный информационный текст, о том что на этом шаге происходит, с какими вершинами и ребрами идет взаимодействие, а также будут подсвечиваться используемые ребра. В конце появляется всплывающее окно с результатами работы алгоритма.

2.4. Уточнение требований после сдачи первой версии

- Конечную вершину указывать не надо: алгоритм Дейкстры находит расстояния до всех вершин от стартовой.
- После завершения алгоритма должна быть возможность отредактировать граф и запустить алгоритм заново (возможно, с другой стартовой вершиной).

2.5. Уточнение требований после сдачи второй версии

- Выделение цветом не только текущей вершины, но и тех вершин, расстояния до которых уже определены.
- Вместо "Текущий узел: x; (He) произошла релаксация; Лучший путь до узла: z; Предыдущий узел: y " выводите: " Ребро: (y, x); (He) произошла релаксация; Лучшее расст. до узла: z (было p) "
- Возможность пользователя редактировать текстовое поле выводимых пояснений.
- Выводить пояснение, почему алгоритм закончился (почему следующий шаг невозможен).
- Если стартовая вершина не имеет рёбер, то всегда должно выводиться "Данная вершина не имеет ребер"; если имеет - то алгоритм должен запускаться. (Это условие иногда нарушается, это надо исправить.)

3. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

3.1. План разработки

Репозиторий с исходным кодом и данной спецификацией расположен на GitHub по адресу: https://github.com/DmitriySosnovskiy/kotlin_training_practice

План разработки программы разбит на следующие шаги:

1. Создание прототипа (03.07): к этому шагу будет создано приложение, демонстрирующее интерфейс, но почти не реализующее основные функции;

2. Первая версия (06.07): на этом шаге будет реализован класс, отвечающий за работу самого алгоритма Дейкстры. Затем он будет соединен с классом GUI, используя интерфейсы обоих классов; в результате получится начальная рабочая версия программы; реализовано взаимодействие с вершинами для ручного ввода; написание тестов к проекту;

Перед релизом первой версии планируется тестирование основных функций работы с GUI (корректная работа основных кнопок, которые планируются к данной версии (запустить алгоритм, очистить сцену)), тестирование алгоритма Дейкстры на данных с ручного ввода (некоторые примеры описаны в плане тестирования).

3. Вторая версия (08.07): реализация пошаговой работы программы. реализация выделения вершин при исполнении шагов алгоритма; реализован ввод из файла; реализован вывод логов;

Перед релизом второй версии планируется тестирование дополнительных функций работы с GUI (корректная работа кнопок сохранить, загрузить, шаги), тестирование корректного вывода логов.

4. Третья (конечная) версия (10.07): добавление возможности генерации входных данных. Реализация вывода дополнительного информационного текста при шагах алгоритма, а также выделение ребер. Исправление ошибок проекта. Тестирование проекта.

Перед релизом финальной версии планируется тестирование корректной генерации входных данных, тестирование окна с информацией, выводимого на шагах алгоритма, тестирование конечного варианта GUI, финальное тестирование алгоритма Дейкстры, используя JUnit 5.

3.2. Распределение ролей в бригаде

Роли в бригаде распределены следующим образом:

- Сосновский Дмитрий: реализация GUI и методов взаимодействия с ним;
- Киреев Константин: реализации алгоритма Дейкстры и методов взаимодействия с ним; тестирование приложения; написание отчета.
- Муковский Даниил: реализация архитектуры программы с использованием паттерна MVP (model view presenter); реализация части presenter и model; проектирование системы классов и их взаимодействия;

4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

4.1. Архитектура программы

- Архитектура данного приложения построена на основе шаблона проектирования MVP (Model View Presenter). Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель. Таким образом, выделив 3 интерфейса (views, presenters, models) разрабатывать данное приложение можно независимо в трех «плоскостях».
- Для реализации пошагового вывода визуализации работы алгоритма был использован шаблон проектирования Снимок. Был создан класс DijkstraAlgorithmController, который хранит в себе информацию о текущем шаге. В определенные моменты алгоритма в контейнер закидываются снимки текущего состояния алгоритма. Затем, когда необходимо сделать шаг вперед просто отображается следующий снимок.

4.2. Структуры данных

- Класс Edge, который хранит информацию о ребре графа.
Поля:
v1: Int – первая вершина;
v2: Int – вторая вершина;
dist: Int – вес ребра;
- Класс Vertex, который хранит информацию о вершине
Поля:
name: Int – имя вершины;
dist: Int – дистанция от начальной вершины;

previous: Vertex? – вершина, из которой перешли в текущую;
neighbours: HashMap<Vertex, Int> – хэш-таблица для хранения пар соседей вершины;

Методы:

printPath(): String - рекурсивный вывод кратчайшего пути;

compareTo(other: Vertex): Int – компаратор для treeSet;

- Класс Graph(private val edges: List<Edges>), который хранит информацию о графе, граф представлен в виде списка ребер.

Поля:

graph: HashMap<Int, Vertex> – хэш-таблица вершин, построенная из списка ребер;

snapshotKeeper: SnapshotKeeper – снимок текущего шага;

Методы:

dijkstra(startname: Int) – запускает алгоритм Дейкстры, используя указанную исходную вершину; устанавливает характеристики графа; заполняет treeSet;

private fun dijkstra(treeSet: TreeSet<Vertex>) - запускает алгоритм Дейкстры для бинарной кучи вершин;

private fun printPath(endName: Int, startName: Int): String – печатает все пути от начальной вершины;

private fun makeSnapshot(vertex: Int, prev: Int, relax: Boolean) – создание снимка текущего шага;

fun getSnapshotHistory(): SnapshotKeeper - все снимки;

fun getPath(startVertex: Int): String — форматированный вывод ответа;

4.3. Основные методы

- Класс MainPresenter, который является основным для реализации presenter;

private fun printLogs(log: String) — печатает логи;

fun onAlgorithmEndConfirmed() - подтверждает конец алгоритма;

`override fun handleEvent(event: Event)` — обрабатывает текущее событие;
`private fun requestGeneratingGraphParameters()` - обрабатывает параметры генерации графа;
`private fun clearScene()` - очистка сцены;
`private fun resetNodesAndEdges()` - обнуление вершин и ребер;
`private fun requestStartNodeNumber(): Int?` - обрабатывает стартовую вершины;
`private fun resetAll()` - сбрасывает все;
`fun addNode(new: UINode)` — добавляет узел;
`fun addEdge(new: UIEdge)` — добавляет ребро;
`fun deleteEdge(deleted: UIEdge)` — удаляет ребро;
`fun deleteNode(deleted: UINode)` — удаляет узел;
`private fun isNodeHaveEdges(startNode: Int): Boolean` — проверяет связана ли вершина;
`private fun isAllNodesConnected(): String` - проверяет связаны ли все вершины;
`private fun startAlgorithm(startNode: Int)` — запускает алгоритм;
`private fun updateAllNodes(snapMap: HashMap<Int, List<String>>)` – обновляет все узлы;
`private fun getLogs(snapMap: HashMap<Int, List<String>>): String` - получает логов;
`private fun nextStep()` - следующий шаг алгоритма;
`private fun previousStep()` - предыдущий шаг алгоритма;
`private fun finishAlgorithm()` - заканчивает алгоритм;
`private fun downloadGraph(fileName: String)` — загрузка графа из файла;
`private fun saveGraph(fileName: String)` — сохранение графа в файл;
`private fun generateGraph()` - генерация графа;

- Класс `GraphSheet`, который реализует графическую интерпретацию алгоритма и графа;
`override fun update() = repaint()` - обновление сцены;

`override fun paintComponent(graphics: Graphics)` — отрисовка компонента графа;

`private fun drawNode(node: UINode, nodeNumber: Int, panelGraphics: Graphics2D)` — рисование узла;

`private fun drawEdge(edge: UIEdge, panelGraphics: Graphics2D)` — рисование ребра;

`private fun drawEdgeArrow(arrow: UIEdgeArrow, panelGraphics: Graphics2D)` — рисование стрелочки ребра;

`private fun drawDualEdge(dualEdge: UIDualEdge, panelGraphics: Graphics2D)` — рисование двойного ребра;

`private fun drawBuildingEdge(buildingEdge: UIBuildingEdge, panelGraphics: Graphics2D)` — рисование ребра;

`override fun mouseReleased(mouseEvent: MouseEvent)` — событие, когда кнопка мыши отжата;

`override fun mouseClicked(mouseEvent: MouseEvent)` — событие по клику мыши;

`override fun mousePressed(mouseEvent: MouseEvent)` — событие по нажатию мыши;

`override fun mouseMoved(mouseEvent: MouseEvent)` — событие по передвижению мыши;

`override fun mouseDragged(mouseEvent: MouseEvent)` — событие по перетаскиванию мыши;

`private fun onRightMouseButtonDragging(dragCoordinate: Coordinate)` — событие по нажатию ПКМ;

`private fun onLeftMouseButtonDragging(dragCoordinate: Coordinate)` — событие по нажатию ЛКМ;

`private fun findNodeUnderMouse(cursorCoordinate: Coordinate): UINode?` — нахождение узла под курсором мыши;

`private fun findEdgeUnderMouse(cursorCoordinate: Coordinate): UIEdge?` — нахождение ребра под курсором мыши;

private fun addEdge(sourceNode: UINode, endNode: UINode, edgeWeight: String) — добавление ребра;

private fun updateCreatingEdge(newEndCoordinate: Coordinate) — обновление созданного ребра;

private fun createAndShowPopupMenuOnNode(sourceMouseEvent: MouseEvent, affectedNode: UINode) — всплывающее меню для вершины;

private fun createAndShowPopupMenuOnEdge(sourceMouseEvent: MouseEvent, affectedEdge: UIEdge) — всплывающее меню для ребра;

override fun displayDijkstraAlgorithmResult(result: String) — результат алгоритма Дейкстры;

На рисунке 2 представлена UML диаграмма проекта.

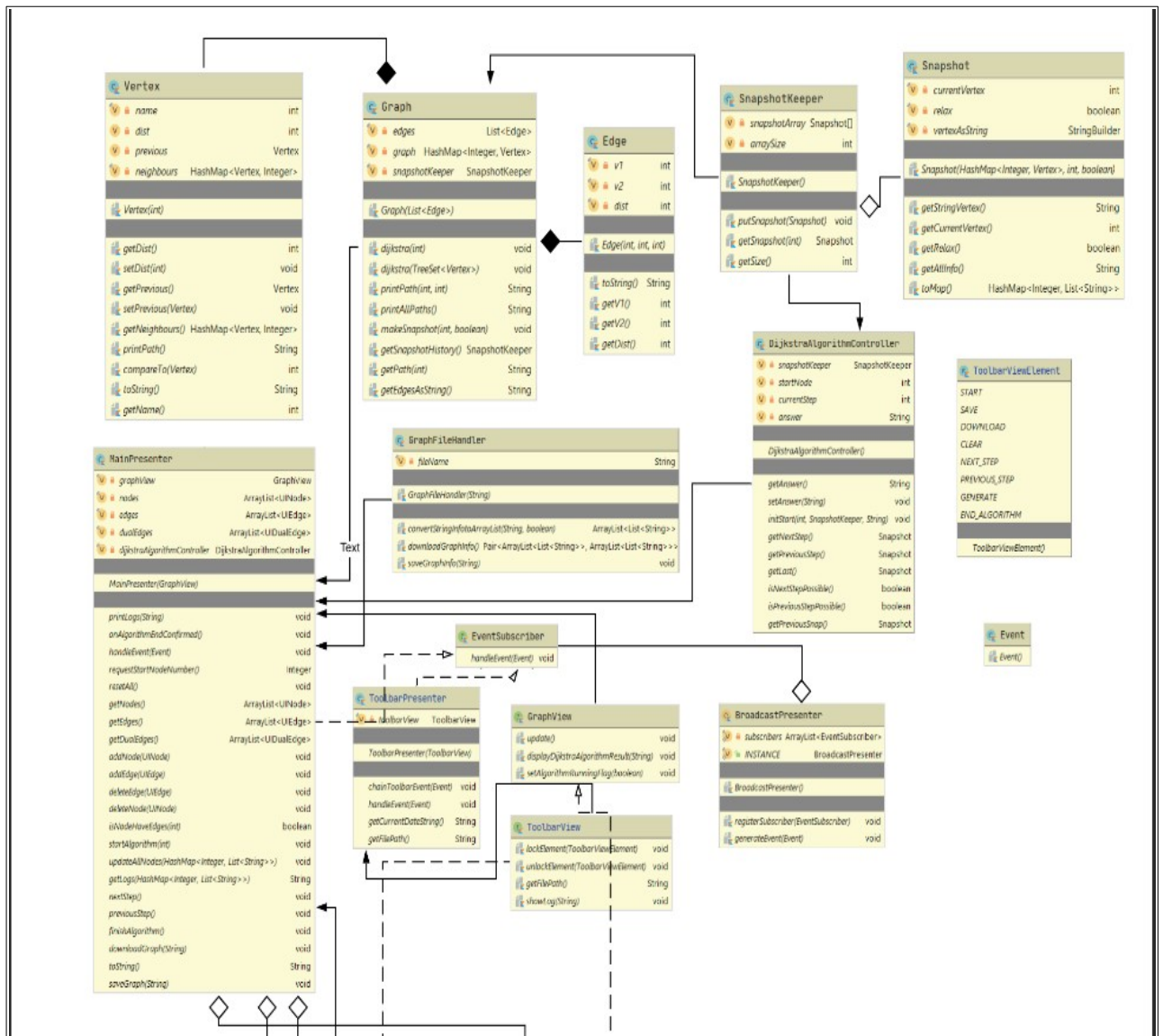


Рисунок 2.1 — UML диаграммы

5. ТЕСТИРОВАНИЕ

5.1. Тестирование графического интерфейса

- Для тестирования используется jar файл.
1. Добавление n вершин, двойным нажатием на холсте. На экране по порядку нажатия появляются вершины с номерами от 1 до n. Вершины можно передвигать для удобства восприятия.

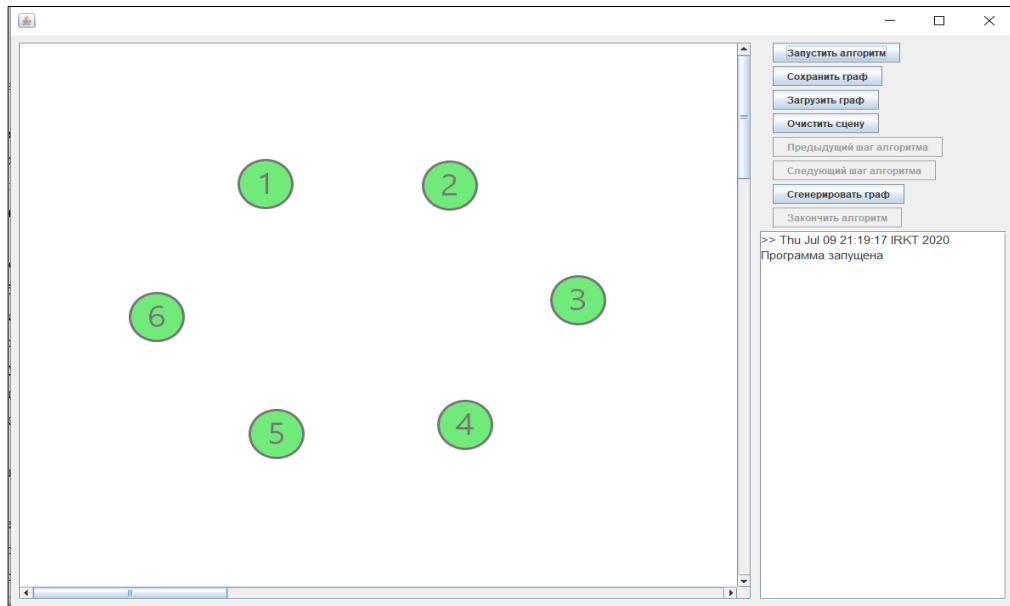


Рисунок 3 - Рисунок тестирования 1

2. Добавление ребер. При нажатии на вершину в контекстном меню задается вес ребра. Далее проводится ребро до второй вершины. На холсте появляются ребра с их весами в центре.

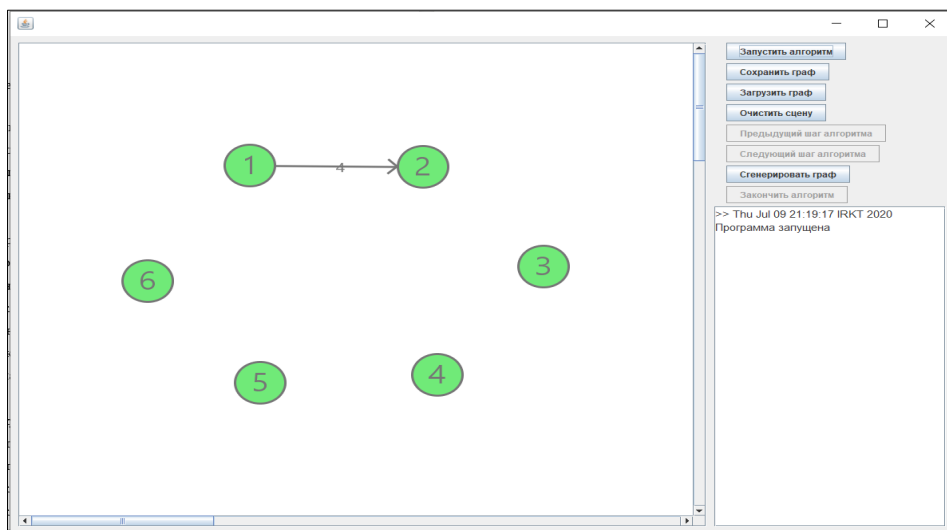


Рисунок 4 - Рисунок промежуточного результата тестирования 2

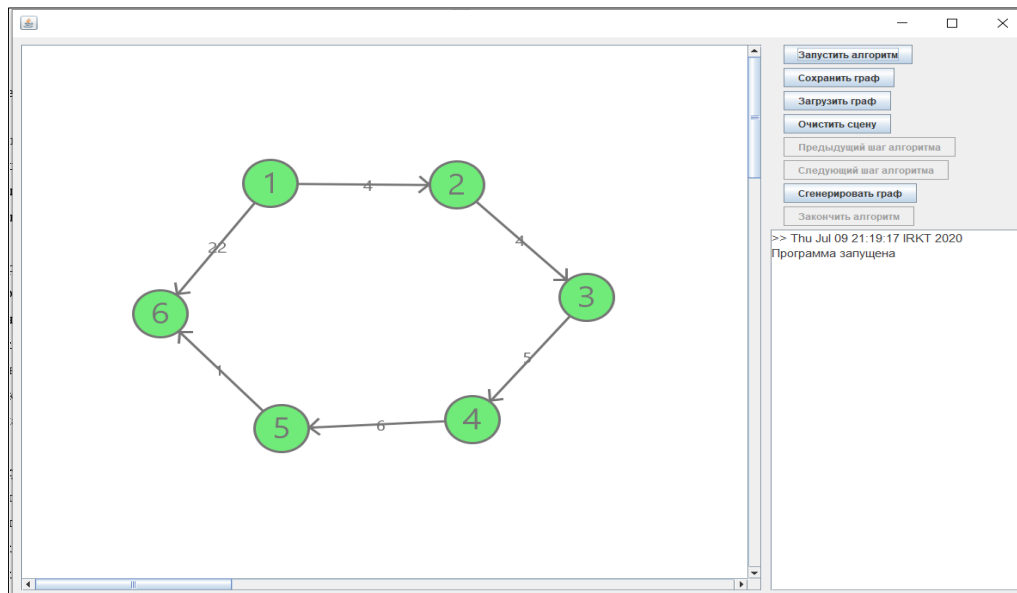


Рисунок 5 - Рисунок конечного результата тестирования 2

3. Нажатие на кнопку «Запустить алгоритм», которая запускает алгоритм Дейкстры. Далее нажатие кнопки «Закончить алгоритм».

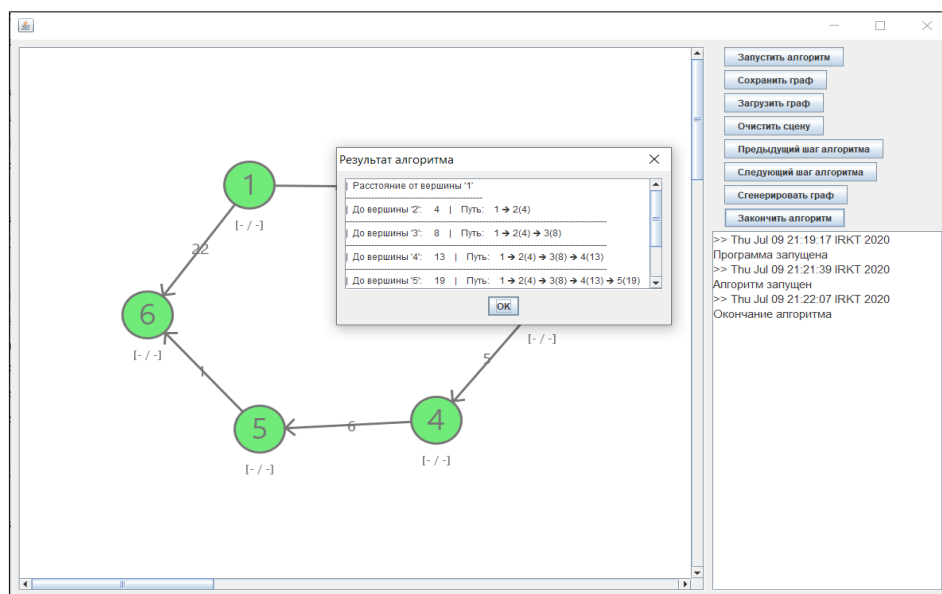


Рисунок 6 - Рисунок тестирования 3

4. Нажатие на кнопку «Следующий шаг алгоритма» после запуска алгоритма на графической сцене. Подсвечивается текущая вершина работы алгоритма. Нажатие происходит пока алгоритм не закончит свою работу. Далее в логах выводится сообщение, что следующий шаг невозможен

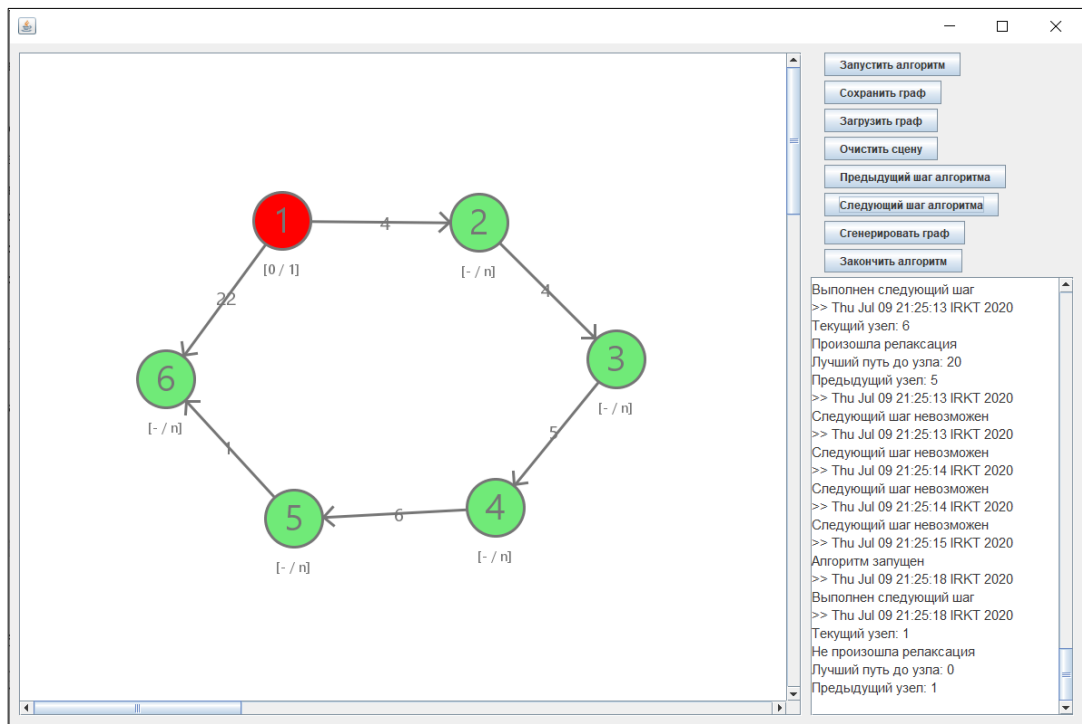


Рисунок 7 - Рисунок промежуточного результата тестирования 4

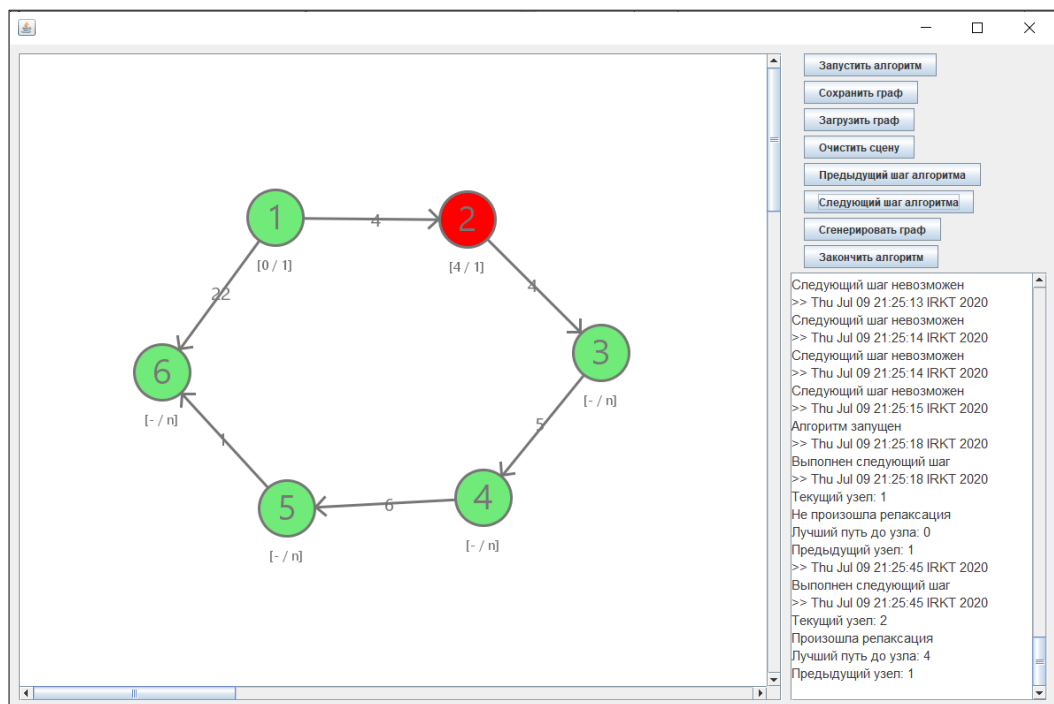


Рисунок 8 - Рисунок промежуточного результата тестирования 4

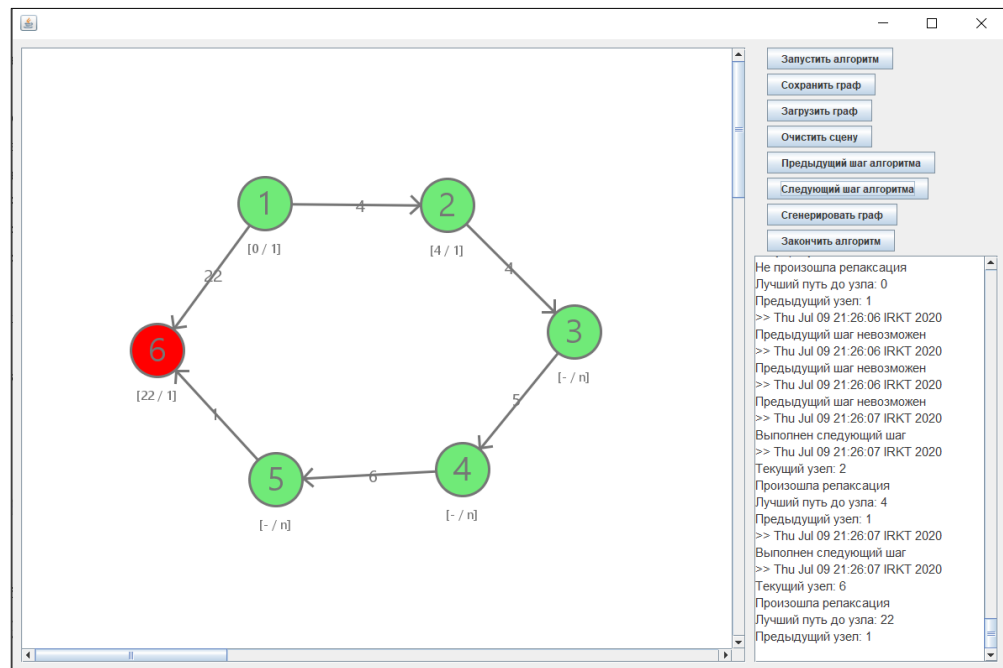


Рисунок 9 - Рисунок промежуточного результата тестирования 4
Алгоритм закончил работу, вывод соответствующего сообщения в всплывающем окне по кнопке «Закончить алгоритм».

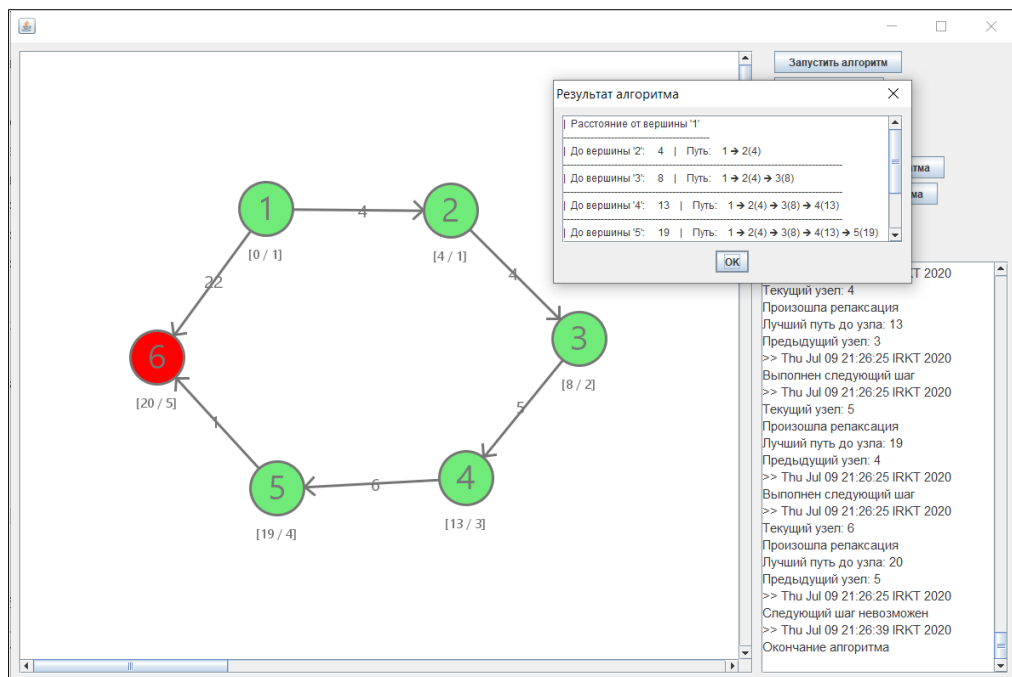


Рисунок 10 - Рисунок конечного результата тестирования 4
5. Нажатие на кнопку «Очистить сцену». Текущий граф исчезает и сцена становится пустой.

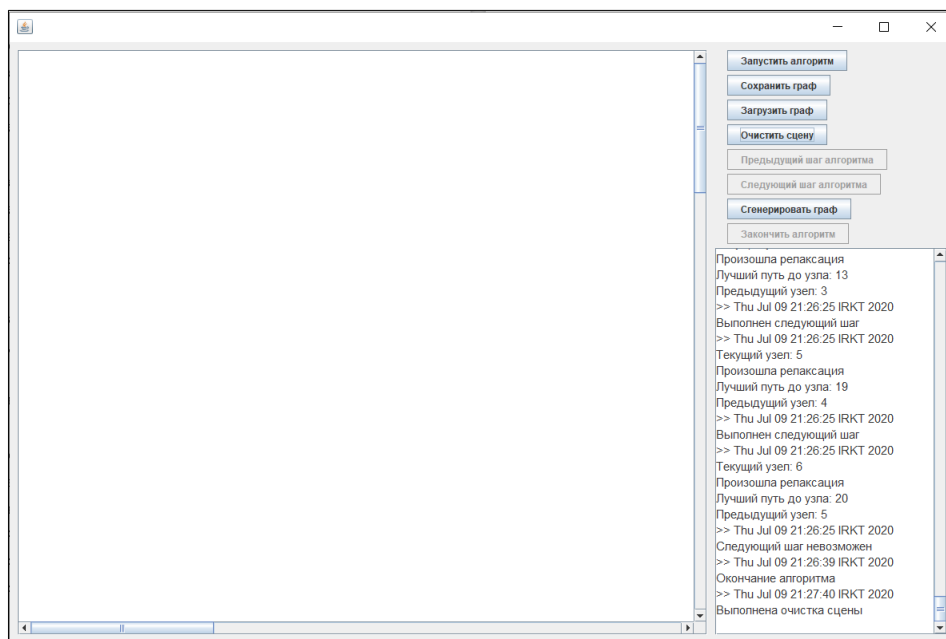


Рисунок 11 - Рисунок тестирования 5

6. Нажатие на кнопку «Загрузить граф». Далее из выбранного файла загружается граф и выводится на сцену.

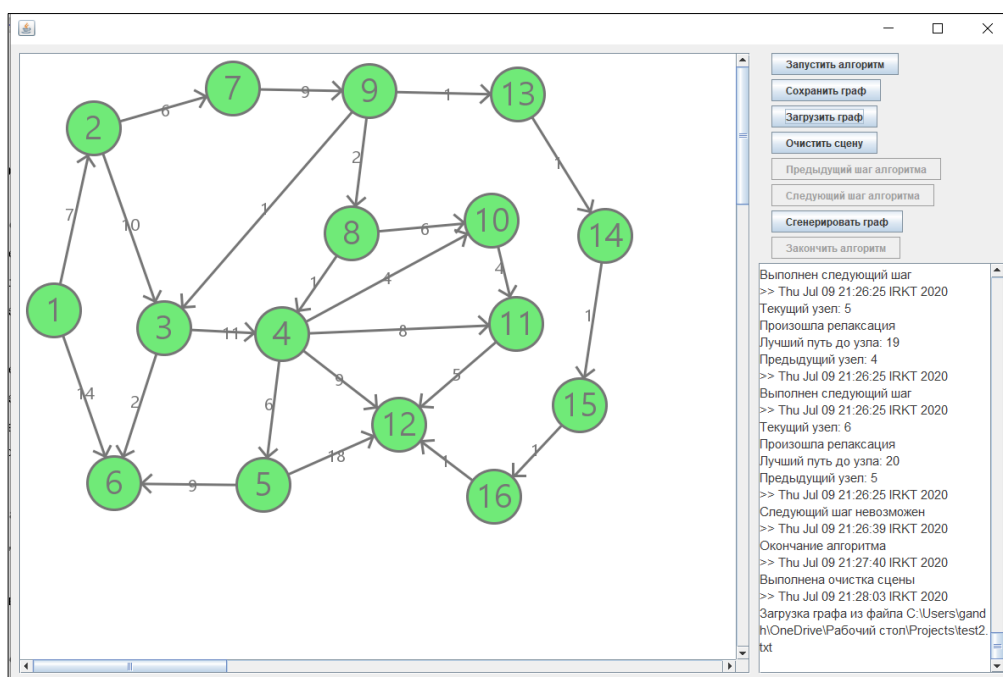


Рисунок 12 - Рисунок тестирования 6

7. Нажатие на кнопку «Сохранить граф». Далее в выбранный файл сохраняется граф.

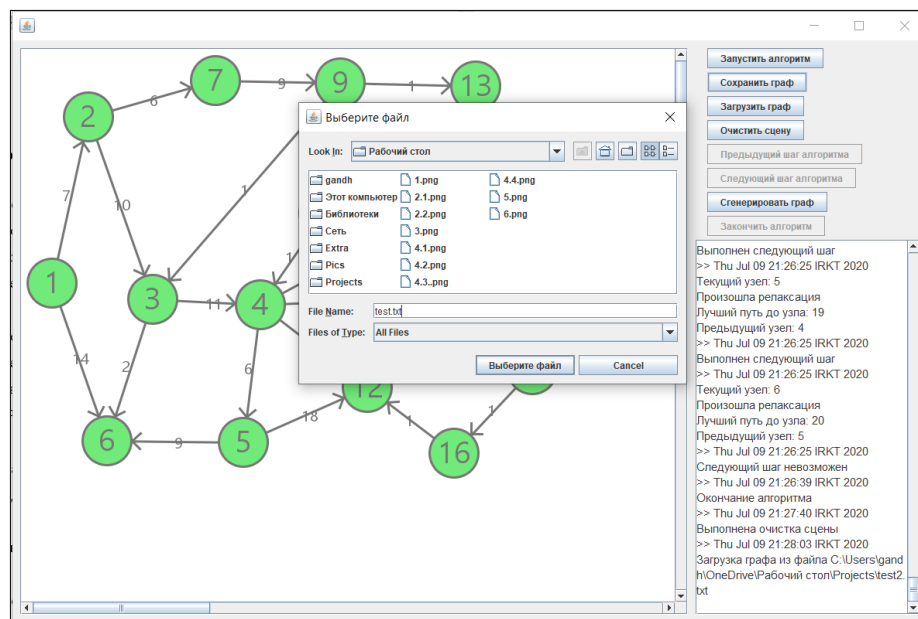


Рисунок 13 - Рисунок тестирования 7

5.2. Тестирование алгоритма

План тестирования алгоритма Дейкстры представлен в таблице 1.

Таблица 1 - Тестирование алгоритма

№	Начальные данные	Результат
1	1 2 7 1 3 9 1 6 14 2 3 10 2 4 15 3 4 11 3 6 2 4 5 6 5 6 9	<p> Расстояние от вершины '1'</p> <p>-----</p> <p> До вершины '2': 7 </p> <p>Путь: 1 → 2(7)</p> <p>-----</p> <p> До вершины '3': 9 </p> <p>Путь: 1 → 3(9)</p> <p>-----</p> <p> До вершины '4': 20 </p> <p>Путь: 1 → 3(9) → 4(20)</p> <p>-----</p> <p> До вершины '5': 26 </p> <p>Путь: 1 → 3(9) → 4(20) → 5(26)</p> <p>-----</p> <p> До вершины '6': 11 </p> <p>Путь: 1 → 3(9) → 6(11)</p> <p>-----</p>

Продолжение таблицы 1

2	Удалим вершину 3 и запустим алгоритм	<p> Расстояние от вершины '1'</p> <p>-----</p> <p> До вершины '2': 7 </p> <p>Путь: 1 → 2(7)</p> <p>-----</p> <p> До вершины '3': 22 </p> <p>Путь: 1 → 2(7) → 3(22)</p> <p>-----</p> <p> До вершины '4': 28 </p> <p>Путь: 1 → 2(7) → 3(22) → 4(28)</p> <p>-----</p> <p> До вершины '5': 14 </p> <p>Путь: 1 → 5(14)</p> <p>-----</p>
3	После запуска алгоритма попробуем нажать на кнопку «Запуск алгоритма»	Алгоритм запускается заново
4	После завершения алгоритма попробуем нажать на кнопку «Следующий шаг алгоритма»	Ошибка: кнопка недоступна; окончание алгоритма
5	Добавим стартовую вершину, не связывая её с графом, и запустим алгоритм	Ошибка: некорректный граф; стартовая вершина не имеет ребер
6	Начальные данные: 1 2 7 1 3 9 1 6 14 2 3 10 5 6 9 1 1 1	Ошибка: некорректный граф; в графе невозможно создать петли
7	Начальные данные: 1 2 1 1 3 4 2 4 2 3 1 1 1 3 2.5	Ошибка: некорректный граф; веса ребер должны задавать натуральными числами

Продолжение таблицы 1

8	1 2 7	Расстояние от вершины '1'
	1 6 14	-----
	2 3 10	До вершины '2': 7
	2 7 6	Путь: 1 → 2(7)
	8 10 9	-----
	10 9 2	До вершины '3': 17
	9 11 6	Путь: 1 → 2(7) → 3(17)
	11 12 4	-----
	12 13 5	До вершины '4': 25
	5 13 18	Путь: 1 → 2(7) → 7(13) →
	5 6 9	9(22) → 8(24) → 4(25)
	10 3 1	-----
	3 4 11	До вершины '5': 31
	9 4 1	Путь: 1 → 2(7) → 7(13) →
	4 5 6	9(22) → 8(24) → 4(25) →
	4 13 9	5(31)
	4 12 8	-----
	4 11 4	До вершины '6': 14
		Путь: 1 → 6(14)

		До вершины '7': 13
		Путь: 1 → 2(7) → 7(13)

		До вершины '8': 24
		Путь: 1 → 2(7) → 7(13) →
		9(22) → 8(24)

		До вершины '9': 22
		Путь: 1 → 2(7) → 7(13) →
		9(22)

		До вершины '10': 29
		Путь: 1 → 2(7) → 7(13) →
		9(22) → 8(24) → 4(25) →
		10(29)

		До вершины '11': 33

Продолжение таблицы 1

		<p>Путь: 1 → 2(7) → 7(13) → 9(22) → 8(24) → 4(25) → 11(33)</p> <p>-----</p> <p> До вершины '12': 34 Путь: 1 → 2(7) → 7(13) → 9(22) → 8(24) → 4(25) → 12(34)</p>
9	<p>1 2 7</p> <p>1 6 14</p> <p>2 3 10</p> <p>2 8 6</p> <p>8 10 9</p> <p>10 9 2</p> <p>9 11 6</p> <p>11 12 4</p> <p>12 13 5</p> <p>5 13 18</p> <p>5 6 9</p> <p>2 6 2</p> <p>10 3 1</p> <p>3 4 11</p> <p>9 4 1</p> <p>4 5 6</p> <p>4 13 9</p> <p>4 12 8</p> <p>4 11 4</p> <p>10 14 1</p> <p>14 15 1</p> <p>15 16 1</p> <p>16 17 1</p> <p>17 13 1</p>	<p> Расстояние от вершины '1'</p> <p>-----</p> <p> До вершины '2': 7 Путь: 1 → 2(7)</p> <p>-----</p> <p> До вершины '3': 17 Путь: 1 → 2(7) → 3(17)</p> <p>-----</p> <p> До вершины '4': 25 Путь: 1 → 2(7) → 7(13) → 9(22) → 8(24) → 4(25)</p> <p>-----</p> <p> До вершины '5': 31 Путь: 1 → 2(7) → 7(13) → 9(22) → 8(24) → 4(25) → 5(31)</p> <p>-----</p> <p> До вершины '6': 14 Путь: 1 → 6(14)</p> <p>-----</p> <p> До вершины '7': 13 Путь: 1 → 2(7) → 7(13)</p> <p>-----</p> <p> До вершины '8': 24 Путь: 1 → 2(7) → 7(13) → 9(22) → 8(24)</p> <p> До вершины '9': 22 Путь: 1 → 2(7) → 7(13) →</p>

Продолжение таблицы 1

		<p>9(22)</p> <p>-----</p> <p> До вершины '10': 29 </p> <p>Путь: 1 → 2(7) → 7(13) → 9(22) → 8(24) → 4(25) → 10(29)</p> <p>-----</p> <p> До вершины '11': 33 </p> <p>Путь: 1 → 2(7) → 7(13) → 9(22) → 8(24) → 4(25) → 11(33)</p> <p>-----</p> <p> До вершины '12': 27 </p> <p>Путь: 1 → 2(7) → 7(13) → 9(22) → 13(23) → 14(24) → 15(25) → 16(26) → 12(27)</p> <p>-----</p> <p> До вершины '13': 23 </p> <p>Путь: 1 → 2(7) → 7(13) → 9(22) → 13(23)</p> <p>-----</p> <p> До вершины '14': 24 </p> <p>Путь: 1 → 2(7) → 7(13) → 9(22) → 13(23) → 14(24)</p> <p>-----</p> <p> До вершины '15': 25 </p> <p>Путь: 1 → 2(7) → 7(13) → 9(22) → 13(23) → 14(24) → 15(25)</p> <p>-----</p> <p> До вершины '16': 26 </p> <p>Путь: 1 → 2(7) → 7(13) → 9(22) → 13(23) → 14(24) → 15(25) → 16(26)</p> <p>-----</p>
10	Попробуем добавить ребро к вершинам уже имеющим ребро	Ошибка: вершины уже имеют ребро

Продолжение таблицы 1

11	Запустим алгоритм, не создавая графа	Ошибка: алгоритм невозможно запустить без графа
12	Не запуская алгоритма, нажмем на кнопку «Следующий шаг алгоритма»	Ошибка: чтобы сделать шаг, запустите алгоритм
13	Нажмем на кнопку «Сохранить граф» без созданного графа	Ошибка: граф не сохранится
14	Начальные данные: 1 2 -1	Ошибка: некорректное значение ребра

5.3. Unit тестирование

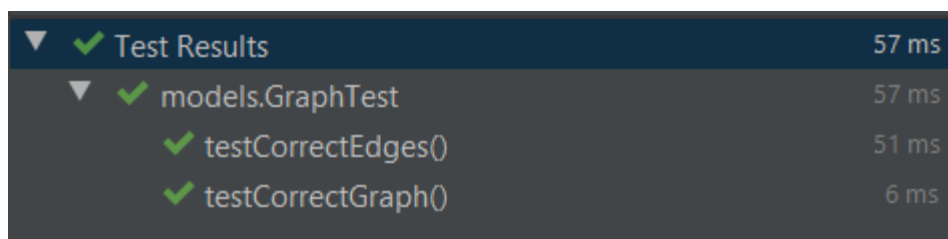
При написании unit тестирования будут созданы классы, в которых выполняется проверка правильности реализации методов. При запуске тестов сначала создается экземпляр тест-класса (для каждого теста в классе отдельный экземпляр класса), затем выполняется метод, который запускает сам тест. Если какой-либо из методов выбрасывает исключение, тест считается провалившимся.

При сравнении ожидаемых и реальных результатов работы методов будет использоваться платформа JUnit 5. Примеры выполненных тестов представлены на следующих рисунках.

Для проверки корректной работы структур данных в программе был создан класс тестов GraphTest.

fun testCorrectEdges() - проверяет корректное заполнение списка вершин edges

fun testCorrectGraph() - проверяет корректное заполнение хэш-таблицы graph



▼	✓ Test Results	57 ms
▼	✓ models.GraphTest	57 ms
	✓ testCorrectEdges()	51 ms
	✓ testCorrectGraph()	6 ms

Рисунок 14 - Результат тестирования GraphTest

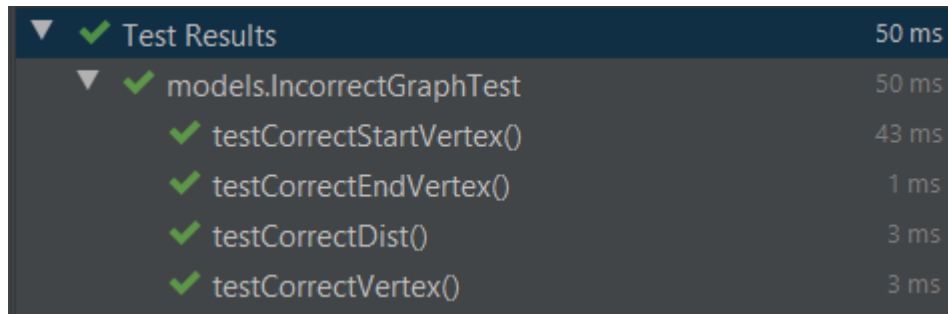
Для проверки корректной работы вызова исключений при подаче некорректного графа был создан класс тестов IncorrectGraphTest.

fun testCorrectDist() - проверяет некорректный вес ребра

fun testCorrectVertex() - проверяет некорректное имя вершины

fun testCorrectStartVertex() - проверяет некорректное имя стартовой вершины

fun testCorrectEndVertex() - проверяет некорректное имя конечной вершины



A screenshot of a test results window showing the execution of five tests for the IncorrectGraphTest class. All tests passed, indicated by green checkmarks. The tests and their durations are: testCorrectStartVertex() (43 ms), testCorrectEndVertex() (1 ms), testCorrectDist() (3 ms), and testCorrectVertex() (3 ms). The total duration for the class is 50 ms.

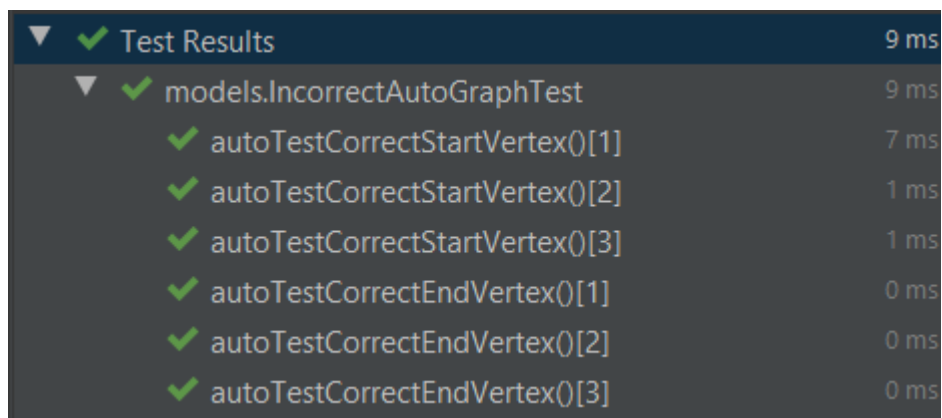
▼	✓	Test Results	50 ms
▼	✓	models.IncorrectGraphTest	50 ms
	✓	testCorrectStartVertex()	43 ms
	✓	testCorrectEndVertex()	1 ms
	✓	testCorrectDist()	3 ms
	✓	testCorrectVertex()	3 ms

Рисунок 15 - Результат тестирования IncorrectGraphTest

Также для проверки корректной работы вызова исключений при подаче некорректного графа был создан класс тестов с автоматическим набором тестов через DynamicTest - IncorrectGraphTest.

fun autoTestCorrectStartVertex() - проверяет некорректное имя стартовой вершины

fun autoTestCorrectEndVertex() - проверяет некорректное имя конечной вершины



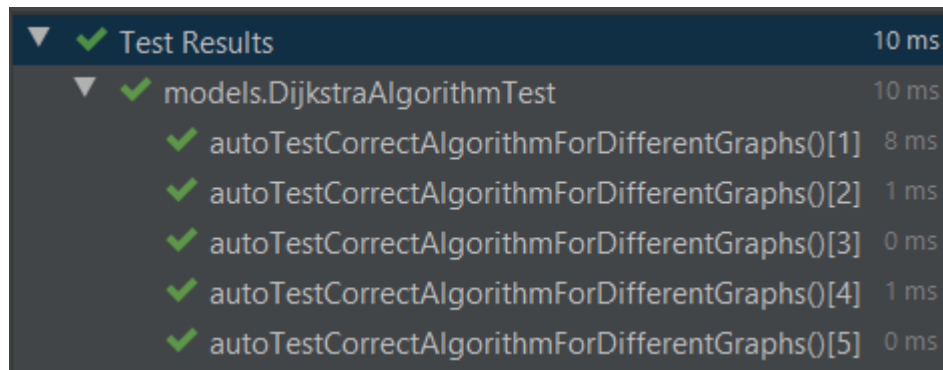
A screenshot of a test results window showing the execution of six automatic tests for the IncorrectAutoGraphTest class. All tests passed, indicated by green checkmarks. The tests and their durations are: autoTestCorrectStartVertex()[1] (7 ms), autoTestCorrectStartVertex()[2] (1 ms), autoTestCorrectStartVertex()[3] (1 ms), autoTestCorrectEndVertex()[1] (0 ms), autoTestCorrectEndVertex()[2] (0 ms), and autoTestCorrectEndVertex()[3] (0 ms). The total duration for the class is 9 ms.

▼	✓	Test Results	9 ms
▼	✓	models.IncorrectAutoGraphTest	9 ms
	✓	autoTestCorrectStartVertex()[1]	7 ms
	✓	autoTestCorrectStartVertex()[2]	1 ms
	✓	autoTestCorrectStartVertex()[3]	1 ms
	✓	autoTestCorrectEndVertex()[1]	0 ms
	✓	autoTestCorrectEndVertex()[2]	0 ms
	✓	autoTestCorrectEndVertex()[3]	0 ms

Рисунок 16 - Результат тестирования IncorrectGraphTest

Для проверки корректной работы алгоритма Дейкстры был создан класс тестов `DijkstraAlgorithmTest`, в котором для разных графов выводится путь из стартовой вершины в конечную.

`fun autoTestCorrectAlgorithmForDifferentGraphs()` - проверяет корректный вывод алгоритма Дейкстры для набора графов.



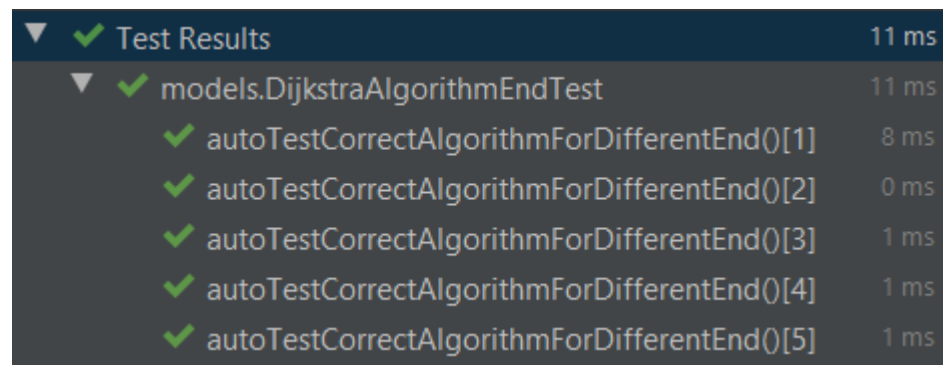
The screenshot shows a test results window with a dark background. It lists the execution of the `DijkstraAlgorithmTest` class and its methods, all of which passed successfully, indicated by green checkmarks. The total execution time for the test class is 10 ms.

Test Results	Time
models.DijkstraAlgorithmTest	10 ms
autoTestCorrectAlgorithmForDifferentGraphs()[1]	8 ms
autoTestCorrectAlgorithmForDifferentGraphs()[2]	1 ms
autoTestCorrectAlgorithmForDifferentGraphs()[3]	0 ms
autoTestCorrectAlgorithmForDifferentGraphs()[4]	1 ms
autoTestCorrectAlgorithmForDifferentGraphs()[5]	0 ms

Рисунок 17 - Результат тестирования `DijkstraAlgorithmTest`

Также для проверки корректной работы алгоритма Дейкстры был создан класс тестов `DijkstraAlgorithmEndTest`, в котором для одного графа выводится путь из стартовой вершины в различные конечные.

`fun autoTestCorrectAlgorithmForDifferentEnd()` - проверяет корректный вывод алгоритма Дейкстры.



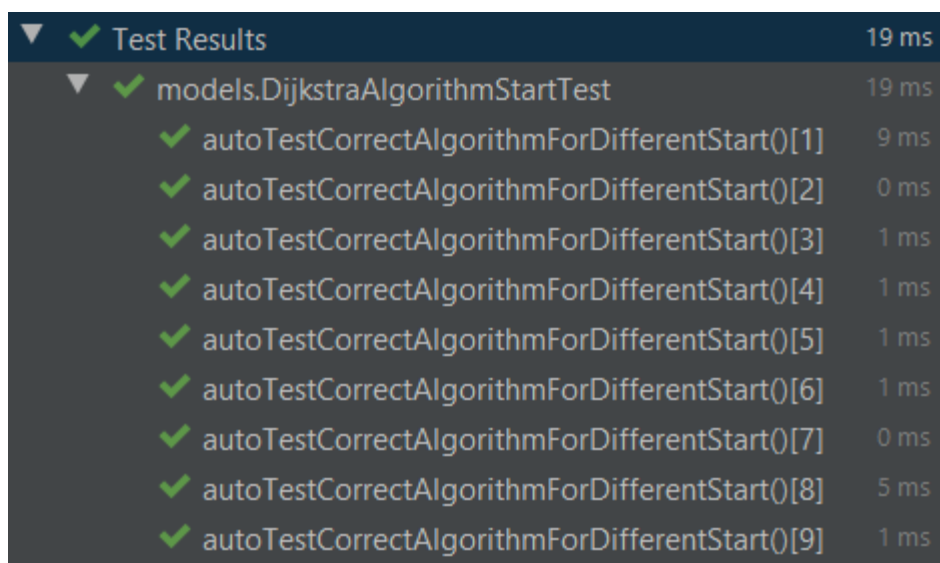
The screenshot shows a test results window with a dark background. It lists the execution of the `DijkstraAlgorithmEndTest` class and its methods, all of which passed successfully, indicated by green checkmarks. The total execution time for the test class is 11 ms.

Test Results	Time
models.DijkstraAlgorithmEndTest	11 ms
autoTestCorrectAlgorithmForDifferentEnd()[1]	8 ms
autoTestCorrectAlgorithmForDifferentEnd()[2]	0 ms
autoTestCorrectAlgorithmForDifferentEnd()[3]	1 ms
autoTestCorrectAlgorithmForDifferentEnd()[4]	1 ms
autoTestCorrectAlgorithmForDifferentEnd()[5]	1 ms

Рисунок 18 - Результат тестирования `DijkstraAlgorithmEndTest`

Также для проверки корректной работы алгоритма Дейкстры был создан класс тестов `DijkstraAlgorithmStartTest`, в котором для одного графа выводится путь из различных стартовых вершин в конечную.

`fun autoTestCorrectAlgorithmForDifferentStart()` - проверяет корректный вывод алгоритма Дейкстры.

A screenshot of a test results window. The window has a dark background with a list of test cases. Each item is preceded by a green checkmark and a downward arrow. The test cases are listed with their names and execution times in milliseconds. The total time for the 'Test Results' group is 19 ms.

▼ ✓ Test Results	19 ms
▼ ✓ models.DijkstraAlgorithmStartTest	19 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[1]	9 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[2]	0 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[3]	1 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[4]	1 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[5]	1 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[6]	1 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[7]	0 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[8]	5 ms
✓ autoTestCorrectAlgorithmForDifferentStart()[9]	1 ms

Рисунок 19 - Результат тестирования `DijkstraAlgorithmStartTest`

ЗАКЛЮЧЕНИЕ

В ходе выполнения задания учебной практики были изучены основы программирования на языке Kotlin. Для изучения данного языка был пройден интерактивный курс «Введение в Kotlin JVM» на платформе Stepik. После чего была разработана программа, которая находит кратчайшие пути в графе с помощью алгоритма Дейкстры. Разработанная программа соответствует всем заявленным требованиям спецификации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дж. Скин, Д. Гринхол. Kotlin. Программирование для профессионалов. М.: Big Nerd Ranch, 2018, 464 с.
2. JUnit 5 для разработчиков Kotlin. URL: <https://www.codeflow.site/ru/article/junit-5-kotlin> (дата обращения: 07.07.2020).
3. Тестирование с помощью JUnit 5 на Kotlin. URL: <https://habr.com/ru/post/346452/> (дата обращения: 07.07.2020).
4. Жемеров Дмитрий, Исакова Светлана. Kotlin в действии. М.: Manhing, 2017, 403 с.
5. Как начать пользоваться Swing GUI-визардом IntelliJ IDEA. Подробная инструкция. URL: <https://habr.com/ru/post/305974/> (дата обращения: 04.07.2020).
6. Паттерны для новичков: MVC vs MVP vs MVVM. URL: <https://habr.com/ru/post/215605/> (дата обращения: 07.07.2020).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД MAIN.KT

```
import views.MainWindow
import java.awt.Dimension
import javax.swing.JFrame

class Main {
    companion object{
        @JvmStatic
        fun main(args: Array<String>) {
            val frame = JFrame()
            frame.size = Dimension(1200, 800)

            frame.defaultCloseOperation = JFrame.EXIT_ON_CLOSE

            frame.add(MainWindow())
            frame.isVisible = true
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД UICONSTANTS.KT

```
package views

import com.sun.org.apache.xpath.internal.functions.FunctionOneArg
import java.awt.Color
import java.awt.Font

object UIConstants {
    const val circleRadius = 30
    const val circleStrokeWidth = 3
    const val spaceBetweenButtonsInToolbar = 5
    const val graphScreenSheetWidth = 3000
    const val graphScreenSheetHeight = 3000
    const val horizontalScrollIncrement = 20
    const val verticalScrollIncrement = 20
    const val edgeWidth = 6
    const val arrowHeight = 10
    const val arrowWidth = 10
    const val edgeWeightTextOffsetHeight = 10

    const val dualEdgeRotationAngleInGrads = Math.PI / 4

    val nodeFillColor = Color(106, 233, 114)
    val nodeActiveFillColor = Color(255, 0, 0)
    val nodeStrokeFillColor = Color(112, 112, 112)
    val textColor = Color(112, 112, 112)
    val edgeColor = Color(112, 112, 112)
    val nodeTextFont = Font("Segoe UI", Font.PLAIN, 40)
    val edgeWeightTextFont = Font("Segoe UI", Font.PLAIN, 20)
    val nodeSignTextFont = Font("Segoe UI", Font.PLAIN, 15)
    val logsOutputTextFont = Font("Sans Serif", Font.PLAIN, 15)
}
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД MAINWINDOW.KT

```
PACKAGE views

import presenters.MainPresenter
import views.graphview.GraphViewHolder
import views.toolbarview.ToolbarView
import java.awt.BorderLayout
import java.awt.Dimension
import javax.swing.JPanel
import javax.swing.border.EmptyBorder

class MainWindow : JPanel() {
    init {
        layout = BorderLayout()
        border = EmptyBorder(10, 10, 10,10)

        val graphViewHolder = GraphViewHolder()
        add(graphViewHolder)

        val toolbarView = ToolbarView()
        toolbarView.preferredSize = Dimension(300, 200)
        add(toolbarView, BorderLayout.EAST)
    }
}
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД TOOLBARVIEW.KT

```
PACKAGE views.toolbarview

import presenters.Event
import presenters.ToolbarPresenter
import presenters.ToolbarView
import presenters.ToolbarViewElement
import views.UIConstants
import java.awt.Color
import javax.swing.*
import javax.swing.border.EmptyBorder

class ToolbarView : JPanel(), ToolbarView {

    private val toolbarPresenter = ToolbarPresenter(this)

    private val btn1 = JButton("Запустить алгоритм")
    private val btn2 = JButton("Сохранить граф")
    private val btn3 = JButton("Загрузить граф")
    private val btn4 = JButton("Очистить сцену")
    private val btn5 = JButton("Предыдущий шаг алгоритма")
    private val btn6 = JButton("Следующий шаг алгоритма")
    private val btn7 = JButton("Сгенерировать граф")
    private val btn8 = JButton("Закончить алгоритм")

    private val logTextView = JTextArea()

    init {
        border = EmptyBorder(0, 10, 0, 0)
        layout = BoxLayout(this, BoxLayout.Y_AXIS)

        btn1.addActionListener {
            toolbarPresenter.chainToolbarEvent(Event.OnStartAlgorithm)}
        add(btn1)
        add(Box.createVerticalStrut(UIConstants.spaceBetweenButtonsInToolbar))

        btn2.addActionListener {
            val filePath: String? = toolbarPresenter.getFilePath()

            if (filePath != null) {
                toolbarPresenter.chainToolbarEvent(Event.SaveGraph(filePath))
            }
        }
        add(btn2)
        add(Box.createVerticalStrut(UIConstants.spaceBetweenButtonsInToolbar))

        btn3.addActionListener {
            val filePath: String? = toolbarPresenter.getFilePath()

            if (filePath != null) {
                toolbarPresenter.chainToolbarEvent(Event.DownloadGraph(filePath))
            }
        }
        add(btn3)
        add(Box.createVerticalStrut(UIConstants.spaceBetweenButtonsInToolbar))
    }
}
```

```

        btn4.addActionListener {
            toolbarPresenter.chainToolbarEvent(Event.Clear)
        }
        add(btn4)
        add(Box.createVerticalStrut(UIConstants.spaceBetweenButtonsInToolbar))

        btn5.addActionListener {
            toolbarPresenter.chainToolbarEvent(Event.PreviousStep)
        }
        add(btn5)
        add(Box.createVerticalStrut(UIConstants.spaceBetweenButtonsInToolbar))

        btn6.addActionListener {
            toolbarPresenter.chainToolbarEvent(Event.NextStep)
        }
        add(btn6)
        add(Box.createVerticalStrut(UIConstants.spaceBetweenButtonsInToolbar))

        btn7.addActionListener {
            toolbarPresenter.chainToolbarEvent(Event.GenerateGraph)
        }
        add(btn7)
        add(Box.createVerticalStrut(UIConstants.spaceBetweenButtonsInToolbar))

        btn8.addActionListener {
            toolbarPresenter.chainToolbarEvent(Event.EndAlgorithm)
        }
        add(btn8)
        add(Box.createVerticalStrut(UIConstants.spaceBetweenButtonsInToolbar))

        btn5.isEnabled = false
        btn6.isEnabled = false
        btn8.isEnabled = false

        logTextView.background = Color.WHITE
        logTextView.isEditable = false
        add(JScrollPane(logTextView))
        logTextView.lineWrap = true
        logTextView.font = UIConstants.logsOutputTextFont
        logTextView.text += ">> ${toolbarPresenter.getCurrentDateString()}\nПрограмма
запущена\n"
    }

    override fun lockElement(element: ToolbarViewElement) {
        when (element) {
            ToolbarViewElement.START -> btn1.isEnabled = false
            ToolbarViewElement.SAVE -> btn2.isEnabled = false
            ToolbarViewElement.DOWNLOAD -> btn3.isEnabled = false
            ToolbarViewElement.CLEAR -> btn4.isEnabled = false
            ToolbarViewElement.NEXT_STEP -> btn5.isEnabled = false
            ToolbarViewElement.PREVIOUS_STEP -> btn6.isEnabled = false
            ToolbarViewElement.GENERATE -> btn7.isEnabled = false
            ToolbarViewElement.END_ALGORITHM -> btn8.isEnabled = false
        }
    }

    override fun unlockElement(element: ToolbarViewElement) {
        when (element) {
            ToolbarViewElement.START -> btn1.isEnabled = true

```



```

        ToolbarViewElement.SAVE -> btn2.isEnabled = true
        ToolbarViewElement.DOWNLOAD -> btn3.isEnabled = true
        ToolbarViewElement.CLEAR -> btn4.isEnabled = true
        ToolbarViewElement.NEXT_STEP -> btn5.isEnabled = true
        ToolbarViewElement.PREVIOUS_STEP -> btn6.isEnabled = true
        ToolbarViewElement.GENERATE -> btn7.isEnabled = true
        ToolbarViewElement.END_ALGORITHM -> btn8.isEnabled = true
    }
}

override fun getFilePath() : String? {
    val fileChooser = JFileChooser()
    //fileChooser.fileFilter = FileNameExtensionFilter(".txt", "txt")
    // fileChooser.isAcceptAllFileFilterUsed = false
    val action = fileChooser.showDialog(null, "Выберите файл")

    if (action == JFileChooser.APPROVE_OPTION)
    {
        val file = fileChooser.selectedFile

        return file.absolutePath
    }

    return null
}

override fun showLog(logMsg: String) {
    logTextView.text += ">> $logMsg"
}
}

```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД GRAPH.KT

```
PACKAGE models

import java.util.*
import kotlin.collections.HashMap

// Класс ребра (v, v, w)
class Edge(val v1: Int, val v2: Int, val dist: Int){
    override fun toString() = "($v1, $v2, $dist)"
}

/* Класс вершины (v)
 * Наследуется от интерфейса Comparable для
 * вершины имеют общий порядок размещения в TreeSet,
 * который зависит от веса, при совпадении весов
 * вершины сортируются по названию
 */

class Vertex(val name: Int) : Comparable<Vertex> {

    var dist = Int.MAX_VALUE // Первоначальная дистанция для всех ребер равна inf
    var previous: Vertex? = null // Вершина, из которой мы перешли в текущую
    val neighbours = HashMap<Vertex, Int>() // Хэш-таблица для хранения пар хар.
    соседей (v, w)

    /* Функция вывода кратчайшего пути
     * если достигнута начальная вершина, она выводится
     * если до начальной вершины невозможно добраться, то соотв. сообщение
     * иначе, рекуррентно выводится путь в виде -> v(w)
     */

    fun printPath(): String{
        var endString: String = ""
        endString = when(previous){
            this -> (name+1).toString()
            null -> "не существует"
            else -> {
                val str: String = previous!!.printPath()
                "$str → ${name+1}($dist)"
            }
        }
        return endString
    }

    override fun compareTo(other: Vertex): Int {
        if (dist == other.dist) return name.compareTo(other.name)
        return dist.compareTo(other.dist)
    }

    override fun toString() :String{
        var pr :String = (previous?.name?.plus(1)).toString()
        if (previous == null)
            pr = "n"
        if (dist == Int.MAX_VALUE)
            return "($name,-,$pr)"
        return "($name,$dist,$pr)"
    }
}
```

```

    }

}

// Класс графа (список ребер)
class Graph(private val edges: List<Edge>){
    // Хэш-таблица вершин (имя, объект), построенная из списка ребер
    private val graph = HashMap<Int, Vertex>(edges.size)
    private val snapshotKeeper: SnapshotKeeper = SnapshotKeeper()

    // Инициализация всех вершин и соседей
    init {
        // Проход для инициализации всех вершин
        for (e in edges) {
            // Если текущей вершины еще нет в graph, то добавить;
            // ключ - имя вершины, значение - объект вершина
            if (!graph.containsKey(e.v1))
                graph.put(e.v1, Vertex(e.v1))
            if (!graph.containsKey(e.v2))
                graph.put(e.v2, Vertex(e.v2))
        }

        // Проход для записи всех соседей
        for (e in edges)
            // Кладет в таблицу соседей (имя соседа, вес ребра до него)
            graph[e.v1]!!.neighbours.put(graph[e.v2]!!, e.dist)

            /* Промежуточный вывод хэш-таблицы соседей вершин
            * print("key = ${e.v2} value = ")
            * println(graph[e.v1]!!.neighbours.getValue(graph[e.v2]!!))
            */
    }

    /* Запускает алгоритм Дейкстры,
    * используя указанную исходную вершину;
    * устанавливает характеристики графа,
    * заполняет treeSet
    */

    fun dijkstra(startName: Int) {
        if (!graph.containsKey(startName)) {
            println("Граф не содержит стартовую вершину '${startName+1}'")
            return
        }

        val source: Vertex? = graph[startName] // Объект исходная вершина
        val treeSet: TreeSet<Vertex> = TreeSet<Vertex>() // Отсортированная коллекция
        вершин в виде бинарной кучи

        // Установка характеристик вершин
        for (v in graph.values) {
            v.previous = if (v == source) source else null
            v.dist = if (v == source) 0 else Int.MAX_VALUE
            treeSet.add(v)
        }
        this.makeSnapshot(startName, false)
        dijkstra(treeSet)
    }

    // Запускается алгоритм Дейкстры для бинарной кучи вершин
    private fun dijkstra(treeSet: TreeSet<Vertex>) {

```

```

while (!treeSet.isEmpty()) {

    /* В currV кладется вершина с кратчайшей дистанцией;
    * в начале алгоритма это начальная вершина,
    * т.к. в source dist = 0, в ост. inf
    */
    val currV = treeSet.pollFirst()

    // Если расстояние inf, то можно пропустить вершины,
    // т.к. они недостижимы
    if (currV!!.dist == Int.MAX_VALUE)
        break

    // Проверка расстояний до каждого соседа
    for (ngb in currV.neighbours) {

        // Текущий сосед
        val currentNgb = ngb.key
        // Пересчитываем расстояние до текущего соседа
        val alternateDist = currV.dist + ngb.value

        // Если новое расстояние < текущего,
        // то меняем в treeSet расстояние до вершины
        if (alternateDist < currentNgb.dist) {

            treeSet.remove(currentNgb)
            currentNgb.dist = alternateDist
            currentNgb.previous = currV
            treeSet.add(currentNgb)
            this.makeSnapshot(currentNgb.name, true)

        }
        else{
            this.makeSnapshot(currentNgb.name, false)
        }
    }
}

// Печатает путь от начальной вершины до заданной конечной
private fun printPath(endName: Int, startName: Int): String {
    var strEnd: String = ""

    if (!graph.containsKey(endName))
        return "Граф не содержит конечную вершину '${endName+1}'"

    /*if (showAllPaths)
        strEnd = printAllPaths()
    else*/
        strEnd = graph[endName]!!.printPath()

    return /*"Кратчайший путь из $startName в ${endName+1}:\n" +
    graph[endName]!!.dist.toString() + "\n" + */strEnd
}

// Печатает путь от начальной вершины до каждой другой
fun printAllPaths(): String{

    var endString: String = ""

```

```

        for (v in graph.values)
            endString += v.printPath() + "\n"

    return endString
}

/*  Пример использования:
 *   val GRAPH:List<Edge> = listOf(
 *       Edge(1, 2, 7),
 *       Edge(1, 3, 9),
 *       Edge(2, 3, 1)
 *   )
 *
 *   val START = 1
 *   val END = 3
 *
 *   with (Graph(GRAPH)){
 *       dijkstra(START)
 *       printPath(END)
 *   }
 */

private fun makeSnapshot(vertex:Int,relax:Boolean){
    snapshotKeeper.putSnapshot(Snapshot(graph,vertex,relax))
}

fun getSnapshotHistory() :SnapshotKeeper{
    return snapshotKeeper
}

fun getPath(startVertex: Int): String{
    val arr: IntArray = IntArray(graph.size) {it+1}
    var ansString: String = "| Расстояние от вершины '$startVertex'\n-----\n"

    arr.forEach {
        if (it != startVertex) {
            ansString += "| До вершины '$it':      "
            if (graph.containsKey(it-1)) {
                ansString += if (graph[it - 1]!!.dist == Int.MAX_VALUE) "-" else
graph[it - 1]!!.dist.toString()
                ansString += " | Путь:      " + printPath(it - 1, startVertex)
+
                " \n-----\n"
            }
        }
    }

    return ansString
}

fun getEdgesAsString(): String {
    val edgesAsString = StringBuilder("")
    for (e in edges)
        edgesAsString.append("${e.toString()}, ")

    edgesAsString.delete(edgesAsString.length - 2, edgesAsString.length).append("")
    return edgesAsString.toString()
}

```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД SNAPSHOT.KT

```
PACKAGE models

interface Memento{
    fun getStringVertex() :String
    fun getCurrentVertex():Int
    fun getAllInfo():String
    fun getRelax():Boolean
}

class Snapshot (graph: HashMap<Int, Vertex>, private val currentVertex: Int,private val
relax:Boolean): Memento
{
    private val vertexAsString = StringBuilder("")

    init{
        for(v in graph) {
            vertexAsString.append(v.value.toString()+" ", ")
        }
        vertexAsString.delete(vertexAsString.length - 2,
vertexAsString.length).append("")
    }

    override fun getStringVertex(): String {
        return vertexAsString.toString()
    }

    override fun getCurrentVertex(): Int {
        return currentVertex
    }
    override fun getRelax():Boolean {
        return relax
    }
    override fun getAllInfo(): String{
        return ("($currentVertex), ($relax), ${vertexAsString.toString()}")
    }
}

class SnapshotKeeper(){
    private var snapshotArray = emptyArray<Snapshot>()
    private var arraySize :Int = 0
    fun putSnapshot(new:Snapshot){
        snapshotArray+=new
        arraySize++
    }
    fun getSnapshot(index:Int):Snapshot?{
        if (index<0||index>=arraySize) return null
        return snapshotArray[index]
    }
    fun getSize():Int{
        return arraySize
    }
}
```

ПРИЛОЖЕНИЕ Ж

ИСХОДНЫЙ КОД BROADCASTPRESENTER.KT

```
PACKAGE presenters

interface EventSubscriber {
    fun handleEvent(event: Event)
}

object BroadcastPresenter {

    private val subscribers = ArrayList<EventSubscriber>()

    fun registerSubscriber(subscriber: EventSubscriber) {
        subscribers.add(subscriber)
    }

    fun generateEvent(event: Event) {
        subscribers.forEach() { it.handleEvent(event) }
    }
}
```

ПРИЛОЖЕНИЕ 3

ИСХОДНЫЙ КОД EVENT.KT

```
PACKAGE presenters
```

```
sealed class Event {  
    //Нажатие на кнопку начать алгоритм  
    object OnStartAlgorithm : Event()  
    //Подтверждение начала алгоритма (выбор стартовой вершины и запуск)  
    object AfterAlgorithmStarted : Event()  
  
    //Полное завершение работы с алгоритмом (после подтверждения)  
    object AfterAlgorithmEnded : Event()  
  
    class SaveGraph(val fileName: String): Event()  
    class DownloadGraph (val fileName: String): Event()  
    object Clear : Event()  
    object PreviousStep : Event()  
    object NextStep : Event()  
    object GenerateGraph : Event()  
    object EndAlgorithm : Event()  
  
    class LogEvent(val logInfo: String) : Event()  
}
```


ПРИЛОЖЕНИЕ И

ИСХОДНЫЙ КОД MAINPRESENTER.KT

```
PACKAGE presenters

import models.Edge
import models.Graph
import models.SnapshotKeeper
import models.Snapshot
import views.graphview.*
import javax.swing.JOptionPane
import java.io.File
import java.io.InputStream
import java.nio.file.Files
import java.nio.file.StandardOpenOption

interface GraphView {
    fun update()
    fun displayDijkstraAlgorithmResult(result: String)
    fun setAlgorithmRunningFlag(isAlgorithmRunning: Boolean)
}

class DijkstraAlgorithmController(){
    var snapshotKeeper : SnapshotKeeper = SnapshotKeeper()
    var startNode : Int = -1
    var currentStep : Int = 0
    var answer:String = ""
    fun initStart(startNode:Int,snapshots : SnapshotKeeper, answer:String){
        this.startNode = startNode-1
        this.snapshotKeeper = snapshots
        this.answer = answer
        currentStep = -1
    }

    fun getNextStep():Snapshot?{
        if (currentStep>=snapshotKeeper.getSize()-1) return null
        currentStep++
        return snapshotKeeper.getSnapshot(currentStep)
    }

    fun getPreviousStep():Snapshot?{
        if (currentStep<=0) return null
        currentStep--
        return snapshotKeeper.getSnapshot(currentStep)
    }

    fun getLast():Snapshot?{
        return snapshotKeeper.getSnapshot(snapshotKeeper.getSize()-1)
    }

    fun isNextStepPossible():Boolean{
        if(currentStep>=snapshotKeeper.getSize()-1) return false
        return true
    }

    fun isPreviousStepPossible():Boolean{
        if(currentStep<=0) return false
        return true
    }
}
```

```

}

class MainPresenter(
    private val graphView: GraphView
) : EventSubscriber {

    init {
        BroadcastPresenter.registerSubscriber(this)
    }

    fun onAlgorithmEndConfirmed() {
        nodes.forEach { it.reset() }
        graphView.setAlgorithmRunningFlag(false)
        graphView.update()
        BroadcastPresenter.generateEvent(Event.AfterAlgorithmEnded)
    }

    override fun handleEvent(event: Event) {
        when (event) {
            is Event.OnStartAlgorithm -> {

                val node: Int? = requestStartNodeNumber()

                if (node != null) {
                    val logEvent = Event.LogEvent("Алгоритм запущен")
                    BroadcastPresenter.generateEvent(logEvent)
                    graphView.setAlgorithmRunningFlag(true)
                    BroadcastPresenter.generateEvent(Event.AfterAlgorithmStarted)
                    startAlgorithm(node)
                    graphView.update()
                }
                else {
                    return
                }
            }

            is Event.Clear -> {
                nodes.clear()
                dualEdges.clear()
                edges.clear()
                graphView.setAlgorithmRunningFlag(false)
                graphView.update()

                val logEvent = Event.LogEvent("Выполнена очистка сцены")
                BroadcastPresenter.generateEvent(logEvent)
            }

            is Event.NextStep->{
                //val logEvent = Event.LogEvent("Выполнен переход на следующий шаг
алгоритма")
                // BroadcastPresenter.generateEvent(logEvent)
                nextStep()
            }

            is Event.PreviousStep->{
                // val logEvent = Event.LogEvent("Выполнен переход на предыдущий шаг
алгоритма")
                // BroadcastPresenter.generateEvent(logEvent)
            }
        }
    }
}

```

```

        previousStep()
    }
    is Event.DownloadGraph->{
        val logEvent = Event.LogEvent("Загрузка графа из файла $
{event.fileName}")
        BroadcastPresenter.generateEvent(logEvent)
        downloadGraph(event.fileName)
    }
    is Event.SaveGraph->{
        val logEvent = Event.LogEvent("Сохранение графа в файл $
{event.fileName}")
        BroadcastPresenter.generateEvent(logEvent)
        saveGraph(event.fileName)
    }
    is Event.EndAlgorithm-> {
        val logEvent = Event.LogEvent("Окончание алгоритма")
        BroadcastPresenter.generateEvent(logEvent)
        finishAlgorithm()
    }
}
}

private fun requestStartNodeNumber() : Int? {
    val numbers = ArrayList<Int>()
    for(i in 1..nodes.size)
        numbers.add(i)

    val optionsHolder = EndNodeRequestPane(numbers)

    val responseCode =
        JOptionPane.showConfirmDialog(null, optionsHolder, "Исходные данные",
JOptionPane.OK_CANCEL_OPTION)

    return when (responseCode) {
        //Нажата "Ок"
        0 -> {
            if (numbers.size == 0) null
            else numbers[optionsHolder.startNodeNumber.selectedIndex]
        }

        else -> {
            null
        }
    }
}

val nodes = ArrayList<UINode>()
val edges = ArrayList<UIEdge>()
val dualEdges = ArrayList<UIDualEdge>()

private val dijkstraAlgorithmController = DijkstraAlgorithmController()

fun addNode(new:UINode){
    nodes.add(new)
    graphView.update()
}

```

```

fun addEdge(new:UIEdge){

    for (e in edges) {
        if (new.sourceNode == e.sourceNode && new.endNode == e.endNode)
            return
    }

    for(e in edges)
    {
        if(new.sourceNode == e.endNode && new.endNode == e.sourceNode) {
            dualEdges.add(UIDualEdge(new, e))
            edges.remove(e)
            graphView.update()
            return
        }
    }
    edges.add(new)
    graphView.update()
}

fun deleteEdge(deleted:UIEdge){

    val isDeleted = edges.remove(deleted)
    if(!isDeleted)
    {
        dualEdges.forEach {
            if(it.edge1 == deleted){
                edges.add(it.edge2)
                dualEdges.remove(it)
                graphView.update()
                return
            }
            if(it.edge2 == deleted)
            {
                edges.add(it.edge1)
                dualEdges.remove(it)
                graphView.update()
                return
            }
        }
    }
    else graphView.update()
}

fun deleteNode(deleted:UINode){

    val removableEdges = ArrayList<UIEdge>()
    for (e in edges){
        if (e.sourceNode == deleted || e.endNode == deleted)
            removableEdges.add(e)
    }

    edges.removeAll(removableEdges)

    val removableDualEdges = ArrayList<UIDualEdge>()
    dualEdges.forEach {
        if(it.edge1.sourceNode == deleted || it.edge2.sourceNode == deleted) {
            removableDualEdges.add(it)
        }
    }
}

```

```

    }
    dualEdges.removeAll(removableDualEdges)

    nodes.remove(deleted)
    graphView.update()
}

fun startAlgorithm(startNode:Int){ //где хранить конечный и начальный узел

    //проверка на существование вершины в массиве ребер

    var flag = false

    for(e in edges){
        if (nodes.indexOf(e.sourceNode) == startNode || nodes.indexOf(e.endNode) ==
startNode){
            flag = true
            break
        }
    }
    for(e in dualEdges){
        if (nodes.indexOf(e.edge1.sourceNode) == startNode ||
nodes.indexOf(e.edge1.endNode) == startNode || nodes.indexOf(e.edge2.sourceNode) ==
startNode || nodes.indexOf(e.edge2.endNode) == startNode){
            flag = true
            break
        }
    }
    if (!flag) {
        val logEvent = Event.LogEvent("Данная вершина не имеет ребер")
        BroadcastPresenter.generateEvent(logEvent)
        onAlgorithmEndConfirmed()
        for(n in nodes){
            n.reset()
        }
        graphView.update()
        return
    }

    val gr:ArrayList<Edge> = ArrayList<Edge>()
    for (e in edges){

gr.add(Edge(nodes.indexOf(e.sourceNode),nodes.indexOf(e.endNode),e.weight.toInt()))
    }
    for (e in dualEdges){

gr.add(Edge(nodes.indexOf(e.edge1.sourceNode),nodes.indexOf(e.edge1.endNode),e.edge1.we
ight.toInt()))

gr.add(Edge(nodes.indexOf(e.edge2.sourceNode),nodes.indexOf(e.edge2.endNode),e.edge2.we
ight.toInt()))
    }

    val graph = Graph(gr)
    graph.dijkstra(startNode-1) //прогнали алгоритм

    dijkstraAlgorithmController.initStart(startNode,graph.getSnapshotHistory(),graph.getPat
h(startNode)) // здесь принимаю ответ
}

```

```

private fun snapshotToMap(snap:Snapshot):HashMap<Int,List<String>>{
    val list = snap.getAllInfo().substring(1,snap.getAllInfo().length-1).split(",")
    val double = HashMap<Int,List<String>>(list.size)
    for (e in list){
        double[list.indexOf(e)] = e.split(",")
    }
    return double
}

private fun updateAllNodes(snapMap:HashMap<Int,List<String>>){
    // snapMap[0][0] - текущий узел
    // snapMap[1+] - список из 3 элементов, где элемент с индексом 0 - номер
    // вершины, 1 - текущее лучшее расстояние до нее, 2 - номер вершины, из которого пришли в
    // текущую
    nodes[snapMap[0]!![0].toInt()].isActive = true

    for (i in 2..snapMap.size-1){
        nodes[snapMap[i]!![0].toInt()].bestWay = snapMap[i]!![1]
        nodes[snapMap[i]!![0].toInt()].nodeFrom = snapMap[i]!![2]
    }
}

private fun getLogs(snapMap:HashMap<Int,List<String>>):String{
    val logs = StringBuilder("")
    val indexCurNode = snapMap[0]!![0].toInt()
    val isRelax = snapMap[1]!![0].toBoolean()

    logs.append("Текущий узел: ${indexCurNode+1}\n")
    if (isRelax){
        logs.append("Произошла релаксация\n")
    }
    else{
        logs.append("Не произошла релаксация\n")
    }

    logs.append("Лучший путь до узла: ${snapMap[indexCurNode+2]!![1].toInt()}\n")
    logs.append("Предыдущий узел: ${snapMap[indexCurNode+2]!![2].toInt()}\n")

    return logs.toString()
}

fun nextStep(){
    if (!dijkstraAlgorithmController.isNextStepPossible()){
        val logEvent = Event.LogEvent("Следующий шаг невозможен")
        BroadcastPresenter.generateEvent(logEvent)
        return
    }

    val event = Event.LogEvent("Выполнен следующий шаг")
    BroadcastPresenter.generateEvent(event)

    for(n in nodes){
        n.reset()
    }
    val snapMap = snapshotToMap(dijkstraAlgorithmController.getNextStep()?:return)
    //обновляем состояния узлов
    updateAllNodes(snapMap)
}

```

```

        val logEvent = Event.LogEvent(getLogs(snapMap))
        BroadcastPresenter.generateEvent(logEvent)

        //перерисовываем
        graphView.update()
    }

    fun previousStep(){

        if (!dijkstraAlgorithmController.isPreviousStepPossible()){
            val logEvent = Event.LogEvent("Предыдущий шаг невозможен")
            BroadcastPresenter.generateEvent(logEvent)
            return
        }

        val event = Event.LogEvent("Выполнен предыдущий шаг")
        BroadcastPresenter.generateEvent(event)

        for(n in nodes){
            n.reset()
        }

        val snapMap =
        snapshotToMap(dijkstraAlgorithmController.getPreviousStep():return)
        updateAllNodes(snapMap)

        val logEvent = Event.LogEvent(getLogs(snapMap))
        BroadcastPresenter.generateEvent(logEvent)

        graphView.update()
    }

    fun finishAlgorithm(){
        for(n in nodes){
            n.reset()
        }
        val snapMap = snapshotToMap(dijkstraAlgorithmController.getLast())
        updateAllNodes(snapMap)
        graphView.displayDijkstraAlgorithmResult(dijkstraAlgorithmController.answer)
        graphView.update()
    }
}

private fun convertEdgeInfoInArrayList(info_ :String) : ArrayList<List<String>>?{
    val string = info_.replace(" ", "")
    val reg = (Regex("^(\\(\\d+,\\d+,\\d+\\),)*\\(\\d+,\\d+,\\d+\\)\\$"))
    if (!string.matches(reg)) return null

    val match = Regex("(\\d+,\\d+,\\d+)").findAll(string)

    val list = ArrayList<List<String>>()
    for(e in match){
        val temp = e.destructured.toList().toString()
        list.add(temp.substring(1,temp.length-1).split(","))
    }
    return list
}

```

```

private fun convertNodeInfoinArrayList(info_ :String) : ArrayList<List<String>>?{
    val string = info_.replace(" ", "")
    val reg = (Regex("^((\\d+,\\d+\\d+),)*\\((\\d+,\\d+\\d+)\\)$"))
    if (!string.matches(reg)) return null

    val match = Regex("(\\d+,\\d+)").findAll(string)

    val list = ArrayList<List<String>>()
    for(e in match){
        val temp = e.destructured.toList().toString()
        list.add(temp.substring(1,temp.length-1).split(","))
    }
    return list
}

fun downloadGraph(fileName:String){
    val inputStream: InputStream = File(fileName).inputStream()

    val allInfo = mutableListOf<String>()
    inputStream.bufferedReader().forEachLine { allInfo.add(it) }

    if(allInfo.size !in 1..2) return //если в файле больше, чем нужно строк

    val nodeInfoList = (convertNodeInfoinArrayList(allInfo[0].replace(" ", ""))) ?:
return //ошибка

    var edgeInfoList : ArrayList<List<String>>? = null

    if (allInfo.size==2) //нет информации о ребрах
        edgeInfoList = (convertEdgeInfoinArrayList(allInfo[1].replace(" ", ""))) ?:
return //ошибка

    //инициализируем UI граф

    nodes.clear()
    edges.clear()
    dualEdges.clear()
    //инициализируем вершины
    for (n in nodeInfoList){
        addNode(UINode(Coordinate(n[0].toInt(),n[1].toInt())))
    }
    if(edgeInfoList!=null) //если есть информация о ребрах
        for (e in edgeInfoList) {
            addEdge(UIEdge(nodes[e[0].toInt()], nodes[e[1].toInt()], e[2]))
        }

    graphView.update()
}

fun saveGraph(fileName:String){
    //создаем граф из строки
    val graphAsString = StringBuilder("")
    for (n in nodes)
        graphAsString.append("(${n.toString()}), ")

    if(graphAsString.isEmpty())

```



```

        return //нет узлов

        graphAsString.delete(graphAsString.length - 2, graphAsString.length).append("")
        graphAsString.append("\n")
        for (e in edges)
            graphAsString.append("({nodes.indexOf(e.sourceNode)}, $
{nodes.indexOf(e.endNode)}, ${e.weight}), ")
            for (e in dualEdges){
                graphAsString.append("({nodes.indexOf(e.edge1.sourceNode)}, $
{nodes.indexOf(e.edge1.endNode)}, ${e.edge1.weight}), ")
                graphAsString.append("({nodes.indexOf(e.edge2.sourceNode)}, $
{nodes.indexOf(e.edge2.endNode)}, ${e.edge2.weight}), ")
            }

            if(!(edges.isEmpty() && dualEdges.isEmpty()))
                graphAsString.delete(graphAsString.length - 2,
graphAsString.length).append("")

            //Записываем в файл
            val resultFile = File(fileName)
            Files.write(resultFile.toPath(), graphAsString.toString().toByteArray(),
StandardOpenOption.CREATE)

        }
    }
}

```

ПРИЛОЖЕНИЕ К

ИСХОДНЫЙ КОД TOOLBARPRESENTER.KT

```
PACKAGE presenters

import java.util.*
import javax.swing.JFileChooser

enum class ToolbarViewElement {
    START,
    SAVE,
    DOWNLOAD,
    CLEAR,
    NEXT_STEP,
    PREVIOUS_STEP,
    GENERATE,
    END_ALGORITHM
}

interface ToolbarView {
    fun lockElement(element: ToolbarViewElement)
    fun unlockElement(element: ToolbarViewElement)
    fun getFilePath() : String?
    fun showLog(logMsg: String)
}

class ToolbarPresenter(private val toolbarView: ToolbarView) : EventSubscriber {

    fun chainToolbarEvent(event: Event) = BroadcastPresenter.generateEvent(event)

    init {
        BroadcastPresenter.registerSubscriber(this)
    }

    override fun handleEvent(event: Event) {
        when (event) {
            is Event.AfterAlgorithmStarted -> {
                toolbarView.unlockElement(ToolbarViewElement.NEXT_STEP)
                toolbarView.unlockElement(ToolbarViewElement.PREVIOUS_STEP)
                toolbarView.unlockElement(ToolbarViewElement.END_ALGORITHM)
            }

            is Event.AfterAlgorithmEnded -> {
                toolbarView.lockElement(ToolbarViewElement.NEXT_STEP)
                toolbarView.lockElement(ToolbarViewElement.PREVIOUS_STEP)
                toolbarView.lockElement(ToolbarViewElement.END_ALGORITHM)
            }

            is Event.LogEvent -> {
                toolbarView.showLog(getCurrentDateString() + "\n" + event.logInfo + "\n")
            }
        }
    }

    fun getCurrentDateString() : String { return Date().toString() }

    fun getFilePath() = toolbarView.getFilePath()
}
```

ПРИЛОЖЕНИЕ Л

ИСХОДНЫЙ КОД COORDINATE.KT

```
PACKAGE views.graphview

data class Coordinate(
    val x: Int,
    val y: Int
) {
    operator fun minus(anotherCoordinate: Coordinate) : Coordinate {
        return Coordinate(x - anotherCoordinate.x, y - anotherCoordinate.y)
    }

    operator fun plus(anotherCoordinate: Coordinate) : Coordinate {
        return Coordinate(x + anotherCoordinate.x, y + anotherCoordinate.y)
    }
}
```

ПРИЛОЖЕНИЕ М

ИСХОДНЫЙ КОД ENDNODEREQUESTPANE.KT

```
PACKAGE views.graphview

import java.awt.GridLayout
import javax.swing.JComboBox
import javax.swing.JLabel
import javax.swing.JPanel
import javax.swing.JTextField

class EndNodeRequestPane(numbersRange: ArrayList<Int>) : JPanel() {

    val startNodeNumber = JComboBox(numbersRange.toArray())

    init {
        layout = GridLayout(2, 2)

        val startNodeTextField = JLabel("Начальная вершина: ")

        add(startNodeTextField)
        add(startNodeNumber)
    }
}
```

ПРИЛОЖЕНИЕ Н

ИСХОДНЫЙ КОД GRAPHMATHPROVIDER.KT

```
PACKAGE views.graphview

import views.UIConstants
import kotlin.math.*

class GraphMathProvider {

    data class DoubleCoordinate (
        val x: Double = 0.0,
        val y: Double = 0.0
    ) {

        operator fun minus(anotherCoordinate: DoubleCoordinate) : DoubleCoordinate {
            return DoubleCoordinate(x - anotherCoordinate.x, y - anotherCoordinate.y)
        }

        operator fun plus(anotherCoordinate: DoubleCoordinate) : DoubleCoordinate {
            return DoubleCoordinate(x + anotherCoordinate.x, y + anotherCoordinate.y)
        }

        operator fun times(multiplier: Int): DoubleCoordinate {
            return DoubleCoordinate(x * multiplier, y * multiplier)
        }

        fun toInt() : Coordinate {
            return Coordinate(x.toInt(), y.toInt())
        }
    }

    fun isPointInsideEdgeRectangle(edge: UIEdge, pointCoordinate: Coordinate) : Boolean
    {
        val x1 = edge.sourceNode.coordinate.x
        val y1 = edge.sourceNode.coordinate.y

        val x2 = edge.endNode.coordinate.x
        val y2 = edge.endNode.coordinate.y

        //ax + by + c = 0 - уравнение прямой
        val a = y1-y2
        val b = x2-x1
        val c = x1*y2-x2*y1

        val x0 = pointCoordinate.x
        val y0 = pointCoordinate.y

        val distanceToLine = abs(a*x0 + b*y0 + c) / sqrt(a.toDouble().pow(2) +
        b.toDouble().pow(2))

        if(distanceToLine <= UIConstants.edgeWidth) {
            val maxX = if(x1 >= x2) x1 else x2
            val maxY = if(y1 >= y2) y1 else y2
            val minX = if(x1 >= x2) x2 else x1
            val minY = if(y1 >= y2) y2 else y1

            if(x0 in minX..maxX && y0 in minY..maxY) { return true }
        }
    }
}
```

```

    }

    return false
}

fun isPointInsideEdgeRectangle(edgeStart: Coordinate, edgeEnd: Coordinate,
pointCoordinate: Coordinate) : Boolean {
    val x1 = edgeStart.x
    val y1 = edgeStart.y

    val x2 = edgeEnd.x
    val y2 = edgeEnd.y

    //ax + by + c = 0 - уравнение прямой
    val a = y1-y2
    val b = x2-x1
    val c = x1*y2-x2*y1

    val x0 = pointCoordinate.x
    val y0 = pointCoordinate.y

    val distanceToLine = abs(a*x0 + b*y0 + c) / sqrt(a.toDouble().pow(2) +
b.toDouble().pow(2))

    if(distanceToLine <= UIConstants.edgeWidth) {
        val maxX = if(x1 >= x2) x1 else x2
        val maxY = if(y1 >= y2) y1 else y2
        val minX = if(x1 >= x2) x2 else x1
        val minY = if(y1 >= y2) y2 else y1

        if(x0 in minX..maxX && y0 in minY..maxY) { return true }
    }

    return false
}

fun calculateDualEdgeCoordinates(dualEdge: UIDualEdge)
    : Pair<Pair<Coordinate, Coordinate>, Pair<Coordinate,Coordinate>> {
    val x1 = dualEdge.edge1.sourceNode.coordinate.x
    val y1 = dualEdge.edge1.sourceNode.coordinate.y

    val x2 = dualEdge.edge1.endNode.coordinate.x
    val y2 = dualEdge.edge1.endNode.coordinate.y

    val twoPoints = findPointOnCircle(DoubleCoordinate(x1.toDouble(),
y1.toDouble()),
        DoubleCoordinate(x2.toDouble(), y2.toDouble()),
dualEdge.edge1.endNode.radius.toDouble())

    val point11Rotated = rotatePointRelativelyToAnotherPoint(twoPoints.first,
DoubleCoordinate(x1.toDouble(), y1.toDouble()),
UIConstants.dualEdgeRotationAngleInGrads)

    val point12Rotated = rotatePointRelativelyToAnotherPoint(twoPoints.first,
DoubleCoordinate(x1.toDouble(), y1.toDouble()),
-UIConstants.dualEdgeRotationAngleInGrads)

    val point21Rotated = rotatePointRelativelyToAnotherPoint(twoPoints.second,
DoubleCoordinate(x2.toDouble(), y2.toDouble()),
UIConstants.dualEdgeRotationAngleInGrads)
    val point22Rotated = rotatePointRelativelyToAnotherPoint(twoPoints.second,

```

```

        DoubleCoordinate(x2.toDouble(), y2.toDouble()),
        -UIConstants.dualEdgeRotationAngleInGrads)

    return Pair(Pair(point11Rotated.toInt(), point12Rotated.toInt()),
Pair(point21Rotated.toInt(), point22Rotated.toInt()))
}

private fun findPointOnCircle(
    circleCenter1: DoubleCoordinate,
    circleCenter2: DoubleCoordinate, radius: Double) : Pair<DoubleCoordinate,
DoubleCoordinate>
    //first это координаты на 1 ок-ти, second на 2.
    {
        val x1 = circleCenter1.x
        val y1 = circleCenter1.y

        val x2 = circleCenter2.x
        val y2 = circleCenter2.y

        val len1 = abs(x1-x2)
        val len2 = abs(y1-y2)

        val hypotenuselen = sqrt(len1.pow(2) + len2.pow(2))

        val angleSin: Double = len2 / hypotenuselen
        val angleCos: Double = sqrt(1 - angleSin.pow(2))

        val xOffset = radius * angleCos
        val yOffset = radius * angleSin

        var pointOnCircle1 = DoubleCoordinate()
        var pointOnCircle2 = DoubleCoordinate()

        when(determineRelativeQuarter(
            Coordinate(x2.toInt(), y2.toInt()),
            Coordinate(x1.toInt(), y1.toInt()) ))
        {
            1 -> {
                pointOnCircle1 = DoubleCoordinate(x1 - xOffset, y1 - yOffset)
                pointOnCircle2 = DoubleCoordinate(x2 + xOffset, y2 + yOffset)
            }

            2 -> {
                pointOnCircle1 = DoubleCoordinate(x1 + xOffset, y1 - yOffset)
                pointOnCircle2 = DoubleCoordinate(x2 - xOffset, y2 + yOffset)
            }

            3 -> {
                pointOnCircle1 = DoubleCoordinate(x1 + xOffset, y1 + yOffset)
                pointOnCircle2 = DoubleCoordinate(x2 - xOffset, y2 - yOffset)
            }

            4 -> {
                pointOnCircle1 = DoubleCoordinate(x1 - xOffset, y1 + yOffset)
                pointOnCircle2 = DoubleCoordinate(x2 + xOffset, y2 - yOffset)
            }
        }

        return Pair(pointOnCircle1, pointOnCircle2)
    }
}

```

```

        private fun rotatePointRelativelyToAnotherPoint(rotatingPoint: DoubleCoordinate,
                                                         centerPoint: DoubleCoordinate,
                                                         angleInRads: Double) :
DoubleCoordinate {
    return DoubleCoordinate(
        centerPoint.x + (rotatingPoint.x - centerPoint.x) * cos(angleInRads) -
        (rotatingPoint.y - centerPoint.y) * sin(angleInRads),
        centerPoint.y + (rotatingPoint.x - centerPoint.x) * sin(angleInRads) +
        (rotatingPoint.y - centerPoint.y) * cos(angleInRads)
    )
}

fun calculateEdgeWeightTextPosition(edge: UIEdge): Coordinate {
    val x1 = edge.sourceNode.coordinate.x
    val y1 = edge.sourceNode.coordinate.y

    val x2 = edge.endNode.coordinate.x
    val y2 = edge.endNode.coordinate.y

    val centerPoint = DoubleCoordinate((x1+x2)/2.0, (y1+y2)/2.0)

    val lineVector = DoubleCoordinate(x1 - centerPoint.x, y1 - centerPoint.y)

    val normalToLineVector =
        if (lineVector.x == 0.0)
            DoubleCoordinate(-1.0, 0.0)
        else
            DoubleCoordinate(-1 * abs(lineVector.y/lineVector.x), 1.0)

    val unitNormalToLineVector = normaliseVector(normalToLineVector)

    val textOffsetVector = unitNormalToLineVector *
        UIConstants.edgeWeightTextOffsetHeight

    return (centerPoint + textOffsetVector).toInt()
}

fun calculateEdgeWeightTextPosition(point1: Coordinate, point2: Coordinate):
Coordinate {
    val x1 = point1.x
    val y1 = point1.y

    val x2 = point2.x
    val y2 = point2.y

    val centerPoint = DoubleCoordinate((x1+x2)/2.0, (y1+y2)/2.0)

    val lineVector = DoubleCoordinate(x1 - centerPoint.x, y1 - centerPoint.y)

    val normalToLineVector =
        if (lineVector.x == 0.0)
            DoubleCoordinate(-1.0, 0.0)
        else
            DoubleCoordinate(-1 * abs(lineVector.y/lineVector.x), 1.0)

    val unitNormalToLineVector = normaliseVector(normalToLineVector)

    val textOffsetVector = unitNormalToLineVector *
        UIConstants.edgeWeightTextOffsetHeight

    return (centerPoint + textOffsetVector).toInt()
}

```



```

    }

    //Используется уравнение окружности
    fun isPointInsideNodeCircle(pointCoordinate: Coordinate, node: UINode) : Boolean {
        return (((pointCoordinate.x - node.coordinate.x).toDouble()).pow(2.0) +
            ((pointCoordinate.y - node.coordinate.y).toDouble()).pow(2.0)
            <= node.radius.toDouble().pow(2.0))
    }

    fun calculateArrowForEdge(startPoint: Coordinate, endPoint: Coordinate) :
    UIEdgeArrow {
        val x1 = startPoint.x
        val y1 = startPoint.y

        val x2 = endPoint.x
        val y2 = endPoint.y

        val point1 = DoubleCoordinate(x2.toDouble(), y2.toDouble())

        val mainLineVector = DoubleCoordinate(x1.toDouble(), y1.toDouble()) - point1
        val mainLineUnitVector = normaliseVector(mainLineVector)

        val widthOffsetVector = mainLineUnitVector * UIConstants.arrowWidth

        val widthPoint = point1 + widthOffsetVector

        //перпендикулярный вектор
        val heightVector = if (mainLineUnitVector.x == 0.0)
            DoubleCoordinate(1.0, 0.0)
        else
            if(mainLineUnitVector.y == 0.0)
                DoubleCoordinate(0.0, 1.0)
            else
                DoubleCoordinate(
                    -1 * (mainLineUnitVector.y/mainLineUnitVector.x), 1.0)

        val heightUnitVector = normaliseVector(heightVector)

        val heightOffsetVector = heightUnitVector * UIConstants.arrowHeight

        val point2 = widthPoint + heightOffsetVector
        val point3 = widthPoint - heightOffsetVector

        return UIEdgeArrow(point1.toInt(), point2.toInt(), point3.toInt())
    }

    fun calculateEdgeArrow(edge: UIEdge) : UIEdgeArrow {
        val x1 = edge.sourceNode.coordinate.x
        val y1 = edge.sourceNode.coordinate.y

        val x2 = edge.endNode.coordinate.x
        val y2 = edge.endNode.coordinate.y

        val len1 = abs(x1-x2)
        val len2 = abs(y1-y2)

        val hypotenuseLen = sqrt(len1.toDouble().pow(2) + len2.toDouble().pow(2))

        val angleSin: Double = len2 / hypotenuseLen
        val angleCos: Double = sqrt(1 - angleSin.pow(2))
    }

```

```

        val radius = edge.endNode.radius

        val xOffset = radius * angleCos
        val yOffset = radius * angleSin

        var point1 = DoubleCoordinate()

        when(determineRelativeQuarter(edge.endNode.coordinate,
            edge.sourceNode.coordinate))
        {
            1 -> {
                point1 = DoubleCoordinate(x2 + xOffset, y2 + yOffset)
            }

            2 -> {
                point1 = DoubleCoordinate(x2-xOffset, y2 + yOffset)
            }

            3 -> {
                point1 = DoubleCoordinate(x2-xOffset, y2 - yOffset)
            }

            4 -> {
                point1 = DoubleCoordinate(x2+xOffset, y2 - yOffset)
            }
        }

        val mainLineVector = DoubleCoordinate(x1.toDouble(), y1.toDouble()) - point1
        val mainLineUnitVector = normaliseVector(mainLineVector)

        val widthOffsetVector = mainLineUnitVector * UIConstants.arrowWidth

        val widthPoint = point1 + widthOffsetVector

        //перпендикулярный вектор
        val heightVector = if(mainLineUnitVector.x == 0.0)
            DoubleCoordinate(1.0, 0.0)
        else
            DoubleCoordinate(
                -1 * (mainLineUnitVector.y/mainLineUnitVector.x), 1.0
            )

        val heightUnitVector = normaliseVector(heightVector)

        val heightOffsetVector = heightUnitVector * UIConstants.arrowHeight

        val point2 = widthPoint + heightOffsetVector
        val point3 = widthPoint - heightOffsetVector

        return UIEdgeArrow(point1.toInt(), point2.toInt(), point3.toInt())
    }

    //Возвращает единичный вектор по заданному
    private fun normaliseVector(vector: DoubleCoordinate) : DoubleCoordinate {
        val vectorLen = sqrt(vector.x.pow(2) + vector.y.pow(2))
        if(vectorLen == 0.0) return DoubleCoordinate(1.0, 1.0)
        return DoubleCoordinate(vector.x / vectorLen, vector.y / vectorLen)
    }

    //Определяет четверть в которой находится relativePoint по отношению к centerPoint

```

```

        private fun determineRelativeQuarter(centerPoint: Coordinate, relativePoint:
Coordinate) : Int
        {
            val centerX = centerPoint.x
            val centerY = centerPoint.y

            val x = relativePoint.x
            val y = relativePoint.y

            if (x >= centerX && y >= centerY) return 1
            if (x < centerX && y >= centerY) return 2
            if (x <= centerX && y < centerY) return 3
            return if (x > centerX && y < centerY) 4
            else 1
        }

        fun areNodesColliding(node1: UINode, node2: UINode) : Boolean {
            val len1 = abs(node1.coordinate.x-node2.coordinate.x)
            val len2 = abs(node1.coordinate.y-node2.coordinate.y)

            val distance = sqrt(len1.toDouble().pow(2) + len2.toDouble().pow(2))

            //просто чтобы не рисовать рёбра когда не надо
            return distance < (UIConstants.circleStrokeWidth + UIConstants.circleRadius)
        }
    }
}

```

ПРИЛОЖЕНИЕ О

ИСХОДНЫЙ КОД GRAPHSHEET.KT

```
PACKAGE views.graphview

import presenters.GraphView
import presenters.MainPresenter
import views.UIConstants
import java.awt.*
import java.awt.event.MouseEvent
import java.awt.event.MouseListener
import java.awt.event.MouseMotionListener
import java.awt.geom.Ellipse2D
import java.lang.NumberFormatException
import javax.swing.*

interface GraphViewObserver {
    fun onSheetDragged(offsetX: Int, offsetY: Int)
}

class GraphSheet: JPanel(), MouseListener, MouseMotionListener, GraphView {

    init {
        background = Color.WHITE

        addMouseListener(this)
        addMouseMotionListener(this)
    }

    private val presenter = MainPresenter(this)

    override fun update() = repaint()

    var sheetDraggingObserver: GraphViewObserver? = null

    private val nodes = presenter.nodes
    private val edges = presenter.edges
    private val dualEdges = presenter.dualEdges

    private val mathProvider = GraphMathProvider()

    private var isAlgorithmRunning = false

    private var currentGraphViewState: GraphViewState = GraphViewState.DefaultState
    set(newGraphViewState) {
        when(newGraphViewState)
        {
            GraphViewState.DefaultState -> {
                resetCursorToArrow()
            }

            is GraphViewState.SheetMovingState -> {
                setCursorHand()
            }

            is GraphViewState.NodeDraggingState -> {
                setCursorHand()
            }
        }
    }
}
```

```

        }
        field = newGraphViewState
    }

    override fun paintComponent(graphics: Graphics) {
        super.paintComponent(graphics)

        val graphics2D = graphics as Graphics2D

        graphics2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON)

        when(currentGraphViewState)
        {
            is GraphViewState.CreatingEdgeState, is
GraphViewState.CreatingEdgeAndSheetMovingState -> {
                val currentCreatingEdgeState: GraphViewState.CreatingEdgeState =
                    if(currentGraphViewState is GraphViewState.CreatingEdgeState)
                        currentGraphViewState as GraphViewState.CreatingEdgeState
                    else (currentGraphViewState as
GraphViewState.CreatingEdgeAndSheetMovingState).creatingEdgeState

                drawBuildingEdge(currentCreatingEdgeState.creatingEdge, graphics2D)
            }
        }
        //Сначала рисуем рёбра, чтобы вершины были сверху них.

        edges.forEach {
            if (!mathProvider.areNodesColliding(it.sourceNode, it.endNode))
                drawEdge(it, graphics2D)
        }

        dualEdges.forEach {
            if (!mathProvider.areNodesColliding(it.edge1.sourceNode, it.edge1.endNode))
                drawDualEdge(it, graphics2D)
        }

        var nodeIndex = 1
        nodes.forEach {
            drawNode(it, nodeIndex, graphics2D)
            nodeIndex++
        }
    }

    private fun drawNode(node: UINode, nodeNumber: Int, panelGraphics: Graphics2D) {
        val strokeCircle = Ellipse2D.Double((node.coordinate.x-node.radius).toDouble(),
            (node.coordinate.y-node.radius).toDouble(),
            (2*node.radius).toDouble(),
            (2*node.radius).toDouble())

        val graphicsCircle = Ellipse2D.Double((node.coordinate.x-
UIConstants.circleRadius).toDouble(),
            (node.coordinate.y-UIConstants.circleRadius).toDouble(),
            (2*UIConstants.circleRadius).toDouble(),
            (2*UIConstants.circleRadius).toDouble())

        val nodeNumberString = nodeNumber.toString()
        panelGraphics.font = UIConstants.nodeTextFont
        val fontMetrics = panelGraphics.fontMetrics
        val textWidth = fontMetrics.getStringBounds(nodeNumberString,

```

```

panelGraphics).width.toInt()
    val textHeight = fontMetrics.getStringBounds(nodeNumberString,
panelGraphics).height.toInt()

    panelGraphics.color = UIConstants.nodeStrokeFillColor
    panelGraphics.fill(strokeCircle)

    if(node.isActive) {
        panelGraphics.color = UIConstants.nodeActiveFillColor
    }
    else panelGraphics.color = UIConstants.nodeFillColor

    panelGraphics.fill(graphicsCircle)

    panelGraphics.color = UIConstants.textColor
    panelGraphics.drawString(nodeNumberString,
        node.coordinate.x - textWidth/2,
        node.coordinate.y + textHeight/4)

    if(isAlgorithmRunning) {
        val sign =
            "[${if (node.bestWay.isEmpty()) "-" else node.bestWay} / ${if
(node.nodeFrom.isEmpty()) "-" else node.nodeFrom}]"
        panelGraphics.font = UIConstants.nodeSignTextFont
        val signHeight = fontMetrics.getStringBounds(sign,
panelGraphics).height.toInt()
        val signWidth = fontMetrics.getStringBounds(sign,
panelGraphics).width.toInt()
        panelGraphics.drawString(
            sign,
            node.coordinate.x - signWidth/5,
            node.coordinate.y + node.radius + signHeight/2
        )
    }
}

private fun drawEdge(edge: UIEdge, panelGraphics: Graphics2D)
{
    panelGraphics.stroke = BasicStroke((edge.width/2).toFloat())
    panelGraphics.color = UIConstants.edgeColor
    panelGraphics.drawLine(edge.sourceNode.coordinate.x,
        edge.sourceNode.coordinate.y,
        edge.endNode.coordinate.x,
        edge.endNode.coordinate.y)

    val arrow = mathProvider.calculateEdgeArrow(edge)

    val textCoordinate = mathProvider.calculateEdgeWeightTextPosition(edge)
    panelGraphics.font = UIConstants.edgeWeightTextFont
    panelGraphics.drawString(edge.weight,
        textCoordinate.x, textCoordinate.y)

    drawEdgeArrow(arrow, panelGraphics)
}

private fun drawEdgeArrow(arrow: UIEdgeArrow, panelGraphics: Graphics2D)
{
    panelGraphics.drawLine(arrow.point2.x,
        arrow.point2.y,
        arrow.point1.x,

```

```

        arrow.point1.y)

    panelGraphics.drawLine(arrow.point3.x,
        arrow.point3.y,
        arrow.point1.x,
        arrow.point1.y)
}

private fun drawDualEdge(dualEdge: UIDualEdge, panelGraphics: Graphics2D){
    panelGraphics.stroke = BasicStroke((UIConstants.edgeWidth/2).toFloat())
    panelGraphics.color = UIConstants.edgeColor

    val coordinates = mathProvider.calculateDualEdgeCoordinates(dualEdge)
    val arrow1 = mathProvider.calculateArrowForEdge(
        coordinates.first.first, coordinates.second.second
    )
    val weightPosition1 =
mathProvider.calculateEdgeWeightTextPosition(coordinates.first.first,
        coordinates.second.second)
    panelGraphics.drawLine(
        coordinates.first.first.x,
        coordinates.first.first.y,
        coordinates.second.second.x,
        coordinates.second.second.y
    )
    drawEdgeArrow(arrow1, panelGraphics)
    panelGraphics.font = UIConstants.edgeWeightTextFont
    panelGraphics.drawString(dualEdge.edge1.weight,
        weightPosition1.x, weightPosition1.y)

    val arrow2 = mathProvider.calculateArrowForEdge(
        coordinates.second.first,
        coordinates.first.second
    )
    val weightPosition2 =
mathProvider.calculateEdgeWeightTextPosition(coordinates.first.second,
        coordinates.second.first)
    panelGraphics.drawLine(
        coordinates.first.second.x,
        coordinates.first.second.y,
        coordinates.second.first.x,
        coordinates.second.first.y
    )
    drawEdgeArrow(arrow2, panelGraphics)
    panelGraphics.drawString(dualEdge.edge2.weight,
        weightPosition2.x, weightPosition2.y)
}

private fun drawBuildingEdge(buildingEdge: UIBuildingEdge, panelGraphics:
Graphics2D)
{
    panelGraphics.stroke = BasicStroke((buildingEdge.width/2).toFloat())
    panelGraphics.color = UIConstants.edgeColor
    panelGraphics.drawLine(buildingEdge.startCoordinate.x,
        buildingEdge.startCoordinate.y,
        buildingEdge.endCoordinate.x,
        buildingEdge.endCoordinate.y)
}

override fun mouseReleased(mouseEvent: MouseEvent) {

```

```

when(currentGraphViewState)
{
    is GraphViewState.CreatingEdgeState -> {
        //ничего не делаем, т.к. mouseClicked обрабатывается ПОСЛЕ mouseReleased
    }

    is GraphViewState.CreatingEdgeAndSheetMovingState -> {
        currentGraphViewState = (currentGraphViewState as
GraphViewState.CreatingEdgeAndSheetMovingState)
            .creatingEdgeState
    }

    is GraphViewState.NodeDraggingState, is GraphViewState.SheetMovingState ->
    { currentGraphViewState = GraphViewState.DefaultState}

    else -> {
        currentGraphViewState = GraphViewState.DefaultState
    }
}

}

override fun mouseEntered(mouseEvent: MouseEvent) { }

//Нажатие любой кнопки мышки (и колёсика)
@ExperimentalUnsignedTypes
override fun mouseClicked(mouseEvent: MouseEvent) {
    when(mouseEvent.clickCount) {

        //одиночное нажатие
        1 -> {
            when(mouseEvent.button) {
                //Правая кнопка
                MouseEvent.BUTTON3 -> {
                    when(currentGraphViewState)
                    {
                        is GraphViewState.DefaultState -> {
                            if(isAlgorithmRunning) return

                            val node: UINode? =
findNodeUnderMouse(Coordinate(mouseEvent.x, mouseEvent.y))
                            if(node != null)
                                createAndShowPopupMenuOnNode(mouseEvent, node)

                            else {
                                val edge: UIEdge? =
findEdgeUnderMouse(Coordinate(mouseEvent.x, mouseEvent.y))
                                edge ?: return

                                createAndShowPopupMenuOnEdge(mouseEvent, edge)
                            }
                        }
                    }
                }
            }
        }

        //Левая кнопка
        MouseEvent.BUTTON1 -> {

            when(currentGraphViewState)

```



```

        {
            is GraphViewState.CreatingEdgeState -> {
                val node: UINode? =
findNodeUnderMouse(Coordinate(mouseEvent.x, mouseEvent.y))
                if(node == null) {
                    currentGraphViewState = GraphViewState.DefaultState
                    repaint()
                    return
                }

                val currentCreatingEdgeState = currentGraphViewState as
GraphViewState.CreatingEdgeState

                addEdge(currentCreatingEdgeState.sourceNode, node,
currentCreatingEdgeState.edgeWeight.toString())

                currentGraphViewState = GraphViewState.DefaultState
            }
        }
    }

    //Двойной клик
    2 -> {
        //Левая кнопка
        if(mouseEvent.button == MouseEvent.BUTTON1)
            if(currentGraphViewState is GraphViewState.DefaultState)
            {
                if(isAlgorithmRunning) return
                else {
                    if(findNodeUnderMouse(Coordinate(mouseEvent.x,
mouseEvent.y)) == null)
                        presenter.addNode(UINode(Coordinate(mouseEvent.x,
mouseEvent.y)))
                }
            }
    }

    override fun mouseExited(mouseEvent: MouseEvent) { }

    override fun mousePressed(mouseEvent: MouseEvent) { }

    override fun mouseMoved(mouseEvent: MouseEvent) {
        if(findNodeUnderMouse(Coordinate(mouseEvent.x, mouseEvent.y)) != null)
        {
            setCursorHand()
        }
        else resetCursorToArrow()

        updateCreatingEdge(Coordinate(mouseEvent.x, mouseEvent.y))
    }

    override fun mouseDragged(mouseEvent: MouseEvent) {
        val dragCoordinate = Coordinate(mouseEvent.x, mouseEvent.y)

        if(SwingUtilities.isRightMouseButton(mouseEvent)){

```

```

        if (currentGraphViewState == GraphViewState.DefaultState ||
            currentGraphViewState is GraphViewState.SheetMovingState ||
            currentGraphViewState is GraphViewState.CreatingEdgeState ||
            currentGraphViewState is
GraphViewState.CreatingEdgeAndSheetMovingState)
            onRightMouseButtonDragging(dragCoordinate)
    }

    if(SwingUtilities.isLeftMouseButton(mouseEvent)) {
        if (currentGraphViewState == GraphViewState.DefaultState ||
            currentGraphViewState is GraphViewState.NodeDraggingState)
            onLeftMouseButtonDragging(dragCoordinate)
    }

    repaint()
}

private fun onRightMouseButtonDragging(dragCoordinate: Coordinate){
    when(currentGraphViewState)
    {
        GraphViewState.DefaultState, is GraphViewState.CreatingEdgeState-> {

            currentGraphViewState =
                if(currentGraphViewState is GraphViewState.DefaultState)
                    GraphViewState.SheetMovingState(dragCoordinate)
                else
                    GraphViewState.CreatingEdgeAndSheetMovingState(
                        currentGraphViewState as GraphViewState.CreatingEdgeState,
                        GraphViewState.SheetMovingState(dragCoordinate))

        }

        is GraphViewState.SheetMovingState, is
GraphViewState.CreatingEdgeAndSheetMovingState -> {

            val currentSheetMovingState: GraphViewState.SheetMovingState =
                if (currentGraphViewState is GraphViewState.SheetMovingState)
                    currentGraphViewState as GraphViewState.SheetMovingState
                else (currentGraphViewState as
GraphViewState.CreatingEdgeAndSheetMovingState).sheetMovingState

            val offsetX = dragCoordinate.x -
currentSheetMovingState.draggingStartPoint.x
            val offsetY = dragCoordinate.y -
currentSheetMovingState.draggingStartPoint.y

            sheetDraggingObserver?.onSheetDragged(offsetX, offsetY)
            updateCreatingEdge(dragCoordinate)
        }
    }
}

private fun onLeftMouseButtonDragging(dragCoordinate: Coordinate) {
    when(currentGraphViewState)
    {
        is GraphViewState.NodeDraggingState -> {
            val draggableState = currentGraphViewState as
GraphViewState.NodeDraggingState

            draggableState.draggingNode.coordinate = dragCoordinate +

```

```

draggableState.draggingOffset
    }

    is GraphViewState.DefaultState -> {
        val node: UINode? = findNodeUnderMouse(dragCoordinate)

        if(node == null) {
            currentGraphViewState = GraphViewState.EmptyDraggingState
            return
        }

        val coordinateOffset = node.coordinate - dragCoordinate
        currentGraphViewState =
GraphViewState.NodeDraggingState(coordinateOffset, node)
    }

    }
}

private fun findNodeUnderMouse(cursorCoordinate: Coordinate): UINode? {
    nodes.forEach() {
        if(mathProvider.isPointInsideNodeCircle(cursorCoordinate, it))
        {
            return it
        }
    }
    return null
}

private fun findEdgeUnderMouse(cursorCoordinate: Coordinate) : UIEdge? {
    edges.forEach {
        if(mathProvider.isPointInsideEdgeRectangle(it, cursorCoordinate))
            return it
    }

    dualEdges.forEach {
        val edgesPoints = mathProvider.calculateDualEdgeCoordinates(it)

        if(mathProvider.isPointInsideEdgeRectangle(
            edgesPoints.first.first, edgesPoints.second.second,
cursorCoordinate))
            return it.edge1

        if(mathProvider.isPointInsideEdgeRectangle(
            edgesPoints.first.second, edgesPoints.second.first,
cursorCoordinate))
            return it.edge2
    }

    return null
}

private fun addEdge(sourceNode: UINode, endNode: UINode, edgeWeight: String) {
    presenter.addEdge(UIEdge(sourceNode, endNode, edgeWeight))
    repaint()
}

private fun updateCreatingEdge(newEndCoordinate: Coordinate) {
    when(currentGraphViewState) {
        is GraphViewState.CreatingEdgeState, is
GraphViewState.CreatingEdgeAndSheetMovingState -> {

```

```

        val currentCreatingEdgeState: GraphViewState.CreatingEdgeState =
            if (currentGraphViewState is GraphViewState.CreatingEdgeState)
                currentGraphViewState as GraphViewState.CreatingEdgeState
            else (currentGraphViewState as
GraphViewState.CreatingEdgeAndSheetMovingState).creatingEdgeState

        currentCreatingEdgeState.creatingEdge.endCoordinate = newEndCoordinate
        repaint()
    }
}

private fun setCursorHand() = run { cursor =
Cursor.getPredefinedCursor(Cursor.HAND_CURSOR) }
private fun resetCursorToArrow() = run { cursor = Cursor.getDefaultCursor() }
@ExperimentalUnsignedTypes
private fun createAndShowPopupMenuOnNode(sourceMouseEvent: MouseEvent,
affectedNode: UINode) {
    val popupMenu = JPopupMenu()

    val addEdgeItem = JMenuItem("Добавить ребро")
    addEdgeItem.addActionListener {
        val edgeWeightString: String? = JOptionPane.showInputDialog(null, "Введите
вес ребра",
            "Ввод веса ребра", JOptionPane.QUESTION_MESSAGE)
        if(edgeWeightString != null) {
            try {
                val edgeWeight: UInt = edgeWeightString.toUInt()
                currentGraphViewState =
GraphViewState.CreatingEdgeState(affectedNode, edgeWeight.toInt())
            } catch (exception: NumberFormatException) {
                JOptionPane.showMessageDialog(
                    null, "Вы ввели некорректное значение веса ребра",
                    "Ошибка ввода", JOptionPane.ERROR_MESSAGE
                )
            }
        }
    }
    popupMenu.add(addEdgeItem)
    val removeNodeItem = JMenuItem("Удалить вершину")
    removeNodeItem.addActionListener { presenter.deleteNode(affectedNode) }
    popupMenu.add(removeNodeItem)
    popupMenu.show(sourceMouseEvent.component, sourceMouseEvent.x,
sourceMouseEvent.y)
}
private fun createAndShowPopupMenuOnEdge(sourceMouseEvent: MouseEvent,
affectedEdge: UIEdge)
{
    val popupMenu = JPopupMenu()

    val removeEdgeItem = JMenuItem("Удалить ребро")
    removeEdgeItem.addActionListener { presenter.deleteEdge(affectedEdge)}
    popupMenu.add(removeEdgeItem)
    popupMenu.show(sourceMouseEvent.component, sourceMouseEvent.x,
sourceMouseEvent.y)
}
override fun displayDijkstraAlgorithmResult(result: String) {
    val resultTextArea = JTextArea(result)
    resultTextArea.isEditable = false
    resultTextArea.lineWrap = true
    val scrollResultField = JScrollPane(resultTextArea)

```

```

        scrollResultField.preferredSize = Dimension(400, 150)
        JOptionPane.showMessageDialog(null, scrollResultField, "Результат алгоритма",
JOptionPane.PLAIN_MESSAGE)

        val response = JOptionPane.showConfirmDialog(null, "Закончить алгоритм?",
            "Подтвердите действие", JOptionPane.YES_NO_OPTION)

        if(response == 0) {
            presenter.onAlgorithmEndConfirmed()
        }
    }
    override fun setAlgorithmRunningFlag(isAlgorithmRunning: Boolean) {
        this.isAlgorithmRunning = isAlgorithmRunning
    }
}

```

ПРИЛОЖЕНИЕ П

ИСХОДНЫЙ КОД GRAPHVIEWHOLDER.KT

```
PACKAGE views.graphview

import presenters.MainPresenter
import views.UIConstants
import java.awt.Dimension
import javax.swing.JScrollPane
import javax.swing.ScrollPaneConstants

class GraphViewHolder() : JScrollPane(GraphSheet()), GraphViewObserver {

    init {
        //достаём переданный graphview
        (viewport.view as GraphSheet).sheetDraggingObserver = this
        viewport.view.preferredSize = Dimension(UIConstants.graphScreenSheetWidth,
        UIConstants.graphScreenSheetHeight)
        horizontalScrollBarPolicy = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS
        verticalScrollBarPolicy = ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS
        verticalScrollBar.unitIncrement = UIConstants.verticalScrollIncrement
        horizontalScrollBar.unitIncrement = UIConstants.horizontalScrollIncrement
    }

    override fun onSheetDragged(offsetX: Int, offsetY: Int) {
        horizontalScrollBar.value -= offsetX
        verticalScrollBar.value -= offsetY
    }
}
```

ПРИЛОЖЕНИЕ Р

ИСХОДНЫЙ КОД GRAPHVIEWSTATE.KT

```
PACKAGE views.graphview

sealed class GraphViewState {
    object DefaultState : GraphViewState()
    class NodeDraggingState(val draggingOffset: Coordinate, val draggingNode: UINode) :
GraphViewState()
    class SheetMovingState(val draggingStartPoint: Coordinate) : GraphViewState()
    object EmptyDraggingState : GraphViewState()

    class CreatingEdgeState(val sourceNode: UINode, val edgeWeight: Int):
GraphViewState() {
        val creatingEdge = UIBuildingEdge(sourceNode.coordinate, sourceNode.coordinate)
    }

    class CreatingEdgeAndSheetMovingState(val creatingEdgeState: CreatingEdgeState,
                                           val sheetMovingState: SheetMovingState) :
GraphViewState()
}
```

ПРИЛОЖЕНИЕ С

ИСХОДНЫЙ КОД UIBUILDINGEDGE.KT

```
PACKAGE views.graphview

import views.UIConstants

class UIBuildingEdge(
    var startCoordinate: Coordinate,
    var endCoordinate: Coordinate
) {
    val width = UIConstants.edgeWidth
}
```


ПРИЛОЖЕНИЕ Т

ИСХОДНЫЙ КОД UIEDGE.KT

```
PACKAGE views.graphview

import views.UIConstants

class UIEdge(
    var sourceNode: UINode,
    var endNode: UINode,
    var weight: String
) {
    val width = UIConstants.edgeWidth
}

data class UIDualEdge(val edge1 : UIEdge, val edge2 : UIEdge)
```

ПРИЛОЖЕНИЕ У

ИСХОДНЫЙ КОД UIEDGEARROW.KT

```
PACKAGE views.graphview

data class UIEdgeArrow (
    val point1 : Coordinate,
    val point2: Coordinate,
    val point3: Coordinate
)
```

ПРИЛОЖЕНИЕ Ф

ИСХОДНЫЙ КОД UINODE.KT

```
PACKAGE views.graphview

import views.UIConstants

class UINode(
    var coordinate: Coordinate
) {
    val radius = UIConstants.circleRadius + UIConstants.circleStrokeWidth
    var isActive: Boolean = false
    var bestWay: String = ""
    var nodeFrom: String = ""

    fun reset() {
        isActive = false
        bestWay = ""
        nodeFrom = ""
    }

    override fun toString() = "${coordinate.x}, ${coordinate.y}"
}
```

ПРИЛОЖЕНИЕ X

ИСХОДНЫЙ КОД GRAPHTEST.KT

```
PACKAGE models

import org.junit.jupiter.api.Assertions
import org.junit.jupiter.api.Assertions.*
import org.junit.jupiter.api.DisplayName
import org.junit.jupiter.api.Test
import org.junit.jupiter.api.function.Executable
import java.util.List
import java.util.*
import javax.xml.stream.events.StartDocument
import kotlin.collections.HashMap
import kotlin.properties.Delegates
import org.junit.jupiter.api.DynamicTest
import org.junit.jupiter.api.DynamicTest.dynamicTest
import org.junit.jupiter.api.TestFactory

internal class GraphTest() {

    lateinit var GRAPH: kotlin.collections.List<Edge>
    var START: Int = 0
    var END: Int = 0

    @org.junit.jupiter.api.BeforeEach
    fun setUp(){
        GRAPH = listOf(
            Edge(1, 2, 7),
            Edge(1, 3, 9),
            Edge(2, 3, 1)
        )
        START = 1
        END = 3
    }

    @org.junit.jupiter.api.AfterEach
    fun tearDown() {
        if(GRAPH.isNotEmpty())
            GRAPH = listOf(
                Edge(1, 1, 0)
            )
    }

    @Test
    fun testCorrectEdges() {
        with(Graph(GRAPH)){
            dijkstra(START)
            assertTrue(edges::isNotEmpty)
            assertEquals(3, edges.size)
        }
    }

    @Test
    fun testCorrectGraph() {
        with(Graph(GRAPH)){
            dijkstra(START)
            assertTrue(graph::isNotEmpty)
        }
    }
}
```

```

        assertEquals(2, graph[1]!!.neighbours.size)
        assertEquals(1, graph[2]!!.neighbours.size)
        assertEquals(0, graph[3]!!.neighbours.size)
        assertTrue{
            graph[1]!!.dist < Int.MAX_VALUE && graph[1]!!.dist >= 0
            graph[2]!!.dist < Int.MAX_VALUE && graph[2]!!.dist >= 0
            graph[3]!!.dist < Int.MAX_VALUE && graph[3]!!.dist >= 0
        }
        assertEquals(0, graph[1]!!.dist)
        assertEquals(8, graph[3]!!.dist) //поменялось минимальное расстояние а
процессе алгоритма
        assertEquals(2, graph[3]!!.previous?.name)
        assertEquals(1, graph[2]!!.previous?.name)
    }
}
}

```

```

internal class IncorrectGraphTest() {

    lateinit var GRAPH: kotlin.collections.List<Edge>
    var START: Int = 0
    var END: Int = 0

    @org.junit.jupiter.api.BeforeEach
    fun setUp(){
        START = 1
        END = 3
    }

    @Test
    fun testCorrectDist() {
        val exception = assertThrows(IllegalArgumentException::class.java) {
            Edge(1, 2, -3)
        }
        assertEquals("Dist < 0", exception.message)
    }

    @Test
    fun testCorrectVertex() {
        val exceptionVertex1 = assertThrows(IllegalArgumentException::class.java) {
            Edge(1, -2, 3)
        }
        val exceptionVertex2 = assertThrows(IllegalArgumentException::class.java) {
            Edge(-1, 2, 3)
        }
        assertEquals("Value < 0", exceptionVertex1.message)
        assertEquals("Value < 0", exceptionVertex2.message)
    }

    @Test
    fun testCorrectStartVertex() {
        GRAPH = listOf(
            Edge(1, 2, 7),
            Edge(1, 3, 9),
            Edge(2, 3, 1)
        )
        START = -1
        END = 3
        with(Graph(GRAPH)) {
            dijkstra(START)
        }
    }
}

```

```

        assertEquals("Граф не содержит стартовую вершину '${START}'", testOutput)
    }
}

@Test
fun testCorrectEndVertex() {
    GRAPH = listOf(
        Edge(1, 2, 7),
        Edge(1, 3, 9),
        Edge(2, 3, 1)
    )
    START = 1
    END = 4
    with(Graph(GRAPH)) {
        printPath(END)
        assertEquals("Граф не содержит конечную вершину '${END}'", testOutput)
    }
}
}

```

```

internal class IncorrectAutoGraphTest() {

    lateinit var GRAPH: kotlin.collections.List<Edge>
    var START: Int = 0
    var END: Int = 0

    @org.junit.jupiter.api.BeforeEach
    fun setUp(){
        GRAPH = listOf(
            Edge(1, 2, 7),
            Edge(1, 3, 9),
            Edge(2, 3, 1)
        )
    }

    @TestFactory
    fun autoTestCorrectEndVertex() = listOf(
        -1 to "Граф не содержит конечную вершину '-1'",
        2 to "",
        6 to "Граф не содержит конечную вершину '6'"
    ).map { (input, expected) ->
        dynamicTest("Конечная вершина $input: $expected") {
            with(Graph(GRAPH)) {
                START = 1
                printPath(input)
                assertEquals(expected, testOutput)
                testOutput = ""
            }
        }
    }

    @TestFactory
    fun autoTestCorrectStartVertex() = listOf(
        -11 to "Граф не содержит стартовую вершину '-11'",
        1 to "",
        66 to "Граф не содержит стартовую вершину '66'"
    ).map { (input, expected) ->
        dynamicTest("Стартовая вершина $input: $expected") {
            with(Graph(GRAPH)) {
                END = 3
            }
        }
    }
}

```

```
        dijkstra(input)
        assertEquals(expected, testOutput)
        testOutput = ""
    }
}
}
```

ПРИЛОЖЕНИЕ Ц

ИСХОДНЫЙ КОД OPTIONALGRAPH.KT

```
PACKAGE models

import java.util.*
import kotlin.collections.HashMap

var testOutput: String = ""

class Edge(v1: Int, v2: Int, dist: Int){
    val v1 = if (v1 < 0) throw IllegalArgumentException("Value < 0") else v1
    val v2 = if (v2 < 0) throw IllegalArgumentException("Value < 0") else v2
    val dist = if (dist < 0) throw IllegalArgumentException("Dist < 0") else dist
}

class Vertex(val name: Int) : Comparable<Vertex> {
    var dist = Int.MAX_VALUE
    var previous: Vertex? = null
    val neighbours = HashMap<Vertex, Int>()
    fun printPath(){
        when(previous){
            this -> print(name)
            null -> print("До вершины $name невозможно добраться")
            else -> {
                previous!!.printPath()
                print(" -> $name($dist)")
            }
        }
    }
    override fun compareTo(other: Vertex): Int {
        if (dist == other.dist) return name.compareTo(other.name)
        return dist.compareTo(other.dist)
    }
}

class Graph(val edges: kotlin.collections.List<Edge>){
    val graph = HashMap<Int, Vertex>(edges.size)
    init {
        for (e in edges) {
            if (!graph.containsKey(e.v1))
                graph.put(e.v1, Vertex(e.v1))
            if (!graph.containsKey(e.v2))
                graph.put(e.v2, Vertex(e.v2))
        }
        for (e in edges)
            graph[e.v1]!!.neighbours.put(graph[e.v2]!!, e.dist)
    }
    fun dijkstra(startName: Int) {
        if (!graph.containsKey(startName)) {
            testOutput = "Граф не содержит стартовую вершину '${startName}'"
            println("Граф не содержит стартовую вершину '${startName}'")
            return
        }
        val source: Vertex? = graph[startName]
        val treeSet: TreeSet<Vertex> = TreeSet<Vertex>()
        for (v in graph.values) {
            v.previous = if (v == source) source else null
        }
    }
}
```



```

        v.dist = if (v == source) 0 else Int.MAX_VALUE
        treeSet.add(v)
    }
    dijkstra(treeSet)
}
private fun dijkstra(treeSet: TreeSet<Vertex>) {
    while (!treeSet.isEmpty()) {
        val currV = treeSet.pollFirst()
        if (currV!!.dist == Int.MAX_VALUE)
            break
        for (ngb in currV.neighbours) {
            val currentNgb = ngb.key
            val alternateDist = currV.dist + ngb.value
            if (alternateDist < currentNgb.dist) {
                treeSet.remove(currentNgb)
                currentNgb.dist = alternateDist
                currentNgb.previous = currV
                treeSet.add(currentNgb)
            }
        }
    }
}
fun printPath(endName: Int) {
    if (!graph.containsKey(endName)) {
        testOutput = "Граф не содержит конечную вершину '${endName}'"
        println("Граф не содержит конечную вершину '${endName}'")
        return
    }
    graph[endName]!!.printPath()
    println()
}
}

```