

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

**ОТЧЕТ**  
**по учебной практике**  
**Тема: Визуализация алгоритма Гольдберга**

Студент гр. 9381	_____	Шахин Н.С.
Студент гр. 9381	_____	Колованов Р.А.
Студентка гр. 9381	_____	Андрух И.А.
Руководитель	_____	Жангиров Т.Р.

Санкт-Петербург  
2021

## ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Шахин Н.С. группы 9381

Студент Колованов Р.А. группы 9381

Студентка Андрух И.А. группы 9381

Тема практики: Визуализация алгоритма Гольдберга

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: Гольдберга

Сроки прохождения практики: 01.06.2021 – 14.06.2021

Дата сдачи отчета: 12.06.2021

Дата защиты отчета: 12.06.2021

Студент	_____	Шахин Н.С.
Студент	_____	Колованов Р.А.
Студентка	_____	Андрух И.А.
Руководитель	_____	Жангиров Т.Р.

## **АННОТАЦИЯ**

Целью данной учебной практики является итеративная разработка GUI приложения для визуализации работы алгоритма Гольдберга, предназначенного для поиска максимального потока в сети. Программа разрабатывается на языке Java с использованием библиотеки JavaFX для реализации GUI. Разработка ведется командой из трех человек, за которыми закреплены определенные роли.

## **SUMMARY**

The purpose of this training practice is the iterative development of a GUI application for visualizing the work of the Goldberg algorithm, designed to find the maximum flow in the network. The program is developed in Java using the JavaFX library for GUI implementation. The development is carried out by a team of three people, who are assigned certain roles.

## СОДЕРЖАНИЕ

	Введение	5
1.	Требования к программе	6
1.1.	Исходные требования к программе	6
2.	План разработки и распределение ролей в бригаде	7
2.1.	План разработки	7
2.2.	Распределение ролей в бригаде	7
3.	Особенности реализации	8
3.1.	Архитектура программы	8
3.2.	Описание алгоритма	12
3.3.	Структуры данных	15
3.4.	Графический интерфейс программы	18
3.5.	Консольный интерфейс программы	21
4.	Тестирование алгоритма	22
4.1.	План тестирования алгоритма	22
4.2.	Тестовые случаи	23
4.3.	Результаты тестирования алгоритма	28
5.	Тестирование графического интерфейса и взаимодействия с пользователем	29
5.1.	План тестирования графического интерфейса и взаимодействия с пользователем	29
5.2.	Тестовые случаи	29
5.3.	Результаты тестирования графического интерфейса и взаимодействия с пользователем	37
	Заключение	38
	Список использованных источников	39
	Приложение А. Исходный код программы	40

## **ВВЕДЕНИЕ**

Целью данной учебной практики является итеративная разработка GUI приложения для визуализации работы алгоритма Гольдберга, предназначенного для поиска максимального потока в сети.

Программа должна предоставить пользователю удобный интерфейс, дающий возможность изучить работу алгоритма на каждом шагу: визуализация сети и высотной функции на данном шаге, отображение параметров вершин и ребер на данном шаге, управление работой алгоритма (а именно переход к следующему или предыдущему шагу, или прогон алгоритма до завершения). Пользователю должна быть предоставлена возможность задать сеть через GUI или загрузить из файла.

Разработка ведется командой из трех человек, за каждым из которых закреплены определенные роли: разработка GUI приложения, визуализация алгоритма, реализация алгоритма Гольдберга, сборка и тестирование.

# 1. ТРЕБОВАНИЯ К ПРОГРАММЕ

## 1.1. Исходные требования к программе

Программа должна предоставлять интерфейс для пошаговой визуализации алгоритма Гольдберга.

### *1.1.1 Требования к проекту*

- 1) Возможность запуска через GUI и по желанию CLI (в данном случае достаточно вывода промежуточных значений);
- 2) Загрузка данных из файла или ввод через интерфейс;
- 3) GUI должен содержать интерфейс управления работой алгоритма, визуализацию алгоритма, окно с логами работы;
- 4) Должна быть возможность запустить алгоритм заново на новых данных без перезапуска программы;
- 5) Должна быть возможность выполнить один шаг алгоритма, либо завершить его до конца. В данном случае должны быть автоматически продемонстрированы все шаги;
- 6) Должна быть возможность вернуться на один шаг назад;
- 7) Должна быть возможность сбросить алгоритма в исходное состояние.

### *1.1.2 Требования реализации алгоритма*

- 1) Алгоритм должен быть реализован так, чтобы можно было использовать любой тип данных (Например через generic классы);
- 2) Алгоритм должен поддерживать возможность включения промежуточных выводов и пошагового выполнения.

## **2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ**

### **2.1. План разработки**

- 1.06: Распределение ролей в бригаде;
- 2.06: Создание репозитория для проекта и настройка системы автоматической сборки;
- 4.06: Написание плана разработки и создание UML-диаграмм классов, состояний и последовательностей;
- 5.06: Разработка прототипа: создание GUI без выполняемого функционала;
- 6.06: Отчёт по результатам первой итерации;
- 8.06: Реализация алгоритма Гольдберга, реализация частичного функционала GUI и тестирование;
- 9.06: Отчёт по результатам второй итерации;
- 10.06: Реализация взаимодействия с алгоритмом через GUI;
- 11.06: Реализация визуализации сети и тестирование GUI;
- 12.06: Отчёт по результатам третьей итерации;
- 14.06: Реализация дополнительного функционала.

### **2.2. Распределение ролей в бригаде**

Колованов Р.А. – разработка GUI программы и визуализация алгоритма;

Шахин Н.С. – реализация алгоритма Гольдберга;

Андрух И.А. – сборка и тестирование алгоритма и графического интерфейса.

### 3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

#### 3.1. Архитектура программы

*ResidualNetwork*<*T extends Number*> - класс, предназначенный для хранения остаточной сети. Вспомогательные классы *Node* и *EdgeProperties*<*T extends Number*> представляют собой вершины и параметры ребра сети (пропускная способность и проходящий через ребро поток) соответственно. Параметр шаблона *T* определяет тип данных пропускной способности и потока ребра сети.

*AlgorithmExecutor*<*T extends Number*> - класс, предназначенный для выполнения алгоритма Гольдберга в пошаговом режиме для заданной сети.

*NetworkLoader* – класс для загрузки графа из файла;

*NetworkSaver* – класс для сохранения графа в файл;

*Controller* – класс, предоставляющий интерфейс для взаимодействия классов GUI с остальными классами. Взаимодействие классов GUI с классом контроллера происходит с использованием паттерна Команды (классы, наследуемые от интерфейса *Command*).

*ViewPainter* - абстрактный класс, который служит родителем для классов *NetworkViewPainter*, *OriginalNetworkViewPainter*, *ResidualNetworkViewPainter* и *HeightFunctionViewPainter*, предназначенных для визуализации сети различными способами: в виде сети без обратных ребер, в виде остаточной сети и в виде высотной функции вершин. Для изменения способа отрисовки предполагается использовать паттерн Стратегия.

*NetworkParametersFormatter* - класс для форматирования всех параметров сети в текст, который впоследствии будет отображаться в GUI программы.

Класс *MessageHandler* используется для обработки сообщений логов и вывода их в консоль GUI интерфейса.

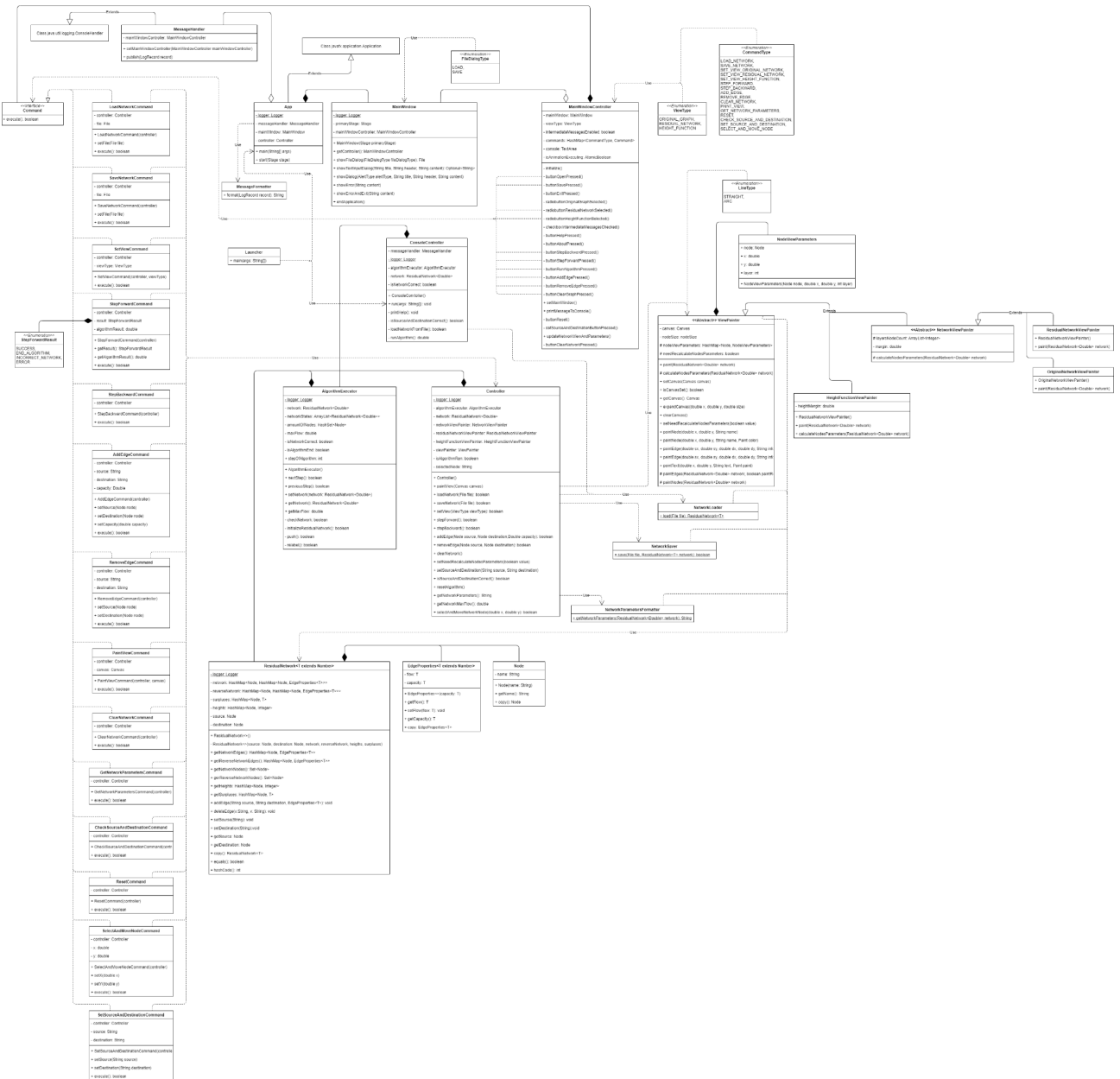
Классы *MainWindow* и *MainWindowController* представляют собой окно приложения и его контроллер соответственно. *MainWindowController*



обрабатывает взаимодействия пользователя с элементами GUI, а также изменяет состояние элементов GUI.

### 3.1.1 UML-диаграмма классов

На рисунке 1 представлена UML-диаграмма классов.



*Рисунок 1 - Диаграмма классов.*

### 3.1.2 UML-диаграмма состояний

На рисунке 2 представлена UML-диаграмма состояний.

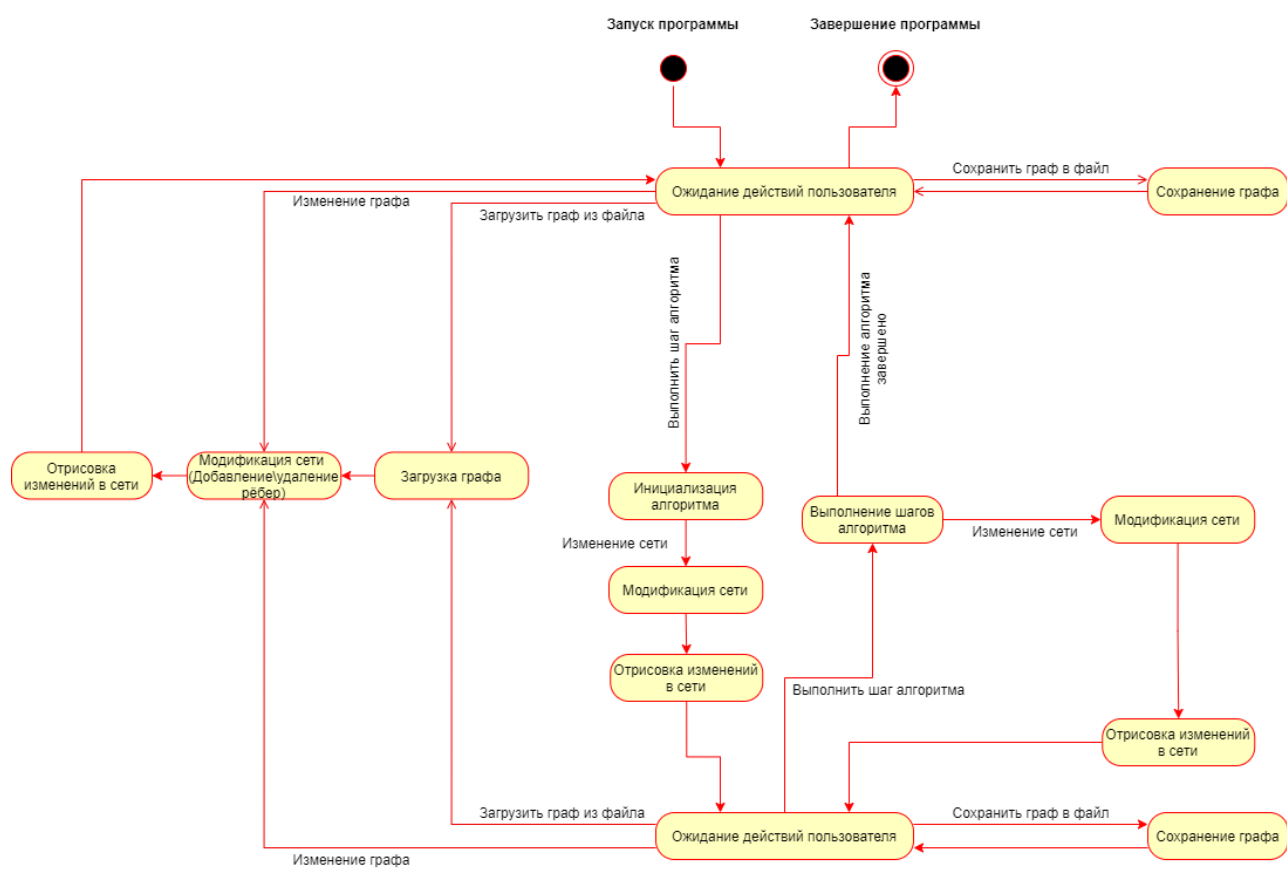


Рисунок 2 - Диаграмма состояний.

### 3.1.3 UML-диаграмма последовательностей

На рисунке 3 представлена UML-диаграмма последовательностей.

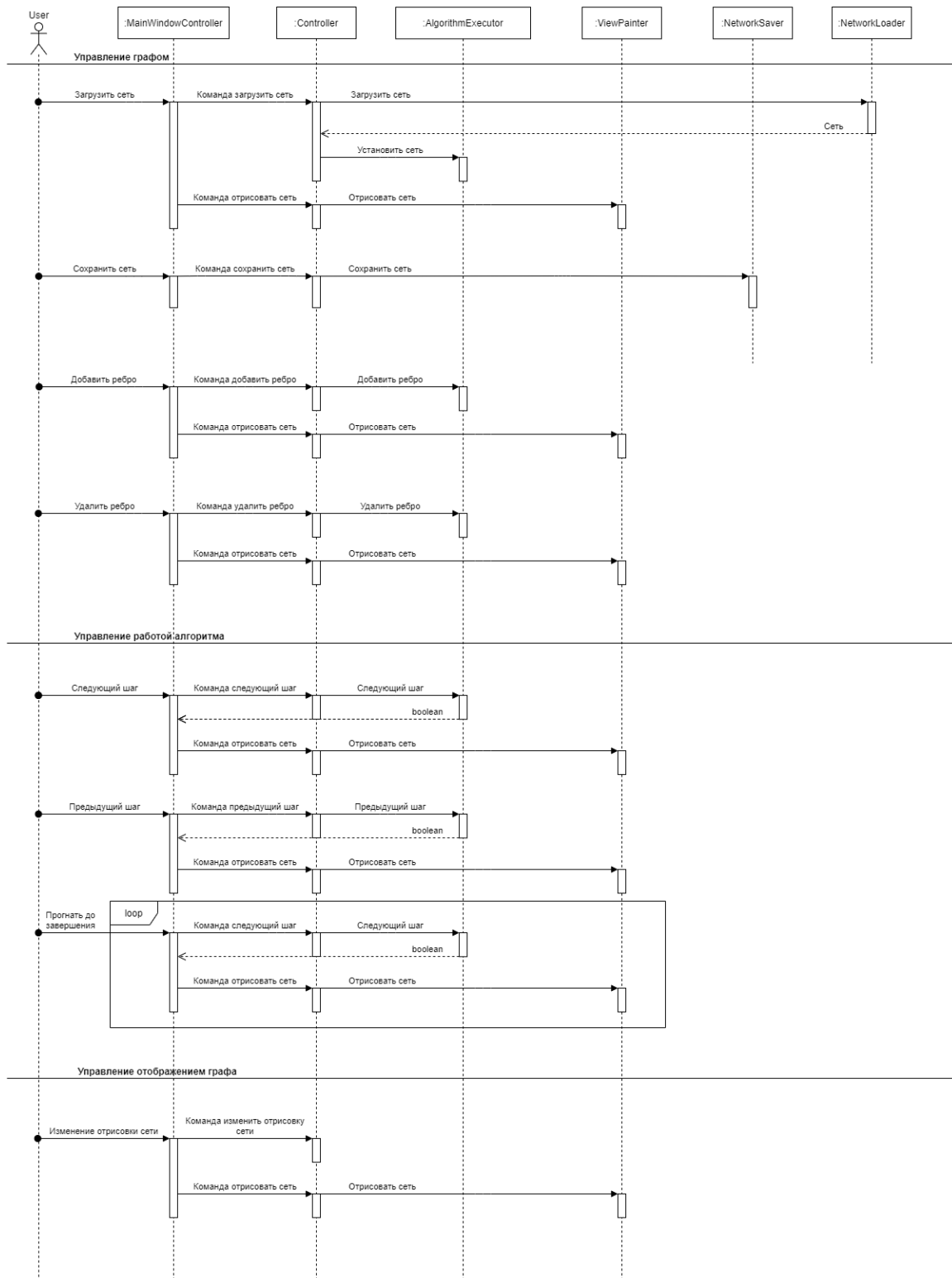


Рисунок 3 - Диаграмма последовательностей.

### 3.2. Описание алгоритма

Алгоритм Гольдберга решает задачу нахождения максимального потока в сети. Для описания алгоритма введем следующие определения:

#### 3.2.1 Предпоток.

Предпотоком будем называть функцию  $f: V \times V \rightarrow \mathbb{R}$ , удовлетворяющую следующим свойствам:

- Антисимметричность:  $f(u, v) = -f(v, u)$
- Ограничение пропускной способностью:  $f(u, v) \leq c(u, v)$
- Предпоток может проливаться (в вершину может втекать больше потока, чем вытекать):  $\forall u \in (V \setminus \{s, t\}) \sum_{v \in V} f(v, u) \geq 0$

*Максимальный предпоток* - предпоток становится максимальным предпотоком, когда в остаточной сети нет пути из истока в сток, а также нет пути из вершин с переполнением в сток.

#### 3.2.2 Высотная функция.

Введем высотную функцию, которую будем называть высотой вершины. Высотная функция удовлетворяет следующим свойствам:

- $h(\text{source}) = |V|$
- $h(\text{destination}) = 0$
- Для любого ребра  $(u, v)$ ,  $h(u) \leq h(v) + 1$

Эти три правила, приводят к тому, что в остаточной сети не будет пути от истока к стоку.

#### 3.2.2 Операции.

Определим две операции:

##### 1) Проталкивание (Push):

Операция push из вершины  $u$  в вершину  $v$  может применяться, когда:

- $e(u) > 0$ , то есть вершина  $u$  является переполненной
- остаточная пропускная способность ребра  $(u,v) > 0$
- $h(u) = h(v) + 1$

Операция выполняется следующим образом: по ребру  $(u,v)$  пропускается максимально возможный поток, то есть минимум из избытка вершины  $u$  и остаточной пропускной способности ребра  $(u,v)$ , по обратному ребру  $(v,u)$  пускается поток с отрицательной величиной, избыток вершины  $u$  уменьшается на данную величину, а избыток вершины  $v$  увеличивается.

## 2) Подъем (relabel):

Операция relabel применима для вершины  $u$ , если:

- $e(u) > 0$ , то есть вершина  $u$  является переполненной
- $\forall (u,v) \in E \ h(u) \leq h(v)$  если все вершины, для которых в остаточной сети есть рёбра из  $u$ , расположены не ниже  $u$

Операция выполняется следующим образом: в результате подъема высота текущей вершины становится на единицу больше высоты самой низкой смежной вершины в остаточной сети, вследствие чего появляется как минимум одно ребро, по которому можно протолкнуть поток.

*Выполнение алгоритма:*

### 1) Шаг 1: Инициализация алгоритма

Пропустим максимально возможный поток по рёбрам, инцидентным истоку, увеличив избыточный поток для каждой смежной с истоком вершиной на соответствующую величину. Все остальные потоки не несут, следовательно, для вершин не смежных с истоком избыточный поток изначально будет нулевым. Также для всех вершин, кроме, естественно, истока, установим высоту, равную нулю.

2) Шаг 2: Пока можем выполнить push или relabel, выполняем эти операции в произвольном порядке.

*Корректность алгоритма:*

После того, как мы не можем выполнять операции push и relabel достигается состояние, когда в остаточной сети нет вершин с избытками, значит нет пути в остаточной сети от истока в сток, а так же нет дополняющего пути, следовательно поток превращается в максимальный предпоток, который становится максимальным потоком.

*Сложность алгоритма:*  $O(V^2E)$

### 3.3. Структуры данных

#### 3.3.1 Класс *Node*

Класс *Node* используется для хранения вершины.

Поля класса:

- *String name* - хранит имя вершины.

Методы класса:

- *Node(String name)* - конструктор класса принимает имя вершины и присваивает полю *name* значение.
- *String getName()* возвращает значение поля *name*.
- Методы *equals()* и *hashCode()* - переопределены для сравнения объектов класса и хранения его в коллекции *HashMap*.
- *copy()* - возвращает копию экземпляра класса.

#### 3.3.2 Класс *EdgeProperties*

Класс *EdgeProperties* используется для хранения информации о ребре.

Поля класса:

- *T flow* - хранит величину текущего потока.
- *T capacity* - хранит величину максимального потока.

Методы класса:

- *EdgeProperties(T capacity, T flow)* - конструктор класса. Принимает значение *capacity* и *flow*, и присваивает их соответствующим полям.
- У класса есть “геттеры” которые возвращают значения полей *flow* и *capacity*, а также “сеттер” поля *flow*.
- *equals()* и *hashCode()* - переопределены для сравнения объектов класса и хранения его в коллекции *HashMap*.
- *copy()* - возвращает копию экземпляра класса.

### 3.3.3 Класс *ResidualNetwork*.

Класс *ResidualNetwork* используется для хранения остаточной сети.

Поля класса:

- *HashMap<Node, HashMap<Node, EdgeProperties<T>>> network* - ребра графа
- *HashMap<Node, HashMap<Node, EdgeProperties<T>>> reverseNetwork* - обратные ребра
- *HashMap<Node, T> surpluses* - переполнения вершин.
- *HashMap<Node, Integer> heights* - высоты вершин.
- *Node source* - источник.
- *Node destination* - сток.

Методы класса:

- “Геттеры” и “Сеттеры” для всех полей класса.
- *addEdge(Node from, Node to, EdgeProperties<T>edgeProperties)* - добавление ребра в остаточную сеть. Метод принимает концы ребра, и его параметры.
- *deleteEdge(Node from, Node to)* - удаление ребра из сети. Метод принимает концы ребра.
- *equals()* и *hashCode()* - переопределены для сравнения объектов класса.
- *copy()* - возвращает копию экземпляра класса.

### 3.3.4 Класс *AlgorithmExecutor*

Класс *AlgorithmExecutor* реализован для выполнения алгоритма Гольдберга.

Поля класса:

- *ResidualNetwork<Double> network* - остаточная сеть.
- *ArrayList<ResidualNetwork<Double>> networkStates* - список состояний остаточной сети.
- *HashSet<Node> amountOfNodes* - множество вершин графа.
- *double maxFlow* - величина максимального потока.



- *boolean isNetworkCorrect* - результат проверки сети на корректность.
- *boolean isAlgorithmEnd* - флаг отвечающий за завершения алгоритма.

Методы класса:

- *boolean setNetwork(ResidualNetwork<Double> network)* - принимает остаточную сеть, проверяет её на корректность и присваивает её полю *network*.
- *double getMaxFlow()* - возвращает максимальный поток в сети.
- *boolean checkNetwork()* - проверяет сеть на корректность.
- *boolean initializeNetwork()* - выполняет инициализацию алгоритма.
- *boolean push()* - выполняет операцию проталкивания.
- *boolean relabel()* - выполняет операцию поднятия.
- *boolean nextStep()* - следующий шаг алгоритма.
- *boolean previousStep()* - предыдущий шаг алгоритма.

### **3.4. Графический интерфейс программы**

#### *3.4.1 Описание графического интерфейса программы*

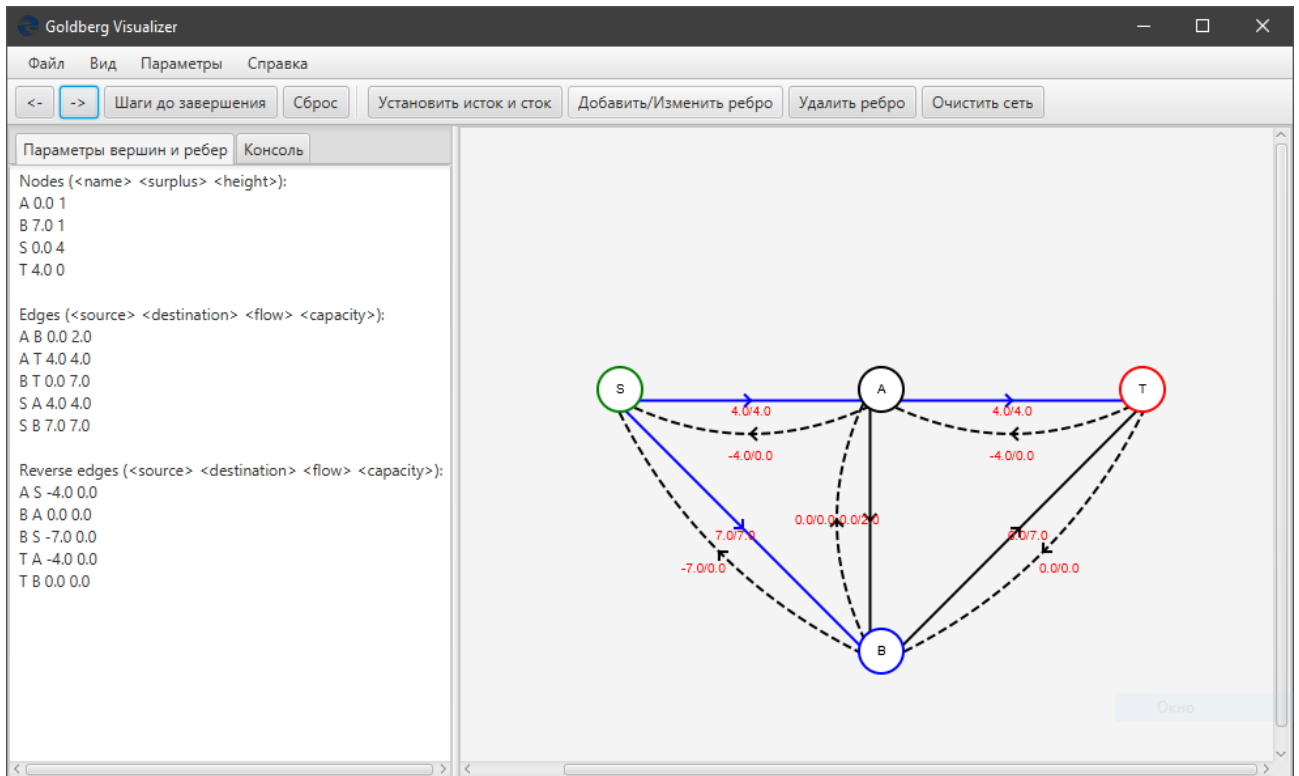
Для программы был разработан графический интерфейс, который позволяет пользователю выполнять следующие действия:

- Загрузка сети в файл (Вкладка Файл - Загрузить сеть...);
- Сохранение сети в файл (Вкладка Файл - Сохранить сеть...);
- Выход из программы (Вкладка Файл - Выход);
- Изменение вида визуализации сети: Исходная сеть, Остаточная сеть или Высотная функция (Вкладка Вид);
- Включить или выключить вывод промежуточных данных алгоритма в консоль (Вкладка Параметры - Промежуточные сообщения);
- Открыть справку приложения (Справка - Справка);
- Открыть информацию о программе (Справка - О программе);
- Совершить шаг алгоритма назад (Кнопка '<-' в меню);
- Совершить шаг алгоритма вперед (Кнопка '->' в меню);
- Прогнать шаги до завершения алгоритма (Кнопка 'Шаги до завершения' в меню);
- Сбросить сеть в исходное состояние до запуска алгоритма (Кнопка 'Сброс' в меню);
- Установить исток и сток сети (Кнопка 'Установить исток и сток' в меню);
- Добавить или изменить ребро сети (Кнопка 'Добавить/Изменить ребро' в меню);
- Удалить ребро сети (Кнопка 'Удалить ребро' в меню);
- Очистить текущую сеть (Кнопка 'Очистить сеть' в меню);
- Перемещение вершин (Перетаскивание вершин указателем мыши на сцене).

Во вкладке 'Консоль' отображаются промежуточные данные алгоритма и другая информация о действиях пользователя, а во вкладке 'Параметры вершин и ребер' отображаются все параметры вершин и ребер сети: для вершин - имя,

избыток и высота соответственно, а для ребер - величина потока и пропускная способность соответственно.

Графический интерфейс программы выглядит следующим образом:



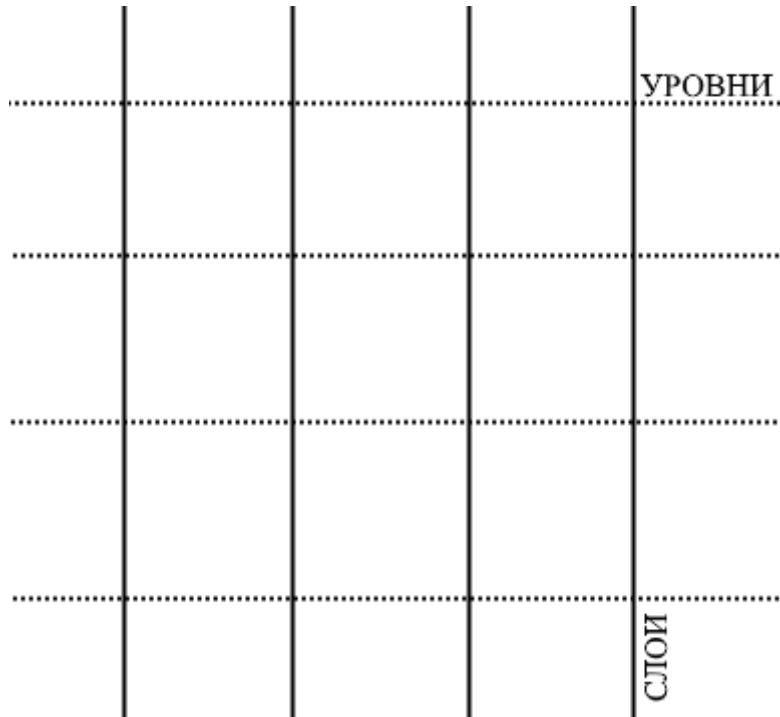
Вершины отображаются в виде окружностей, в центре которых отображаются имена вершин. Ребра исходной сети отображаются сплошной линией, а ребра обратной сети - пунктирной линией. Исток и сток окрашиваются в темно-синий и светло-синий соответственно. Ребра, через которые на данный момент проходит какой-либо поток, окрашиваются в синий цвет.

Обращение классов GUI к остальным классам программы происходит через класс *Controller* с использованием паттерна Команды.

### 3.4.2 Реализация визуализации сети

Для отрисовки различных примитивов (фигуры, линии) используются классы *Canvas* и *GraphicsContext* библиотеки JavaFX. Для начала при помощи

обхода графа в ширину вычисляются координаты вершин в окне отрисовки. Вершины располагаются по слоям, которые расположены друг за другом слева направо. В каждом слое вершины располагаются на уровнях друг за другом сверху вниз:



Исток размещается на самом левом слое. Вершины, в которые ведут исходящие из истока ребра, размещаются уже на следующем слое. В свою очередь вершины, в которые ведут исходящие из вершин некоторого слоя ребра, размещаются на следующем уровне.

После того, как координаты вершин были вычислены, производится отрисовка ребер, а после - вершин. Для реализации отрисовки были созданы классы *ViewPainter*, *NetworkViewPainter*, *OriginalNetworkViewPainter* и *ResidualNetworkViewPainter*, которые предоставляют методы для отрисовки примитивов (линий и фигур) и вычисления координат вершин в окне отрисовки.

Для визуализации высотной функции используется класс *HeightFunctionViewPainter*.

### 3.5. Консольный интерфейс программы

Для программы был разработан консольный интерфейс. Для запуска интерфейса командной строки пользователю необходимо передать программе флаг `-cli`. CLI позволяет пользователю выполнять следующие действия:

- Распечатать справку (Флаг `-h`);
- Включить промежуточные выводы (Флаг `-o`);
- Выбрать файл, из которого программа считывает сеть (Флаг `-f <путь к файлу>`).

Консольный интерфейс выглядит следующим образом:

При запуске программы с ключом `-cli` выводится справка.

```
C:\Users\ASUS\IdeaProjects\goldberg-visualizer\build\libs>java -jar goldberg-visualizer-0.1.jar -cli
Available flags:
-h          Print help
-f <file>   Choose file with network
-o          to enable intermediate outputs
```

Запуск работы алгоритма и передача сети из файла.

```
C:\Users\ASUS\IdeaProjects\goldberg-visualizer\build\libs>java -jar goldberg-visualizer-0.1.jar -cli -f C:\Users\ASUS\Documents\graph.txt
The maximum flow in the network is equal to 4.0
```

Включение промежуточных выводов флагом `-o`.

```
C:\Users\ASUS\IdeaProjects\goldberg-visualizer\build\libs>java -jar goldberg-visualizer-0.1.jar -cli -f C:\Users\ASUS\Documents\graph.txt -o
The output of intermediate messages is activated.
Checking correction of the network

|-----|
Start initialization of network
Push flow through edge 'S' 'A' flow is 10.0
Surplus of node 'A' is 10.0
Push flow through edge 'S' 'B' flow is 6.0
Surplus of node 'B' is 6.0
End initialization of network
|-----|
|-----|
Make relabel with node 'A'
Height of node 'A' was '0'
Height of node 'A' is 1
|-----|
```

Передача несуществующего файла.

```
C:\Users\ASUS\IdeaProjects\goldberg-visualizer\build\libs>java -jar goldberg-visualizer-0.1.jar -cli -f
The path to the file is not specified.
```

## 4. ТЕСТИРОВАНИЕ АЛГОРИТМА

### 4.1. План тестирования программы.

#### 4.1.1. Объект тестирования

Объектом тестирования является программа для визуализации работы алгоритма Гольдберга.

#### 4.1.2. Тестируемый функционал.

Необходимо протестировать метод *runAlgorithm()* класса *AlgorithmExecutor*, так как в этом методе происходит полное выполнение алгоритма.

#### 4.1.3. Подход к тестированию.

Тестирование будет проводиться на модульном уровне при помощи фреймворка автоматического тестирования JUnit.

#### 4.1.4. Критерии прекращения тестирования.

Тестирование считается успешно завершенным, если все тесты выполнены без ошибок. В противном случае программа возвращается на доработку.

#### 4.2. Тестовые случаи.

В таблице представлены все тест-кейсы, необходимые для полной проверки алгоритма. В том числе негативные проверки: для пустого графа, графа с отрицательными весами, графа из двух вершин без ребра.

Тест-кейсы для проверки алгоритма:

№	Суть теста	Тестовые данные	Результат работы алгоритма Гольдберга
1	Граф из двух вершин	a b a b 1	1
2 3 4	Произвольный граф	a d a b 2 b c 13 c d 9 h c 4 e f 7 g h 7 b e 7 a g 8 f d 10	6
		a d a b 1 b c 1 c d 1 a c 100 b d 100	2
		a h a c 12 a d 22 c d 14 c f 21 d g 10	14

		g f 10 g h 7 f h 7	
5	В графе есть ребро с нулевым весом	a f a b 5 a c 7 b d 0 c f 12 d e 9 d f 6 e c 6	7
6	Все ребра графа с нулевым весом	a f a b 0 a c 0 b d 0 c f 0 d e 0 d f 0 e c 0	0
7	В исток входят ребра	a f a b 5 a c 7 b a 5 b d 0 c a 7 c f 12 d e 9 d f 6 e c 6	7
8	Из стока выходят ребра	a d a b 4 a c 2 b c 3	5



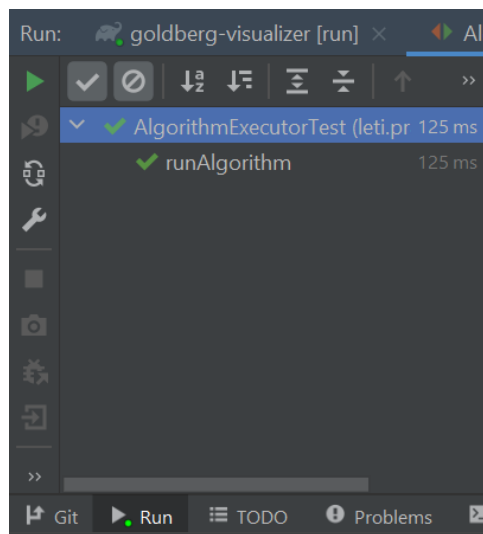
		b d 3 c d 2 d c 2	
9	Симметричный граф	a g a b 4 a c 4 b d 3 b e 6 c d 3 c f 6 d e 3 d f 3 e g 5 f g 5	8
10	Все ребра имеют одинаковый вес	a f a b 5 a c 5 b d 5 c f 5 d e 5 d f 5 e c 5	10
11	Двусторонний граф	16 a e a b 20 b a 20 a d 10 d a 10 a c 30 c a 30 b c 40 c b 40 c d 10 d c 10 c e 20 e c 20 b e 30 e b 30	60

		d e 10 e d 10	
12	Простой несвязный граф из двух частей	a d a b 2 c d 2	0
13	Большой несвязный граф из двух частей	a h a b 5 a c 6 a d 7 b c 3 b e 7 c d 4 d e 10 f g 4 f h 4	0
14	Несвязный граф из трех частей	a h a c 6 b c 7 d e 10 f g 4 f h 4	0
15	Граф с дробными весами ребер	a d a b 1,5 a c 0,1 b c 0,2 b d 0,1 c d 0,2	0,3
16	Граф с отрицательными весами ребер	a f a b -5 a c -7 b d 0 c f -12 d e -9 d f -6	-1

		е с -6	
17	Пустой граф		предупреждение “nullPointerException”
18	Две вершины без ребра	a b	-1

### 4.3. Результаты тестирования.

#### 4.3.1. Скриншот запуска unit-теста:



Таким образом, тестирование можно считать пройденным, так как фактические и ожидаемые результаты всех запланированных тестов совпали. Покрытие кода тестами было оценено в 97%.

## **5. ТЕСТИРОВАНИЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА И ВЗАИМОДЕЙСТВИЯ С ПОЛЬЗОВАТЕЛЕМ.**

### **5.1. План тестирования графического интерфейса и взаимодействия с пользователем.**

#### *5.1.1. Объект тестирования.*

Объектом тестирования также является приложение для визуализации работы алгоритма Гольдберга.

#### *5.1.2. Тестируемый функционал.*

В отличие от пункта 4, в этом пункте предполагается протестировать графический интерфейс приложения. Также будут проверены методы *nextStep()* и *previousStep()* для перехода по алгоритму на шаг вперед и назад.

#### *5.1.3. Подход к тестированию.*

Для GUI будет производиться ручное тестирование согласно написанному чек-листу. Для методов взаимодействия пользователя и алгоритма планируется написать unit-тесты.

#### *5.1.4. Критерии прекращения тестирования.*

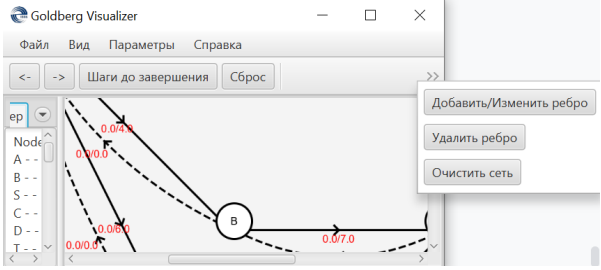
Тестирование можно считать завершенным, когда ожидаемый и фактический результаты для каждого тест-кейса совпали.

### **5.2. Тестовые случаи.**

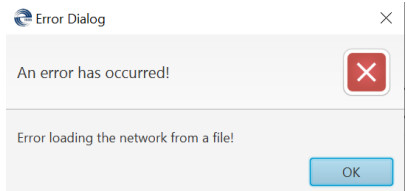
#### *5.2.1. Тестовые случаи для проверки GUI*

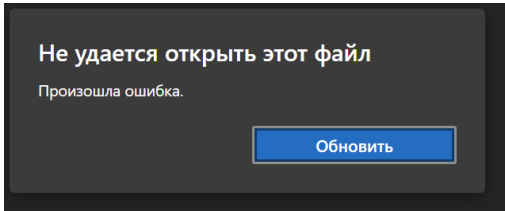
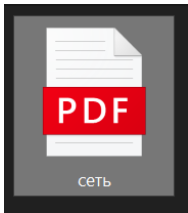
Ниже приведен чек-лист проверок для всех элементов графического интерфейса.

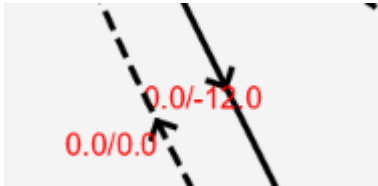
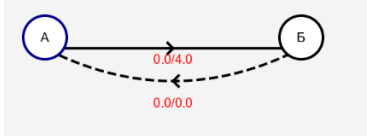
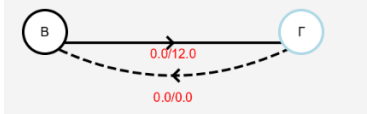
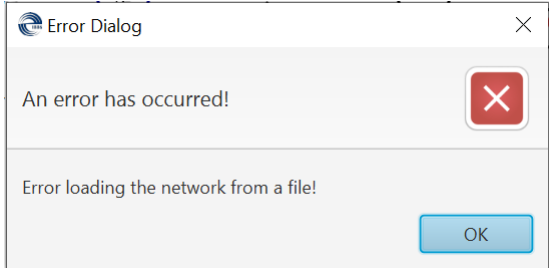
- окно приложения

суть теста	ОР	ФР
увеличить/уменьшить размер окна	размер окна изменился, элементы окна расположены правильно, изображение в центре	размер окна изменился, элементы окна расположены правильно, изображение в центре. Для не влезающих кнопок появляется открывающееся меню. 
перенести окно	окно переносится в любую часть экрана, но не убирается совсем	окно переносится в любую часть экрана, но не убирается совсем; также при попытке убрать окно совсем оно было развернуто на пол экрана.
свернуть	окно свернуто, приложение доступно по иконке приложения	окно свернуто, приложение доступно по иконке приложения
свернуть в окно	окно свернулось и развернулось	окно свернулось и развернулось
закрыть	окно закрылось, работа алгоритма остановлена	окно закрылось, работа алгоритма остановлена

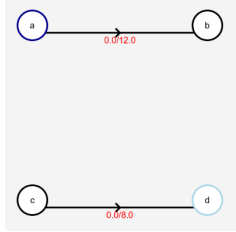
- файл

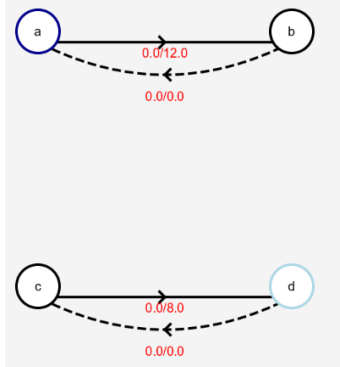
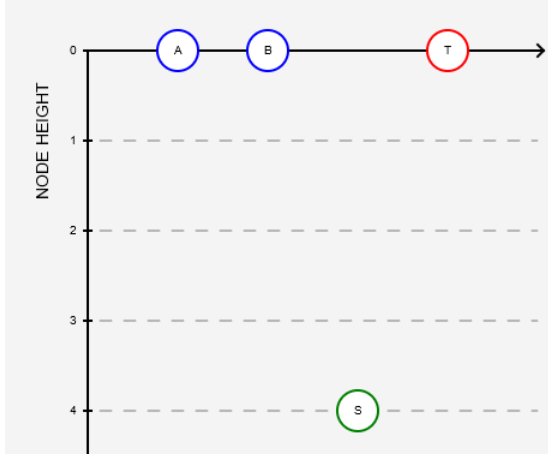
загрузить сеть с корректными данными	сеть загрузилась и отобразилась	сеть загрузилась и отобразилась
загрузить пустой файл	предупреждение	
загрузить файл с	при запуске	при запуске алгоритма будет

указанием только истока и стока	алгоритма будет предупреждение	предупреждение
загрузить файл без указания истока и стока	при запуске алгоритма будет предупреждение	при запуске алгоритма будет предупреждение
загрузить файл .png	нет возможности загрузить не .txt файл	нет возможности загрузить не .txt файл
загрузить файл .doc	нет возможности загрузить не .txt файл	нет возможности загрузить не .txt файл
сохранить в файл	сеть корректно сохранилась	сеть корректно сохранилась
сохранить в файл с неправильным расширением	файл сохраняется в заданном формате, но открывается через блокнот	<p>файл сохраняется в заданном формате, но открывается через блокнот. Попытка открыть pdf:</p>  

<p>сохранить файл, изменить содержимое на некорректное, загрузить сеть</p> <p>1)создать отрицательное ребро</p> <p>2) использовать кириллицу</p> <p>3) использовать специальные символы</p> <p>4)изменить форматирование текста</p>	<p>1)создалось ребро с отриц. весом, но при запуске алгоритма будет предупреждение</p> <p>2)граф отрисован, алгоритм работает</p> <p>3)граф отрисован, алгоритм работает</p> <p>4)предупреждение</p>	 <p>1)создалось ребро с отриц. весом, но при запуске алгоритма предупреждение</p> <p>2)граф отрисован, алгоритм работает</p>  <p>3)граф отрисован, алгоритм работает</p>  <p>4)предупреждение</p> 
---	--	---

● ВИД

<p>исходная сеть</p>	<p>отображается исходная сеть на данном шаге алгоритма</p>	<p>отображается исходная сеть на данном шаге алгоритма</p> 
----------------------	--	---

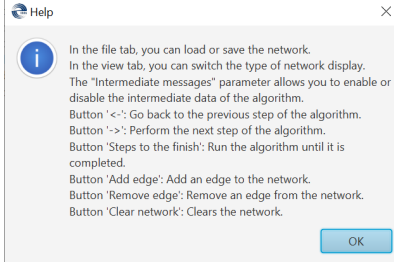
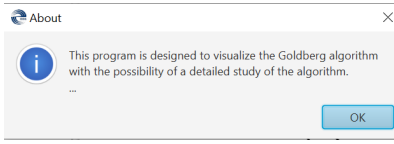
остаточная сеть	отображается остаточная сеть на данном шаге алгоритма	отображается остаточная сеть на данном шаге алгоритма 
высотная функция	отображается высотная функция на данном шаге алгоритма	отображается высотная функция на данном шаге алгоритма 

- параметры

отключить промежуточные сообщения	промежуточные сообщения не выводятся в консоль	промежуточные сообщения не выводятся в консоль
включить промежуточные сообщения	промежуточные сообщения выводятся в консоль	промежуточные сообщения выводятся в консоль

- справка



справка	выводится справка о функционале	
о программе	выводится информация о программе	

- шаг назад

сделать шаг назад	отображение предыдущего шага алгоритма	отображение предыдущего шага алгоритма
сделать несколько шагов назад	отображение предыдущих шагов алгоритма	отображение предыдущих шагов алгоритма
сделать шаг назад при условии что предыдущих шагов нет	ничего	ничего

- шаг вперед

сделать шаг вперед	отображение следующего шага алгоритма	отображение следующего шага алгоритма
сделать несколько шагов вперед	отображение последовательно нескольких следующих шагов алгоритма	отображение последовательно нескольких следующих шагов алгоритма
сделать шаг вперед на последнем шаге	ничего	ничего

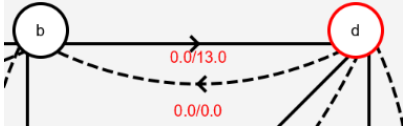
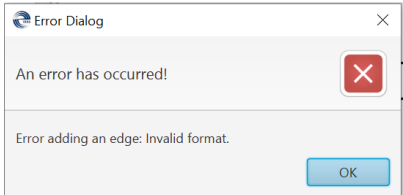
- шаги до завершения

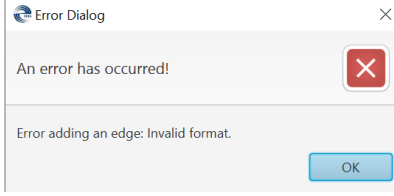
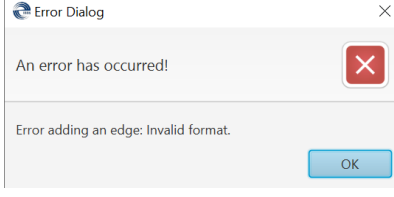
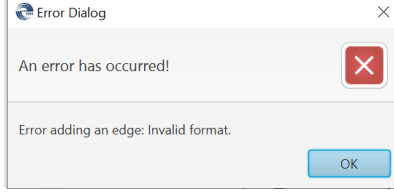
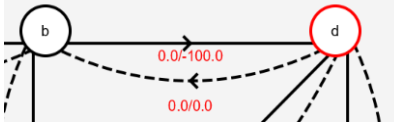
вызвать шаги до завершения в начале программы	Алгоритм выполнен до конца	Алгоритм выполнен до конца
вызвать шаги до завершения в середине программы	Алгоритм выполнен до конца	Алгоритм выполнен до конца
вызвать шаги до завершения когда алгоритм завершился	ничего не происходит	ничего не происходит

- сброс

сброс в середине выполнения алгоритма	возврат к началу алгоритма	возврат к началу алгоритма
сброс в начале выполнения алгоритма	возврат к началу алгоритма	возврат к началу алгоритма
сброс в конце выполнения алгоритма	возврат к началу алгоритма	возврат к началу алгоритма

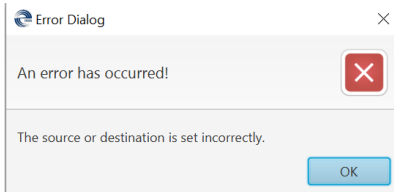
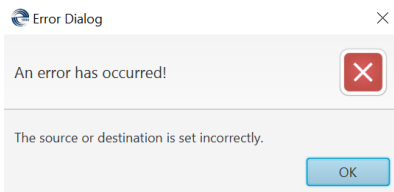
- добавить/изменить ребро

добавить корректное ребро	отобразилось дополнительное ребро	
добавить ребро без веса	предупреждение	

добавить ребро без одной из вершин	предупреждение	
добавить ребро без вершин	предупреждение	
добавить уже существующее ребро	граф не изменился	граф не изменился
изменить вес ребра	изменился вес ребра	изменился вес ребра
изменить вес ребра на нулевой	вес ребра равен нулю	вес ребра равен нулю
изменить вес ребра на некорректный символ	предупреждение	
изменить вес ребра на отрицательный	при запуске алгоритма будет предупреждение	

- удалить ребро

удалить существующее ребро	ребро удалилось	ребро удалилось
удалить несуществующее ребро	граф не изменился	граф не изменился
удалить единственное ребро из истока	предупреждение при запуске алгоритма	ребро удалилось вместе с вершиной, при запуске алгоритма предупреждение

		
удалить единственное ребро из истока и сделать шаг назад	ничего не происходит	ничего не происходит
удалить единственное ребро в сток	ребро и вершина удалились, но при выполнении алгоритма выводится предупреждение	ребро и вершина удалились, но при выполнении алгоритма выводится предупреждение 
удалить единственное ребро в графе	граф очистился, при выполнении алгоритма выводится предупреждение	граф очистился, при выполнении алгоритма выводится предупреждение

- ОЧИСТИТЬ СЕТЬ

очистить непустую сеть	сеть очищена	сеть очищена
очистить пустую сеть	выведен лог	выведен лог

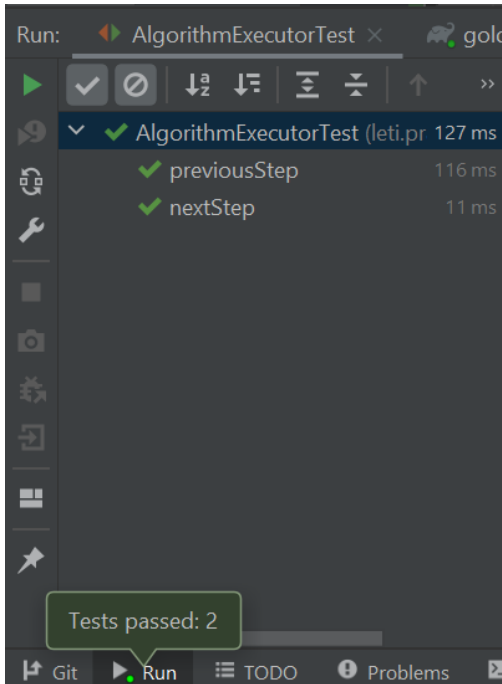
### 5.2.2. Тестовые случаи для *nextStep* и *previousStep()*

Тестовыми данными для проверки шагов вперед и назад являются некоторые корректные графы из пункта 4.2, так как на некорректных графах алгоритм не запускается (это проверено тестированием в пункте 4), соответственно и шаги выполняться не будут.

### 5.3. Результаты тестирования

В результате тестирования GUI для всех тест-кейсов совпал ожидаемый и фактический результат.

При тестировании взаимодействия пользователя и алгоритма были написаны unit-тесты, результат работы которых представлен ниже.



Таким образом, тестирование GUI и взаимодействия пользователя с алгоритмом можно считать завершенными. Покрытие требований можно считать равным 100%, так как чек-лист содержит проверки на весь функционал графического интерфейса, описанный в требованиях. Покрытие кода после этого тестирования также равно 100%.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения практической работы было создано и протестировано приложение для визуализации работы алгоритма Гольдберга. Программа соответствует требованиям, сформированным на момент начала разработки. В создании приложения принимали активное участие все члены команды.

Во время разработки алгоритма были получены практические навыки использования языка программирования Java, получен опыт работы с фреймворком для разработки пользовательских интерфейсов JavaFX и фреймворком для юнит-тестирования JUnit. Также были приобретены навыки составления диаграмм для отображения внутренних процессов программы и навыки командной работы.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java Platform, Standard Edition 8 API Specification // Oracle Help Center.  
URL: <https://docs.oracle.com/javase/8/docs/api/overview-summary.html> (дата обращения: 11.07.2021).
2. Java. Базовый курс // Stepik. URL: <https://stepik.org/course/187/info> (дата обращения: 05.07.2021).
3. JavaFX Reference Documentation // JavaFX. URL: <https://openjfx.io/> (дата обращения: 11.07.2021).
4. Учебник по JavaFX (Русский) // code.makery. URL: <https://code.makery.ch/ru/library/javafx-tutorial/> (дата обращения: 07.07.2021).
5. Руководства JavaFX // betacode. URL: <https://betacode.net/11009/javafx> (дата обращения: 07.07.2021).
6. Алгоритм проталкивания предпотока // Википедия. URL: [https://ru.wikipedia.org/wiki/Алгоритм\\_проталкивания\\_предпотока](https://ru.wikipedia.org/wiki/Алгоритм_проталкивания_предпотока) (дата обращения: 08.07.2021)

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл AlgorithmExecutor.java:

```
package leti.practice.algorithm;

import leti.practice.structures.graph.*;

import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.ArrayList;

public class AlgorithmExecutor {
    private static final Logger logger =
Logger.getLogger(AlgorithmExecutor.class.getName());
    private ResidualNetwork<Double> network;
    private ArrayList<ResidualNetwork<Double>> networkStates;
    private HashSet<Node> amountOfNodes;
    private double maxFlow;
    private boolean isNetworkCorrect;
    private boolean isAlgorithmEnd;
    private int stepOfAlgorithm;

    public AlgorithmExecutor(){
        network = null;
        networkStates = null;
        amountOfNodes = null;
        maxFlow = 0.0;
        isNetworkCorrect = false;
        isAlgorithmEnd = false;
        stepOfAlgorithm = 0;
    }

    public boolean setNetwork(ResidualNetwork<Double> network){
        this.network = null;
        networkStates = null;
        amountOfNodes = null;
        maxFlow = 0.0;
```



```

        isNetworkCorrect = false;
        isAlgorithmEnd = false;
        stepOfAlgorithm = 0;

        if (network == null) {
            logger.log(Level.FINEST, "Error network is empty\n");
            throw new NullPointerException();
        }

        if (network.getSource() == null || network.getDestination() == null) {
            throw new NullPointerException();
        }

        this.network = network;
        networkStates = new ArrayList<>();
        isNetworkCorrect = checkNetwork();
        return isNetworkCorrect;
    }

    public ResidualNetwork<Double> getNetwork() {
        return network;
    }

    public double getMaxFlow() {
        if (isAlgorithmEnd) {
            maxFlow = 0.0;

            for (Node dest :
network.getReverseNetworkEdges(network.getDestination()).keySet()) {
                maxFlow +=
network.getReverseNetworkEdges(network.getDestination()).get(dest).getFlow();
            }

            if (Double.compare(maxFlow, 0.0) != 0) {
                maxFlow *= -1;
            }
        }

        return maxFlow;
    }

```

```

private boolean checkNetwork(){
    logger.log(Level.FINEST, "Checking correction of the network\n");

    amountOfNodes = new HashSet<>(network.getNetworkNodes());
    amountOfNodes.addAll(network.getReverseNetworkNodes());

    if (!amountOfNodes.contains(network.getSource()) ||
!amountOfNodes.contains(network.getDestination())) {
        return false;
    }

    for(Node from : network.getNetworkNodes()){
        for (Node to : network.getNetworkEdges(from).keySet()){
            if(network.getNetworkEdges(from).get(to).getCapacity() < 0.0
|| network.getNetworkEdges(from).get(to).getFlow() <
0.0
|| network.getNetworkEdges(from).get(to).getFlow() >

network.getNetworkEdges(from).get(to).getCapacity()){
                logger.log(Level.FINEST, "Find error edge "+
from.getName()+" "+to.getName());
                return false;
            }
        }
    }

    return true;
}

private boolean initializeNetwork(){
    if(isNetworkCorrect){
        logger.log(Level.FINEST,
"|-----|\nStart initialization of network");

        amountOfNodes.add(network.getDestination());
        amountOfNodes.add(network.getSource());

        /*Initialization of heights and surplus function*/
        for (Node node:amountOfNodes){
            network.getHeights().put(node, 0);
            network.getSurpluses().put(node, 0.0);
        }
    }
}

```

```

        network.getHeights().put(network.getSource(),
amountOfNodes.size());
        //add to the states
        /*First algorithm step*/
        HashMap<Node, EdgeProperties<Double>> sourceEdges =
network.getNetworkEdges(network.getSource());

        if (sourceEdges != null) {
            for (Node to :
network.getNetworkEdges(network.getSource()).keySet()) {
                //add flow to edge

network.getNetworkEdges(network.getSource()).get(to).setFlow(network.getNetworkEdges
(network.getSource()).get(to).getCapacity());
                logger.log(Level.FINEST, "Push flow though edge '" +
network.getSource().getName() + "' '" + to.getName() + "' flow is " +
network.getNetworkEdges(network.getSource()).get(to).getCapacity());
                //add surplus to the node
                network.getSurpluses().put(to,
network.getNetworkEdges(network.getSource()).get(to).getCapacity());
                logger.log(Level.FINEST, "Surplus of node '" +
to.getName() + "' is " + network.getSurpluses().get(to));
                //add flow to the reverse edge

network.getReverseNetworkEdges(to).get(network.getSource()).setFlow(-1 *
network.getNetworkEdges(network.getSource()).get(to).getCapacity());
            }
        }

        amountOfNodes.remove(network.getDestination());
        amountOfNodes.remove(network.getSource());
        logger.log(Level.FINEST, "End initialization of
network\n|-----|");
        return true;
    }

    return false;
}

private boolean push(){
    if(isNetworkCorrect){

```

```

        for(Node node : amountOfNodes){
            //Если переполнение
            if(network.getSurpluses().get(node)>0){
                if(network.getReverseNetworkNodes().contains(node)){
                    for (Node to :
network.getReverseNetworkEdges(node).keySet()){
                        //если можно пропустить поток через обратное
ребро

                        if(network.getReverseNetworkEdges(node).get(to).getFlow()< 0.0){
                            if(network.getHeight().get(node) ==
network.getHeight().get(to) + 1){
                                logger.log(Level.FINEST,
"|-----|");
                                logger.log(Level.FINEST, "Make push with
node '" + node.getName() + "'");
                                double availableAmountOfFlow =
Math.min(network.getSurpluses().get(node),
network.getReverseNetworkEdges(node).get(to).getCapacity() -
network.getReverseNetworkEdges(node).get(to).getFlow());

                                network.getReverseNetworkEdges(node).get(to).setFlow(network.getReverseNetworkEdge
s(node).get(to).getFlow() + availableAmountOfFlow);
                                logger.log(Level.FINEST, "Add flow " +
availableAmountOfFlow + " to the Edge '"+node.getName()+"' '"+to.getName() + "'");

                                network.getNetworkEdges(to).get(node).setFlow(network.getNetworkEdges(to).get(node
).getFlow() - availableAmountOfFlow);
                                logger.log(Level.FINEST, "Flow of edge
 '"+to.getName()+"' '"+ node.getName()+"' is
 '"+network.getNetworkEdges(to).get(node).getFlow());
                                network.getSurpluses().put(node,
network.getSurpluses().get(node) - availableAmountOfFlow);
                                logger.log(Level.FINEST, "Surplus of
node '"+node.getName()+"' is "+network.getSurpluses().get(node));
                                network.getSurpluses().put(to,
network.getSurpluses().get(to) + availableAmountOfFlow);
                                logger.log(Level.FINEST, "Surplus of
node '"+to.getName()+"' is "+network.getSurpluses().get(to));
                                logger.log(Level.FINEST,
"|-----|");
                                return true;

```

```

    }
    }
}

if(network.getNetworkNodes().contains(node)){
    for(Node to
:network.getNetworkEdges(node).keySet()){
        //если можно пропустить поток через ребро

if(network.getNetworkEdges(node).get(to).getCapacity() -
network.getNetworkEdges(node).get(to).getFlow() > 0.0){
            if(network.getHeights().get(node) ==
network.getHeights().get(to) + 1){

                logger.log(Level.FINEST,
"|-----|");
                logger.log(Level.FINEST, "Make push with
node '" + node.getName() + "'");

                double availableAmountOfFlow =
Math.min(network.getSurpluses().get(node),
network.getNetworkEdges(node).get(to).getCapacity() -
network.getNetworkEdges(node).get(to).getFlow());

network.getNetworkEdges(node).get(to).setFlow(network.getNetworkEdges(node).get(to)
).getFlow() + availableAmountOfFlow);

                logger.log(Level.FINEST, "Add flow " +
availableAmountOfFlow + " to the Edge '"+node.getName()+"' '"+to.getName() + "'");

network.getReverseNetworkEdges(to).get(node).setFlow(network.getReverseNetworkEdge
s(to).get(node).getFlow() - availableAmountOfFlow);

                logger.log(Level.FINEST, "Flow of
reverse edge '"+to.getName()+"' '"+ node.getName()+"' is
"+network.getReverseNetworkEdges(to).get(node).getFlow());

                network.getSurpluses().put(node,
network.getSurpluses().get(node) - availableAmountOfFlow);

                logger.log(Level.FINEST, "Surplus of
node '"+node.getName()+"' is "+network.getSurpluses().get(node));

                network.getSurpluses().put(to,
network.getSurpluses().get(to) + availableAmountOfFlow);

                logger.log(Level.FINEST, "Surplus of
node '"+to.getName()+"' is "+network.getSurpluses().get(to));

```

```

        logger.log(Level.FINEST,
"|-----|");
        return true;
    }
}
}
}
}
}

return false;
}

private boolean relabel(){
    if(isNetworkCorrect){
        for(Node node : amountOfNodes) {
            //Если переполнение
            boolean flag = false;

            if (network.getSurpluses().get(node) > 0){
                ArrayList<Integer> heights = new ArrayList<>();
                if(network.getNetworkNodes().contains(node)) {
                    for (Node to :
network.getNetworkEdges(node).keySet()) {
                        if
(!network.getNetworkEdges(node).get(to).getCapacity().equals(network.getNetworkEdges(node).get(to).getFlow())) {
                            if (network.getHeights().get(node) >
network.getHeights().get(to)) {
                                flag = true;
                                break;
                            }

                            heights.add(network.getHeights().get(to));
                        }
                    }
                }
                if(network.getReverseNetworkNodes().contains(node)){
                    if(network.getReverseNetworkNodes().contains(node)){
                        for (Node to :
network.getReverseNetworkEdges(node).keySet()) {

```

```

if(!network.getReverseNetworkEdges(node).get(to).getFlow().equals(network.getReverseNetworkEdges(node).get(to).getCapacity())) {
    if (network.getHeights().get(node) >
network.getHeights().get(to)) {
        flag = true;
        break;
    }

    heights.add(network.getHeights().get(to));
    }
    }
    }
    if(!flag){
        logger.log(Level.FINEST,
"|-----|");
        logger.log(Level.FINEST, "Make relabel with node '"
+ node.getName() + "'");
        logger.log(Level.FINEST, "Height of node '" +
node.getName()+"' was '"+network.getHeights().get(node));
        network.getHeights().put(node,
1+Math.min(network.getHeights().get(node), Collections.min(heights)));
        logger.log(Level.FINEST, "Height of node '" +
node.getName()+"' is '"+network.getHeights().get(node));
        logger.log(Level.FINEST,
"|-----|");
        return true;
    }
    }
    }
    }

    return false;
}

public boolean nextStep(){
    if(isNetworkCorrect && !isAlgorithmEnd) {
        ResidualNetwork<Double> previousNetwork = network.copy();

```

```

        if (networkStates.size() == 0){
            if (initializeNetwork()) {
                networkStates.add(previousNetwork.copy());
                return true;
            }
            return false;
        }

        if (relabel()) {
            networkStates.add(previousNetwork.copy());
            return true;
        }

        if (push()) {
            networkStates.add(previousNetwork.copy());
            return true;
        }

        isAlgorithmEnd = true;
        logger.log(Level.FINEST, "The algorithm is completed");
    }

    if (!isNetworkCorrect) {
        logger.log(Level.FINEST, "The network is incorrect");
    }

    return false;
}

public boolean previousStep(){
    if(isNetworkCorrect) {
        if (networkStates.size() != 0) {
            if (isAlgorithmEnd) {
                isAlgorithmEnd = false;
            }
            network = networkStates.get(networkStates.size() - 1);
            networkStates.remove(networkStates.size() - 1);
            return true;
        }
    }
    return false;
}

```



```

    }

    public double runAlgorithm(ResidualNetwork<Double> network){
        if (setNetwork(network)){
            while (nextStep()){
                return getMaxFlow();
            }

            return -1.0;
        }
    }
}

```

### Файл AlgorithmExecutorTest.java:

```

package leti.practice.algorithm;

import leti.practice.structures.graph.EdgeProperties;
import leti.practice.structures.graph.Node;
import leti.practice.structures.graph.ResidualNetwork;
import org.junit.Test;
import org.junit.jupiter.api.Assertions;

public class AlgorithmExecutorTest {

    @Test
    public void nextStep() {
        AlgorithmExecutor test_alg = new AlgorithmExecutor();
        ResidualNetwork<Double> test_network = new ResidualNetwork<>();

        test_network.setSource(new Node("a"));
        test_network.setDestination(new Node("d"));
        test_network.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(2.0, 0.0));
        test_network.addEdge(new Node("b"), new Node("c"), new
EdgeProperties<>(13.0, 0.0));
        test_network.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(9.0, 0.0));
        test_network.addEdge(new Node("h"), new Node("c"), new
EdgeProperties<>(4.0, 0.0));
        test_network.addEdge(new Node("e"), new Node("f"), new
EdgeProperties<>(7.0, 0.0));
        test_network.addEdge(new Node("g"), new Node("h"), new
EdgeProperties<>(7.0, 0.0));
    }
}

```

```

        test_network.addEdge(new Node("b"), new Node("e"), new
EdgeProperties<>(7.0, 0.0));
        test_network.addEdge(new Node("a"), new Node("g"), new
EdgeProperties<>(8.0, 0.0));
        test_network.addEdge(new Node("f"), new Node("d"), new
EdgeProperties<>(10.0, 0.0));

        test_alg.setNetwork(test_network);
        test_alg.runAlgorithm(test_network);
        ResidualNetwork<Double> test_network_before = test_network.copy();
        test_alg.previousStep();
        ResidualNetwork<Double> test_network_between =
test_alg.getNetwork().copy();
        test_alg.nextStep();
        ResidualNetwork<Double> test_network_after = test_network.copy();

        Assertions.assertTrue(test_network_before.equals(test_network_after));

        Assertions.assertFalse(test_network_before.equals(test_network_between));
    }

    @Test
    public void previousStep() {
        AlgorithmExecutor test_alg = new AlgorithmExecutor();
        ResidualNetwork<Double> test_network = new ResidualNetwork<>();
        test_network.setSource(new Node("a"));
        test_network.setDestination(new Node("h"));
        test_network.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(12.0, 0.0));
        test_network.addEdge(new Node("a"), new Node("d"), new
EdgeProperties<>(22.0, 0.0));
        test_network.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(14.0, 0.0));
        test_network.addEdge(new Node("c"), new Node("f"), new
EdgeProperties<>(21.0, 0.0));
        test_network.addEdge(new Node("d"), new Node("g"), new
EdgeProperties<>(10.0, 0.0));
        test_network.addEdge(new Node("g"), new Node("f"), new
EdgeProperties<>(10.0, 0.0));
        test_network.addEdge(new Node("g"), new Node("h"), new
EdgeProperties<>(7.0, 0.0));

```

```

        test_network.addEdge(new Node("f"), new Node("h"), new
EdgeProperties<>(7.0, 0.0));

        test_alg.setNetwork(test_network);
        nextStep();
        nextStep();
        ResidualNetwork<Double> test_network_before = test_alg.getNetwork();
        test_alg.previousStep();
        ResidualNetwork<Double> test_network_between =
test_alg.getNetwork().copy();
        test_alg.nextStep();
        ResidualNetwork<Double> test_network_after = test_alg.getNetwork();

        Assertions.assertTrue(test_network_before.equals(test_network_after));

        Assertions.assertFalse(test_network_before.equals(test_network_between));
    }

    @Test
    public void runAlgorithm() {
        AlgorithmExecutor test_alg = new AlgorithmExecutor();

        // Тестовые случаи исходных сетей
        // 1
        ResidualNetwork<Double> test_network_1 = new ResidualNetwork<>();
        test_network_1.setSource(new Node("a"));
        test_network_1.setDestination(new Node("b"));
        test_network_1.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(1.0, 0.0));

        // 2
        ResidualNetwork<Double> test_network_2 = new ResidualNetwork<>();
        test_network_2.setSource(new Node("a"));
        test_network_2.setDestination(new Node("d"));
        test_network_2.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(2.0, 0.0));
        test_network_2.addEdge(new Node("b"), new Node("c"), new
EdgeProperties<>(13.0, 0.0));
        test_network_2.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(9.0, 0.0));
        test_network_2.addEdge(new Node("h"), new Node("c"), new
EdgeProperties<>(4.0, 0.0));

```

```

        test_network_2.addEdge(new Node("e"), new Node("f"), new
EdgeProperties<>(7.0,0.0));
        test_network_2.addEdge(new Node("g"), new Node("h"), new
EdgeProperties<>(7.0,0.0));
        test_network_2.addEdge(new Node("b"), new Node("e"), new
EdgeProperties<>(7.0,0.0));
        test_network_2.addEdge(new Node("a"), new Node("g"), new
EdgeProperties<>(8.0,0.0));
        test_network_2.addEdge(new Node("f"), new Node("d"), new
EdgeProperties<>(10.0,0.0));

// 3
ResidualNetwork<Double> test_network_3 = new ResidualNetwork<>();
test_network_3.setSource(new Node("a"));
test_network_3.setDestination(new Node("d"));
test_network_3.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(1.0,0.0));
test_network_3.addEdge(new Node("b"), new Node("c"), new
EdgeProperties<>(1.0,0.0));
test_network_3.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(1.0,0.0));
test_network_3.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(100.0,0.0));
test_network_3.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(100.0,0.0));

// 4
ResidualNetwork<Double> test_network_4=new ResidualNetwork<>();
test_network_4.setSource(new Node("a"));
test_network_4.setDestination(new Node("h"));
test_network_4.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(12.0,0.0));
test_network_4.addEdge(new Node("a"), new Node("d"), new
EdgeProperties<>(22.0,0.0));
test_network_4.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(14.0,0.0));
test_network_4.addEdge(new Node("c"), new Node("f"), new
EdgeProperties<>(21.0,0.0));
test_network_4.addEdge(new Node("d"), new Node("g"), new
EdgeProperties<>(10.0,0.0));
test_network_4.addEdge(new Node("g"), new Node("f"), new
EdgeProperties<>(10.0,0.0));

```

```

        test_network_4.addEdge(new Node("g"), new Node("h"), new
EdgeProperties<>(7.0,0.0));
        test_network_4.addEdge(new Node("f"), new Node("h"), new
EdgeProperties<>(7.0,0.0));

// 5
ResidualNetwork<Double> test_network_5 = new ResidualNetwork<>();
test_network_5.setSource(new Node("a"));
test_network_5.setDestination(new Node("f"));
test_network_5.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(5.0,0.0));
test_network_5.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(7.0,0.0));
test_network_5.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(0.0,0.0));
test_network_5.addEdge(new Node("c"), new Node("f"), new
EdgeProperties<>(12.0,0.0));
test_network_5.addEdge(new Node("d"), new Node("e"), new
EdgeProperties<>(9.0,0.0));
test_network_5.addEdge(new Node("d"), new Node("f"), new
EdgeProperties<>(6.0,0.0));
test_network_5.addEdge(new Node("e"), new Node("c"), new
EdgeProperties<>(6.0,0.0));

// 6
ResidualNetwork<Double> test_network_6 = new ResidualNetwork<>();
test_network_6.setSource(new Node("a"));
test_network_6.setDestination(new Node("f"));
test_network_6.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(0.0,0.0));
test_network_6.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(0.0,0.0));
test_network_6.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(0.0,0.0));
test_network_6.addEdge(new Node("c"), new Node("f"), new
EdgeProperties<>(0.0,0.0));
test_network_6.addEdge(new Node("d"), new Node("e"), new
EdgeProperties<>(0.0,0.0));
test_network_6.addEdge(new Node("d"), new Node("f"), new
EdgeProperties<>(0.0,0.0));
test_network_6.addEdge(new Node("e"), new Node("c"), new
EdgeProperties<>(0.0,0.0));

```

```

// 7
ResidualNetwork<Double> test_network_7 = new ResidualNetwork<>();
test_network_7.setSource(new Node("a"));
test_network_7.setDestination(new Node("f"));
test_network_7.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(5.0,0.0));
test_network_7.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(7.0,0.0));
test_network_7.addEdge(new Node("b"), new Node("a"), new
EdgeProperties<>(5.0,0.0));
test_network_7.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(0.0,0.0));
test_network_7.addEdge(new Node("c"), new Node("a"), new
EdgeProperties<>(7.0,0.0));
test_network_7.addEdge(new Node("c"), new Node("f"), new
EdgeProperties<>(12.0,0.0));
test_network_7.addEdge(new Node("d"), new Node("e"), new
EdgeProperties<>(9.0,0.0));
test_network_7.addEdge(new Node("d"), new Node("f"), new
EdgeProperties<>(6.0,0.0));
test_network_7.addEdge(new Node("e"), new Node("c"), new
EdgeProperties<>(6.0,0.0));

// 8
ResidualNetwork<Double> test_network_8=new ResidualNetwork<>();
test_network_8.setSource(new Node("a"));
test_network_8.setDestination(new Node("d"));
test_network_8.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(4.0,0.0));
test_network_8.addEdge(new Node("d"), new Node("c"), new
EdgeProperties<>(2.0,0.0));
test_network_8.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(2.0,0.0));
test_network_8.addEdge(new Node("b"), new Node("c"), new
EdgeProperties<>(3.0,0.0));
test_network_8.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(3.0,0.0));
test_network_8.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(2.0,0.0));

// 9

```

```

        ResidualNetwork<Double> test_network_9 = new ResidualNetwork<>();
        test_network_9.setSource(new Node("a"));
        test_network_9.setDestination(new Node("g"));
        test_network_9.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(4.0,0.0));
        test_network_9.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(4.0,0.0));
        test_network_9.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(3.0,0.0));
        test_network_9.addEdge(new Node("b"), new Node("e"), new
EdgeProperties<>(6.0,0.0));
        test_network_9.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(3.0,0.0));
        test_network_9.addEdge(new Node("c"), new Node("f"), new
EdgeProperties<>(6.0,0.0));
        test_network_9.addEdge(new Node("d"), new Node("e"), new
EdgeProperties<>(3.0,0.0));
        test_network_9.addEdge(new Node("d"), new Node("f"), new
EdgeProperties<>(3.0,0.0));
        test_network_9.addEdge(new Node("e"), new Node("g"), new
EdgeProperties<>(5.0,0.0));
        test_network_9.addEdge(new Node("f"), new Node("g"), new
EdgeProperties<>(5.0,0.0));

        // 10
        ResidualNetwork<Double> test_network_10=new ResidualNetwork<>();
        test_network_10.setSource(new Node("a"));
        test_network_10.setDestination(new Node("f"));
        test_network_10.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(5.0,0.0));
        test_network_10.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(5.0,0.0));
        test_network_10.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(5.0,0.0));
        test_network_10.addEdge(new Node("c"), new Node("f"), new
EdgeProperties<>(5.0,0.0));
        test_network_10.addEdge(new Node("d"), new Node("e"), new
EdgeProperties<>(5.0,0.0));
        test_network_10.addEdge(new Node("d"), new Node("f"), new
EdgeProperties<>(5.0,0.0));
        test_network_10.addEdge(new Node("e"), new Node("c"), new
EdgeProperties<>(5.0,0.0));

```

```

// 11
ResidualNetwork<Double> test_network_11=new ResidualNetwork<>();
test_network_11.setSource(new Node("a"));
test_network_11.setDestination(new Node("e"));
test_network_11.addEdge(new Node("a"),new Node("b"),new
EdgeProperties<>(20.0,0.0));
test_network_11.addEdge(new Node("b"),new Node("a"),new
EdgeProperties<>(20.0,0.0));
test_network_11.addEdge(new Node("a"),new Node("d"),new
EdgeProperties<>(10.0,0.0));
test_network_11.addEdge(new Node("d"),new Node("a"),new
EdgeProperties<>(10.0,0.0));
test_network_11.addEdge(new Node("a"),new Node("c"),new
EdgeProperties<>(30.0,0.0));
test_network_11.addEdge(new Node("c"),new Node("a"),new
EdgeProperties<>(30.0,0.0));
test_network_11.addEdge(new Node("b"),new Node("c"),new
EdgeProperties<>(40.0,0.0));
test_network_11.addEdge(new Node("c"),new Node("b"),new
EdgeProperties<>(40.0,0.0));
test_network_11.addEdge(new Node("c"),new Node("d"),new
EdgeProperties<>(10.0,0.0));
test_network_11.addEdge(new Node("d"),new Node("c"),new
EdgeProperties<>(10.0,0.0));
test_network_11.addEdge(new Node("c"),new Node("e"),new
EdgeProperties<>(20.0,0.0));
test_network_11.addEdge(new Node("e"),new Node("c"),new
EdgeProperties<>(20.0,0.0));
test_network_11.addEdge(new Node("b"),new Node("e"),new
EdgeProperties<>(30.0,0.0));
test_network_11.addEdge(new Node("e"),new Node("b"),new
EdgeProperties<>(30.0,0.0));
test_network_11.addEdge(new Node("d"),new Node("e"),new
EdgeProperties<>(10.0,0.0));
test_network_11.addEdge(new Node("e"),new Node("d"),new
EdgeProperties<>(10.0,0.0));

// 12
ResidualNetwork<Double> test_network_12=new ResidualNetwork<>();
test_network_12.setSource(new Node("a"));
test_network_12.setDestination(new Node("d"));

```



```

        test_network_12.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(2.0,0.0));
        test_network_12.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(2.0,0.0));

// 13
ResidualNetwork<Double> test_network_13 = new ResidualNetwork<>();
test_network_13.setSource(new Node("a"));
test_network_13.setDestination(new Node("h"));
test_network_13.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(5.0,0.0));
test_network_13.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(6.0,0.0));
test_network_13.addEdge(new Node("a"), new Node("d"), new
EdgeProperties<>(7.0,0.0));
test_network_13.addEdge(new Node("b"), new Node("c"), new
EdgeProperties<>(3.0,0.0));
test_network_13.addEdge(new Node("b"), new Node("e"), new
EdgeProperties<>(7.0,0.0));
test_network_13.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(4.0,0.0));
test_network_13.addEdge(new Node("d"), new Node("e"), new
EdgeProperties<>(10.0,0.0));
test_network_13.addEdge(new Node("f"), new Node("g"), new
EdgeProperties<>(4.0,0.0));
test_network_13.addEdge(new Node("f"), new Node("h"), new
EdgeProperties<>(4.0,0.0));

// 14
ResidualNetwork<Double> test_network_14 = new ResidualNetwork<>();
test_network_14.setSource(new Node("a"));
test_network_14.setDestination(new Node("h"));
test_network_14.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(6.0,0.0));
test_network_14.addEdge(new Node("b"), new Node("e"), new
EdgeProperties<>(7.0,0.0));
test_network_14.addEdge(new Node("d"), new Node("e"), new
EdgeProperties<>(10.0,0.0));
test_network_14.addEdge(new Node("f"), new Node("g"), new
EdgeProperties<>(4.0,0.0));
test_network_14.addEdge(new Node("f"), new Node("h"), new
EdgeProperties<>(4.0,0.0));

```

```

// 15
ResidualNetwork<Double> test_network_15 = new ResidualNetwork<>();
test_network_15.setSource(new Node("a"));
test_network_15.setDestination(new Node("d"));
test_network_15.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(1.5,0.0));
test_network_15.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(0.1,0.0));
test_network_15.addEdge(new Node("b"), new Node("c"), new
EdgeProperties<>(0.2,0.0));
test_network_15.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(0.1,0.0));
test_network_15.addEdge(new Node("c"), new Node("d"), new
EdgeProperties<>(0.2,0.0));

// 16
ResidualNetwork<Double> test_network_16 = new ResidualNetwork<>();
test_network_16.setSource(new Node("a"));
test_network_16.setDestination(new Node("f"));
test_network_16.addEdge(new Node("a"), new Node("b"), new
EdgeProperties<>(-5.0,0.0));
test_network_16.addEdge(new Node("a"), new Node("c"), new
EdgeProperties<>(-7.0,0.0));
test_network_16.addEdge(new Node("b"), new Node("d"), new
EdgeProperties<>(-0.0,0.0));
test_network_16.addEdge(new Node("c"), new Node("f"), new
EdgeProperties<>(-12.0,0.0));
test_network_16.addEdge(new Node("d"), new Node("e"), new
EdgeProperties<>(-9.0,0.0));
test_network_16.addEdge(new Node("d"), new Node("f"), new
EdgeProperties<>(-6.0,0.0));
test_network_16.addEdge(new Node("e"), new Node("c"), new
EdgeProperties<>(-6.0,0.0));

// 17
ResidualNetwork<Double> test_network_17 = new ResidualNetwork<>();

// 18
ResidualNetwork<Double> test_network_18 = new ResidualNetwork<>();
test_network_18.setSource(new Node("a"));
test_network_18.setDestination(new Node("f"));

```

```

        // Проверка ответов
        Assertions.assertEquals(1, test_alg.runAlgorithm(test_network_1));
        Assertions.assertEquals(6, test_alg.runAlgorithm(test_network_2));
        Assertions.assertEquals(2, test_alg.runAlgorithm(test_network_3));
        Assertions.assertEquals(14, test_alg.runAlgorithm(test_network_4));
        Assertions.assertEquals(7, test_alg.runAlgorithm(test_network_5));
        Assertions.assertEquals(0, test_alg.runAlgorithm(test_network_6));
        Assertions.assertEquals(7, test_alg.runAlgorithm(test_network_7));
        Assertions.assertEquals(5, test_alg.runAlgorithm(test_network_8));
        Assertions.assertEquals(8, test_alg.runAlgorithm(test_network_9));
        Assertions.assertEquals(10, test_alg.runAlgorithm(test_network_10));
        Assertions.assertEquals(60, test_alg.runAlgorithm(test_network_11));
        Assertions.assertEquals(0, test_alg.runAlgorithm(test_network_12));
        Assertions.assertEquals(0, test_alg.runAlgorithm(test_network_13));
        Assertions.assertEquals(0, test_alg.runAlgorithm(test_network_14));
        String test_15 =
String.format("%.1f", test_alg.runAlgorithm(test_network_15));
        Assertions.assertEquals("0,3", test_15);
        Assertions.assertEquals(-1, test_alg.runAlgorithm(test_network_16));

        boolean exceptionAppeared = false;
        try {
            test_alg.runAlgorithm(test_network_17);
        } catch (NullPointerException e) {
            exceptionAppeared = true;
        }
        Assertions.assertTrue(exceptionAppeared);
        Assertions.assertEquals(-1, test_alg.runAlgorithm(test_network_18));
    }

}

```

### Файл AddEdgeCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

public class AddEdgeCommand implements Command {
    private final Controller controller;
    private String source, destination;
    private Double capacity;

```

```

    public AddEdgeCommand(Controller controller) {
        this.controller = controller;
    }

    public void setSource(String source) {
        this.source = source;
    }

    public void setDestination(String destination) {
        this.destination = destination;
    }

    public void setCapacity(Double capacity) {
        this.capacity = capacity;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            controller.addEdge(source, destination, capacity);
            return true;
        } else {
            return false;
        }
    }
}

```

### Файл CheckSourceAndDestinationCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

public class CheckSourceAndDestinationCommand implements Command {
    private final Controller controller;

    public CheckSourceAndDestinationCommand(Controller controller) {
        this.controller = controller;
    }

    @Override
    public boolean execute() {
        if (controller != null) {

```

```

        return controller.isSourceAndDestinationCorrect();
    } else {
        return false;
    }
}
}

```

### Файл ClearNetworkCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

public class ClearNetworkCommand implements Command {
    private final Controller controller;

    public ClearNetworkCommand(Controller controller) {
        this.controller = controller;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            controller.clearNetwork();
            return true;
        } else {
            return false;
        }
    }
}

```

### Файл Command.java:

```

package leti.practice.commands;

public interface Command {
    boolean execute();
}

```

### Файл CommandType.java:

```

package leti.practice.commands;

public enum CommandType {
    LOAD_NETWORK,
    SAVE_NETWORK,
    SET_VIEW_ORIGINAL_NETWORK,
    SET_VIEW_RESIDUAL_NETWORK,
    SET_VIEW_HEIGHT_FUNCTION,
    STEP_FORWARD,
    STEP_BACKWARD,
    ADD_EDGE,
    REMOVE_EDGE,
    CLEAR_NETWORK,
    PAINT_VIEW,
    GET_NETWORK_PARAMETERS,
    RESET,
    CHECK_SOURCE_AND_DESTINATION,
    SET_SOURCE_AND_DESTINATION,
    SELECT_AND_MOVE_NODE
}

```

### Файл GetNetworkParametersCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

public class GetNetworkParametersCommand implements Command {
    private final Controller controller;
    private String networkParameters;

    public GetNetworkParametersCommand(Controller controller) {
        this.controller = controller;
    }

    public String getNetworkParameters() {
        return networkParameters;
    }

    @Override
    public boolean execute() {
        if (controller != null) {

```

```

        networkParameters = controller.getNetworkParameters();
        return networkParameters != null;
    } else {
        return false;
    }
}
}

```

### Файл LoadNetworkCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

import java.io.File;

public class LoadNetworkCommand implements Command {
    private final Controller controller;
    private File file;

    public LoadNetworkCommand(Controller controller) {
        this.controller = controller;
    }

    public void setFile(File file) {
        this.file = file;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            return controller.loadNetwork(file);
        } else {
            return false;
        }
    }
}

```

### Файл PaintViewCommand.java:

```

package leti.practice.commands;

```

```

import javafx.scene.canvas.Canvas;
import leti.practice.Controller;

public class PaintViewCommand implements Command {
    private final Controller controller;
    private final Canvas canvas;

    public PaintViewCommand(Controller controller, Canvas canvas) {
        this.controller = controller;
        this.canvas = canvas;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            controller.paintView(canvas);
            return true;
        } else {
            return false;
        }
    }
}

```

### Файл RemoveEdgeCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

public class RemoveEdgeCommand implements Command {
    private final Controller controller;
    private String source, destination;

    public RemoveEdgeCommand(Controller controller) {
        this.controller = controller;
    }

    public void setSource(String source) {
        this.source = source;
    }
}

```



```

    }

    public void setDestination(String destination) {
        this.destination = destination;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            controller.removeEdge(source, destination);
            return true;
        } else {
            return false;
        }
    }
}

```

### Файл ResetCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

public class ResetCommand implements Command {
    private final Controller controller;

    public ResetCommand(Controller controller) {
        this.controller = controller;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            controller.resetAlgorithm();
            return true;
        } else {
            return false;
        }
    }
}

```

### Файл SaveNetworkCommand.java:

```
package leti.practice.commands;

import leti.practice.Controller;

import java.io.File;

public class SaveNetworkCommand implements Command {
    private final Controller controller;
    private File file;

    public SaveNetworkCommand(Controller controller) {
        this.controller = controller;
    }

    public void setFile(File file) {
        this.file = file;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            return controller.saveNetwork(file);
        } else {
            return false;
        }
    }
}
```

### Файл SelectAndMoveNodeCommand.java:

```
package leti.practice.commands;

import leti.practice.Controller;

public class SelectAndMoveNodeCommand implements Command {
    private final Controller controller;
    private double x, y;

    public SelectAndMoveNodeCommand(Controller controller) {
        this.controller = controller;
    }
}
```

```

    }

    public void setX(double x) {
        this.x = x;
    }

    public void setY(double y) {
        this.y = y;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            controller.selectAndMoveNetworkNode(x, y);
            return true;
        } else {
            return false;
        }
    }
}

```

### Файл SetSourceAndDestinationCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

public class SetSourceAndDestinationCommand implements Command {
    private final Controller controller;
    private String source;
    private String destination;

    public SetSourceAndDestinationCommand(Controller controller) {
        this.controller = controller;
    }

    public void setSource(String source) {
        this.source = source;
    }

    public void setDestination(String destination) {
        this.destination = destination;
    }
}

```

```

    }

    @Override
    public boolean execute() {
        if (controller != null) {
            controller.setSourceAndDestination(source, destination);
            return true;
        } else {
            return false;
        }
    }
}

```

### Файл SetViewCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;
import leti.practice.gui.ViewType;

public class SetViewCommand implements Command {
    private final Controller controller;
    private final ViewType viewType;

    public SetViewCommand(Controller controller, ViewType viewType) {
        this.controller = controller;
        this.viewType = viewType;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            return controller.setView(viewType);
        } else {
            return false;
        }
    }
}

```

### Файл StepBackwardCommand.java:

```

package leti.practice.commands;

import leti.practice.Controller;

public class StepBackwardCommand implements Command {
    private final Controller controller;

    public StepBackwardCommand(Controller controller) {
        this.controller = controller;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            return controller.stepBackward();
        } else {
            return false;
        }
    }
}

```

```

    }
}

```

### Файл StepForwardCommand.java:

```

{
    private final Controller controller;
    private StepForwardResult result;
    private double algorithmResult;

    public StepForwardCommand(Controller controller) {
        this.controller = controller;
    }

    public StepForwardResult getResult() {
        return result;
    }

    public double getAlgorithmResult() {
        return algorithmResult;
    }

    @Override
    public boolean execute() {
        if (controller != null) {
            result = controller.stepForward();

            if (result == StepForwardResult.END_ALGORITHM) {
                algorithmResult = controller.getNetworkMaxFlow();
            }

            return result == StepForwardResult.SUCCESS;
        }

        return false;
    }
}

```

### Файл StepForwardResult.java:

```

package leti.practice.commands;

public enum StepForwardResult {
    SUCCESS,
    END_ALGORITHM,
    INCORRECT_NETWORK,
    ERROR
}

```

### Файл NetworkLoader.java:

```

package leti.practice.files;

import leti.practice.structures.graph.EdgeProperties;
import leti.practice.structures.graph.Node;
import leti.practice.structures.graph.ResidualNetwork;

import java.io.BufferedReader;
import java.io.File;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;

```

```

public class NetworkLoader {
    public static ResidualNetwork<Double> loadNetwork(File file) {
        if (file == null) {
            return null;
        }

        ResidualNetwork<Double> loadNetwork = new ResidualNetwork<Double>();

        try (BufferedReader reader = Files.newBufferedReader(file.toPath(),
StandardCharsets.UTF_8)) {
            String line = reader.readLine();

            if (line != null) {
                loadNetwork.setSource(new Node(line));
            } else {
                throw new Exception("Network source not found");
            }

            line = reader.readLine();

            if (line != null) {
                loadNetwork.setDestination(new Node(line));
            } else {
                throw new Exception("Network destination not found");
            }

            while ((line = reader.readLine()) != null) {
                int firstSpaceIndex = line.indexOf(' ');
                int secondSpaceIndex = line.lastIndexOf(' ');

                if (firstSpaceIndex != -1 && secondSpaceIndex != -1) {
                    Node from = new Node(line.substring(0,
firstSpaceIndex));
                    Node to = new Node(line.substring(firstSpaceIndex + 1,
secondSpaceIndex));
                    Double capacity =
Double.valueOf(line.substring(secondSpaceIndex + 1));
                    loadNetwork.addEdge(from, to, new
EdgeProperties<>(capacity, 0.0));
                } else {
                    throw new Exception("Wrong edge format");
                }
            }

        } catch (Exception e) {
            return null;
        }

        return loadNetwork;
    }
}

```

### Файл NetworkSaver.java:

```

package leti.practice.files;

import leti.practice.structures.graph.EdgeProperties;
import leti.practice.structures.graph.Node;
import leti.practice.structures.graph.ResidualNetwork;

import java.io.BufferedWriter;
import java.io.File;

```

```

import java.nio.charset.StandardCharsets;
import java.nio.file.Files;

public class NetworkSaver {
    public static boolean saveNetwork(File file, ResidualNetwork<Double>
network) {
        if (file == null || network == null) {
            return false;
        }

        try (BufferedWriter writer = Files.newBufferedWriter(file.toPath(),
StandardCharsets.UTF_8)) {
            Node source = network.getSource();
            Node destination = network.getDestination();

            writer.write(source.getName());
            writer.newLine();

            writer.write(destination.getName());
            writer.newLine();

            for (Node from : network.getNetworkNodes()) {
                for (Node to : network.getNetworkEdges(from).keySet()) {
                    EdgeProperties<Double> edgeParams =
network.getNetworkEdges(from).get(to);
                    writer.write(from.getName() + " " + to.getName() + " " +
edgeParams.getCapacity());
                    writer.newLine();
                }
            }

        } catch (Exception e) {
            return false;
        }

        return true;
    }
}

```

### Файл FileDialogType.java:

```

package leti.practice.gui;

public enum FileDialogType {
    LOAD,
    SAVE
}

```

### Файл MainWindow.java:

```

package leti.practice.gui;

import javafx.application.Platform;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.TextInputDialog;
import javafx.scene.image.Image;
import javafx.scene.layout.AnchorPane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import leti.practice.App;
import leti.practice.Controller;

```

```

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.Optional;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MainWindow {
    private static final Logger logger =
Logger.getLogger(MainWindow.class.getName());
    private Stage primaryStage;
    private MainWindowController mainWindowController;

    public MainWindow(Stage primaryStage, Controller controller) {
        this.primaryStage = primaryStage;

        primaryStage.setTitle("Goldberg Visualizer");
        primaryStage.setWidth(1000);
        primaryStage.setHeight(600);

        // Setting icon
        try (InputStream inputStream =
App.class.getResourceAsStream("/icons/app.png")) {
            if (inputStream != null) {
                Image image = new Image(inputStream);
                primaryStage.getIcons().add(image);
            } else {
                logger.log(Level.SEVERE, "Cannot open file
'/icons/app.png'.");
            }
        } catch (IOException e) {
            logger.log(Level.SEVERE, "Cannot open file '/icons/app.png'.
Exception caught.", e);
        }

        AnchorPane parent = null;

        // Setting MainWindowController
        try {
            FXMLLoader loader = new FXMLLoader();
            URL url = App.class.getResource("/fxml/MainScene.fxml");
            loader.setLocation(url);
            parent = loader.load();
            mainWindowController = loader.getController();
            mainWindowController.setMainWindow(this);
            mainWindowController.initializeCommands(controller);
        } catch (Exception e) {
            String errorMessage = "Cannot open file '/fxml/MainScene.fxml'";
            logger.log(Level.SEVERE, "Exception caught", e);
            showError(errorMessage);
            System.exit(0);
        }

        if (parent == null) {
            showError("Cannot load MainWindow scene: scene == null");
            System.exit(0);
        }

        Scene scene = new Scene(parent);

        primaryStage.setScene(scene);

```



```

        primaryStage.show();
    }

    public MainWindowController getController() {
        return mainWindowController;
    }

    public File showFileDialog(FileDialogType fileDialogType) {
        File file;
        FileChooser dialog = new FileChooser();

        dialog.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Text Files", "*.txt"));

        try {
            if (fileDialogType == FileDialogType.LOAD) {
                dialog.setTitle("Load Network");
                file = dialog.showOpenDialog(primaryStage);
            } else {
                dialog.setTitle("Save Network");
                file = dialog.showSaveDialog(primaryStage);
            }
        } catch (Exception e) {
            file = null;
        }

        return file;
    }

    public Optional<String> showTextInputDialog(String title, String header,
String content) {
        TextInputDialog dialog = new TextInputDialog();
        dialog.initOwner(primaryStage);
        dialog.setTitle(title);
        dialog.setHeaderText(header);
        dialog.setContentText(content);
        return dialog.showAndWait();
    }

    public void showDialog(Alert.AlertType alertType, String title, String
header, String content) {
        Alert alert = new Alert(alertType);
        alert.initOwner(primaryStage);
        alert.setTitle(title);
        alert.setHeaderText(header);
        alert.setContentText(content);
        alert.showAndWait();
    }

    public void showError(String content) {
        logger.log(Level.SEVERE, "Error has occurred. {0}", content);
        showDialog(Alert.AlertType.ERROR, "Error Dialog", "An error has
occurred!", content);
    }

    public void showErrorAndExit(String content) {
        showError(content);
        endApplication();
    }

    public void endApplication() {
        Platform.exit();
    }

```

```
}
```

## Файл MainWindowController.java:

```
package leti.practice.gui;

import javafx.animation.PauseTransition;
import javafx.fxml.FXML;
import javafx.scene.canvas.Canvas;
import javafx.scene.control.Alert;
import javafx.scene.control.TextArea;
import javafx.util.Duration;
import leti.practice.Controller;
import leti.practice.commands.*;

import java.io.File;
import java.util.HashMap;
import java.util.Optional;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MainWindowController {
    private static final Logger logger =
    Logger.getLogger(MainWindowController.class.getName());
    private MainWindow mainWindow;
    private HashMap<CommandType, Command> commands;
    private ViewType viewType = ViewType.RESIDUAL_NETWORK;
    private boolean intermediateMessagesEnabled = true;
    private AtomicBoolean isAnimationExecuting;

    @FXML
    private TextArea console;
    @FXML
    private TextArea parametersTextArea;
    @FXML
    private Canvas canvas;

    @FXML
    private void initialize() {
        commands = new HashMap<CommandType, Command>();
        isAnimationExecuting = new AtomicBoolean(false);

        canvas.setOnMousePressed(event -> {
            SelectAndMoveNodeCommand command = (SelectAndMoveNodeCommand)
                commands.get(CommandType.SELECT_AND_MOVE_NODE);

            command.setX(Math.max(event.getX(), 0.0));
            command.setY(Math.max(event.getY(), 0.0));
            command.execute();
            updateNetworkViewAndParameters();
        });

        canvas.setOnMouseReleased(event -> {
            SelectAndMoveNodeCommand command = (SelectAndMoveNodeCommand)
                commands.get(CommandType.SELECT_AND_MOVE_NODE);

            command.setX(Math.max(event.getX(), 0.0));
            command.setY(Math.max(event.getY(), 0.0));
            command.execute();
            updateNetworkViewAndParameters();
        });
    }
}
```

```

    public void setMainWindow(MainWindow mainWindow) {
        this.mainWindow = mainWindow;
    }

    public void printMessageToConsole(String text) {
        console.appendText(text + "\n");
    }

    public void setParametersText(String text) {
        parametersTextArea.setText(text);
    }

    public void initializeCommands(Controller controller) {
        commands.put(CommandType.ADD_EDGE, new AddEdgeCommand(controller));
        commands.put(CommandType.REMOVE_EDGE, new
RemoveEdgeCommand(controller));
        commands.put(CommandType.CLEAR_NETWORK, new
ClearNetworkCommand(controller));
        commands.put(CommandType.LOAD_NETWORK, new
LoadNetworkCommand(controller));
        commands.put(CommandType.SAVE_NETWORK, new
SaveNetworkCommand(controller));
        commands.put(CommandType.PAINT_VIEW, new
PaintViewCommand(controller, canvas));
        commands.put(CommandType.GET_NETWORK_PARAMETERS, new
GetNetworkParametersCommand(controller));
        commands.put(CommandType.SET_VIEW_ORIGINAL_NETWORK, new
SetViewCommand(controller, ViewType.ORIGINAL_NETWORK));
        commands.put(CommandType.SET_VIEW_RESIDUAL_NETWORK, new
SetViewCommand(controller, ViewType.RESIDUAL_NETWORK));
        commands.put(CommandType.SET_VIEW_HEIGHT_FUNCTION, new
SetViewCommand(controller, ViewType.HEIGHT_FUNCTION));
        commands.put(CommandType.STEP_FORWARD, new
StepForwardCommand(controller));
        commands.put(CommandType.STEP_BACKWARD, new
StepBackwardCommand(controller));
        commands.put(CommandType.RESET, new ResetCommand(controller));
        commands.put(CommandType.SET_SOURCE_AND_DESTINATION, new
SetSourceAndDestinationCommand(controller));
        commands.put(CommandType.CHECK_SOURCE_AND_DESTINATION, new
CheckSourceAndDestinationCommand(controller));
        commands.put(CommandType.SELECT_AND_MOVE_NODE, new
SelectAndMoveNodeCommand(controller));
        updateNetworkViewAndParameters();
    }

    void updateNetworkViewAndParameters() {
        GetNetworkParametersCommand getParametersCommand =
(GetNetworkParametersCommand) commands.get(
        CommandType.GET_NETWORK_PARAMETERS);
        commands.get(CommandType.PAINT_VIEW).execute();
        getParametersCommand.execute();
        setParametersText(getParametersCommand.getNetworkParameters());
    }

    @FXML
    private void buttonLoadPressed() {
        if (isAnimationExecuting.get()) {
            return;
        }

        File file = mainWindow.showFileDialog(FileDialogType.LOAD);

```

```

        if (file != null) {
            logger.log(Level.INFO, "Chosen file: " +
file.getAbsolutePath());
            LoadNetworkCommand loadCommand = (LoadNetworkCommand)
commands.get(CommandType.LOAD_NETWORK);
            loadCommand.setFile(file);
            if (!loadCommand.execute()) {
                mainWindow.showError("Error loading the network from a
file!");
            }
            updateNetworkViewAndParameters();
        }
    }

@FXML
private void buttonSavePressed() {
    File file = mainWindow.showFileDialog(FileDialogType.SAVE);

    if (file != null) {
        logger.log(Level.INFO, "Chosen file: " +
file.getAbsolutePath());
        SaveNetworkCommand saveCommand = (SaveNetworkCommand)
commands.get(CommandType.SAVE_NETWORK);
        saveCommand.setFile(file);
        if (!saveCommand.execute()) {
            mainWindow.showError("Error saving the network to a file!");
        }
    }
}

@FXML
private void buttonExitPressed() {
    logger.log(Level.INFO, "Closing application...");
    mainWindow.endApplication();
}

@FXML
private void radiobuttonOriginalNetworkSelected() {
    if (viewType != ViewType.ORIGINAL_NETWORK) {
        viewType = ViewType.ORIGINAL_NETWORK;
        logger.log(Level.INFO, "Original network view selected.");
        commands.get(CommandType.SET_VIEW_ORIGINAL_NETWORK).execute();
        commands.get(CommandType.PAINT_VIEW).execute();
    }
}

@FXML
private void radiobuttonResidualNetworkSelected() {
    if (viewType != ViewType.RESIDUAL_NETWORK) {
        viewType = ViewType.RESIDUAL_NETWORK;
        logger.log(Level.INFO, "Residual network view selected.");
        commands.get(CommandType.SET_VIEW_RESIDUAL_NETWORK).execute();
        commands.get(CommandType.PAINT_VIEW).execute();
    }
}

@FXML
private void radiobuttonHeightFunctionSelected() {
    if (viewType != ViewType.HEIGHT_FUNCTION) {
        viewType = ViewType.HEIGHT_FUNCTION;
        logger.log(Level.INFO, "Height function view selected.");
        commands.get(CommandType.SET_VIEW_HEIGHT_FUNCTION).execute();
    }
}

```

```

        commands.get (CommandType.PAINT_VIEW).execute();
    }
}

@FXML
private void checkBoxIntermediateMessagesChecked() {
    intermediateMessagesEnabled = !intermediateMessagesEnabled;

    if (intermediateMessagesEnabled) {
        logger.log(Level.INFO, "Intermediate messages enabled.");
        Logger.getLogger("leti.practice").setLevel(Level.ALL);
    } else {
        logger.log(Level.INFO, "Intermediate messages disabled.");
        Logger.getLogger("leti.practice").setLevel(Level.INFO);
    }
}

@FXML
private void buttonHelpPressed() {
    String help = ""
        In the file tab, you can load or save the network.
        In the view tab, you can switch the type of network display.
        The "Intermediate messages" parameter allows you to enable
or disable the intermediate data of the algorithm.
        Button '<-': Go back to the previous step of the algorithm.
        Button '->': Perform the next step of the algorithm.
        Button 'Steps to the finish': Run the algorithm until it is
completed.

        Button 'Reset': Reset the algorithm to its original state.
        Button 'Set the source and destination': Set the source and
destination for the network.
        Button 'Add edge': Add an edge to the network.
        Button 'Remove edge': Remove an edge from the network.
        Button 'Clear network': Clears the network.
        "";
    mainWindow.showDialog(Alert.AlertType.INFORMATION, "Help", null,
help);
}

@FXML
private void buttonAboutPressed() {
    String about = ""
        This program is designed to visualize the Goldberg algorithm
with the possibility of a detailed study of the algorithm.

        Developers: Nikita Shakhin, Rodion Kolovanov, Irina
Andrukh"";
    mainWindow.showDialog(Alert.AlertType.INFORMATION, "About", null,
about);
}

@FXML
private void setSourceAndDestinationButtonPressed() {
    if (isAnimationExecuting.get()) {
        return;
    }

    Optional<String> answer = mainWindow.showTextInputDialog("Input
Dialog", null,
        "Enter source and destination (<source> <destination>:");

    if (answer.isPresent()) {
        SetSourceAndDestinationCommand setSourceAndDestinationCommand =

```

```

                                (SetSourceAndDestinationCommand)
commands.get(CommandType.SET_SOURCE_AND_DESTINATION);
    String input = answer.get();

    boolean success = true;
    int firstSpaceIndex = input.indexOf(' ');

    try {
        if (firstSpaceIndex != -1) {
            String source = input.substring(0, firstSpaceIndex);
            String destination = input.substring(firstSpaceIndex +
1);
                                setSourceAndDestinationCommand.setSource(source);

setSourceAndDestinationCommand.setDestination(destination);
            success = setSourceAndDestinationCommand.execute();
        } else {
            success = false;
        }
    } catch (Exception e) {
        success = false;
    }

    if (!success) {
        mainWindow.showError("Error setting source and
destination.");
        return;
    }

    updateNetworkViewAndParameters();
}

@FXML
private void buttonStepBackwardPressed() {
    if (isAnimationExecuting.get()) {
        return;
    }

    commands.get(CommandType.STEP_BACKWARD).execute();
    updateNetworkViewAndParameters();
}

@FXML
private void buttonStepForwardPressed() {
    if (isAnimationExecuting.get()) {
        return;
    }

                                boolean isSourceAndDestinationSet =
commands.get(CommandType.CHECK_SOURCE_AND_DESTINATION).execute();

    if (!isSourceAndDestinationSet) {
        mainWindow.showError("The source or destination is set
incorrectly.");
        return;
    }

    StepForwardCommand stepForwardCommand = (StepForwardCommand)
commands.get(CommandType.STEP_FORWARD);

    stepForwardCommand.execute();
    updateNetworkViewAndParameters();

```

```

        switch (stepForwardCommand.getResult()) {
            case END_ALGORITHM -> {
                mainWindow.showDialog(Alert.AlertType.INFORMATION,
                    "Information Dialog", "The work of the algorithm is
completed.",
                    "Maximum network flow: " +
stepForwardCommand.getAlgorithmResult());
            }
            case INCORRECT_NETWORK -> {
                mainWindow.showError("The network is incorrect.");
            }
            case ERROR -> mainWindow.showError("Unknown error occurred.");
        }
    }

    @FXML
    private void buttonRunAlgorithmPressed() {
        boolean isSourceAndDestinationSet =
commands.get(CommandType.CHECK_SOURCE_AND_DESTINATION).execute();

        if (!isSourceAndDestinationSet) {
            mainWindow.showError("The source or destination is set
incorrectly.");
            return;
        }

        isAnimationExecuting.set(true);

        StepForwardCommand stepForwardCommand = (StepForwardCommand)
commands.get(CommandType.STEP_FORWARD);
        PauseTransition pause = new PauseTransition(Duration.seconds(1));
        pause.setOnFinished(event -> {
            boolean stepResult = stepForwardCommand.execute();
            updateNetworkViewAndParameters();

            if (stepResult) {
                pause.play();
            } else {
                isAnimationExecuting.set(false);
            }
        });
        pause.play();
    }

    @FXML
    private void buttonResetPressed() {
        if (isAnimationExecuting.get()) {
            return;
        }

        Command command = commands.get(CommandType.RESET);
        command.execute();
        updateNetworkViewAndParameters();
    }

    @FXML
    private void buttonAddEdgePressed() {
        if (isAnimationExecuting.get()) {
            return;
        }
    }

```

```

Optional<String> answer = mainWindow.showTextInputDialog("Input
Dialog", null,
    "Enter edge (<source> <destination> <capacity>:");

    if (answer.isPresent()) {
        AddEdgeCommand addEdgeCommand = (AddEdgeCommand)
commands.get(CommandType.ADD_EDGE);
        String input = answer.get();

        boolean success = true;
        int firstSpaceIndex = input.indexOf(' ');
        int secondSpaceIndex = input.lastIndexOf(' ');

        try {
            if (firstSpaceIndex != -1 && secondSpaceIndex != -1) {
                String source = input.substring(0, firstSpaceIndex);
                String destination = input.substring(firstSpaceIndex +
1, secondSpaceIndex);
                Double capacity =
Double.valueOf(input.substring(secondSpaceIndex + 1));
                addEdgeCommand.setSource(source);
                addEdgeCommand.setDestination(destination);
                addEdgeCommand.setCapacity(capacity);
                addEdgeCommand.execute();
            } else {
                success = false;
            }
        } catch (Exception e) {
            success = false;
        }

        if (!success) {
            mainWindow.showError("Error adding an edge: Invalid
format.");
            return;
        }

        updateNetworkViewAndParameters();
    }
}

@FXML
private void buttonRemoveEdgePressed() {
    if (isAnimationExecuting.get()) {
        return;
    }

    Optional<String> answer = mainWindow.showTextInputDialog("Input
Dialog", null,
        "Enter edge (<source> <destination>:");

    if (answer.isPresent()) {
        RemoveEdgeCommand removeEdgeCommand = (RemoveEdgeCommand)
commands.get(CommandType.REMOVE_EDGE);
        String input = answer.get();

        boolean success = true;
        int spaceIndex = input.indexOf(' ');

        try {
            if (spaceIndex != -1) {
                String source = input.substring(0, spaceIndex);
                String destination = input.substring(spaceIndex + 1);

```



```

        removeEdgeCommand.setSource(source);
        removeEdgeCommand.setDestination(destination);
        removeEdgeCommand.execute();
    } else {
        success = false;
    }
} catch (Exception e) {
    success = false;
}

if (!success) {
    mainWindow.showError("Error removing an edge: Invalid
format.");
    return;
}

updateNetworkViewAndParameters();
}
}

@FXML
private void buttonClearNetworkPressed() {
    if (isAnimationExecuting.get()) {
        return;
    }

    commands.get(CommandType.CLEAR_NETWORK).execute();
    updateNetworkViewAndParameters();
}
}

```

### Файл ViewType.java:

```

package leti.practice.gui;

public enum ViewType {
    ORIGINAL_NETWORK,
    RESIDUAL_NETWORK,
    HEIGHT_FUNCTION
}

```

### Файл MessageFormatter.java:

```

package leti.practice.logging;

import java.util.logging.Formatter;
import java.util.logging.LogRecord;

public class MessageFormatter extends Formatter {
    @Override
    public String format(LogRecord record) {
        return record.getMessage() + "\n";
    }
}

```

### Файл MessageHandler.java:

```

package leti.practice.logging;

import leti.practice.gui.MainWindowController;

import java.util.logging.ConsoleHandler;

```

```

import java.util.logging.LogRecord;

public class MessageHandler extends ConsoleHandler {
    private MainWindowController mainWindowController;

    public void setMainWindowController(MainWindowController
mainWindowController) {
        this.mainWindowController = mainWindowController;
    }

    @Override
    public void publish(LogRecord record) {
        super.publish(record);
        if (mainWindowController != null) {
            mainWindowController.printMessageToConsole(record.getMessage());
        }
    }
}

```

### Файл EdgeProperties.java:

```

package leti.practice.structures.graph;

import java.util.Objects;

public class EdgeProperties<T extends Number> {
    private T flow;
    private final T capacity;

    public EdgeProperties(T capacity, T flow){
        this.capacity = capacity;
        this.flow = flow;
    }
    /*не уверен что правильно копировать будет надо тестить*/
    public EdgeProperties<T> copy(){
        return new EdgeProperties<T>(capacity, flow);
    }
    public T getFlow() {
        return flow;
    }

    public void setFlow(T flow) {
        this.flow = flow;
    }

    public T getCapacity() {
        return capacity;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        EdgeProperties<?> that = (EdgeProperties<?>) o;
        return Objects.equals(flow, that.flow) && Objects.equals(capacity,
that.capacity);
    }

    @Override
    public int hashCode() {
        return Objects.hash(flow, capacity);
    }
}

```

### Файл Node.java:

```
package leti.practice.structures.graph;

import java.util.Objects;

public class Node {
    private final String name;
    public Node(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }

    public Node copy(){
        return new Node(name);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Node node = (Node) o;
        return Objects.equals(name, node.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }
}
```

### Файл ResidualNetwork.java:

```
package leti.practice.structures.graph;

import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ResidualNetwork<T extends Number> {
    private static final Logger logger =
    Logger.getLogger(ResidualNetwork.class.getName());
    private HashMap<Node, HashMap<Node, EdgeProperties<T>>> network = null;
    private HashMap<Node, HashMap<Node, EdgeProperties<T>>> reverseNetwork =
    null;
    private HashMap<Node, T> surpluses = null;
    private HashMap<Node, Integer> heights = null;
    private Node source = null;
    private Node destination = null;

    private ResidualNetwork(Node source, Node destination, HashMap<Node,
    HashMap<Node, EdgeProperties<T>>> network, HashMap<Node, HashMap<Node,
    EdgeProperties<T>>> reverseNetwork, HashMap<Node, Integer> heights, HashMap<Node,
    T> surpluses){
        this.source = source;
```

```

        this.destination = destination;
        this.network = new HashMap<>();
        this.reverseNetwork = new HashMap<>();
        this.heights = new HashMap<>();
        this.surpluses = new HashMap<>();
        for (Node from : network.keySet()) {
            this.network.put(from.copy(), new HashMap<>());
            for (Node to : network.get(from).keySet()) {
                this.network.get(from).put(to.copy(),
network.get(from).get(to).copy());
            }
        }
        for (Node from : reverseNetwork.keySet()) {
            this.reverseNetwork.put(from.copy(), new HashMap<>());
            for (Node to : reverseNetwork.get(from).keySet()) {
                this.reverseNetwork.get(from).put(to.copy(),
reverseNetwork.get(from).get(to).copy());
            }
        }
        for (Map.Entry<Node, Integer> entry : heights.entrySet()) {
            this.heights.put(entry.getKey().copy(), entry.getValue());
        }
        for (Map.Entry<Node, T> entry : surpluses.entrySet()) {
            this.surpluses.put(entry.getKey().copy(), entry.getValue());
        }
    }

    public ResidualNetwork() {
        this.network = new HashMap<>();
        reverseNetwork = new HashMap<>();
        surpluses = new HashMap<>();
        heights = new HashMap<>();
    }

    public Set<Node> getNetworkNodes() {
        return network.keySet();
    }
    public HashMap<Node, EdgeProperties<T>> getNetworkEdges(Node key) {
        return network.get(key);
    }
    public Set<Node> getReverseNetworkNodes() {
        return reverseNetwork.keySet();
    }
    public HashMap<Node, EdgeProperties<T>> getReverseNetworkEdges(Node
key) {
        return reverseNetwork.get(key);
    }
    public HashMap<Node, Integer> getHeights() {
        return heights;
    }
    public HashMap<Node, T> getSurpluses() {
        return surpluses;
    }

    public void addEdge(Node from, Node to, EdgeProperties<T>
edgeProperties) {
        /*add Edge*/
        if (from != null && to != null && edgeProperties != null) {
            Double zero = 0.0;
            if (network.containsKey(from)) {
                network.get(from).put(to, edgeProperties);
            } else {
                network.put(from, new HashMap<>());
            }
        }
    }

```

```

        network.get(from).put(to, edgeProperties);
    }

    // Add reverse edge
    if (reverseNetwork.containsKey(to)) {
        reverseNetwork.get(to).put(from, new
EdgeProperties<>((T)zero, (T)zero));
    } else {
        reverseNetwork.put(to, new HashMap<>());
        reverseNetwork.get(to).put(from, new
EdgeProperties<>((T)zero, (T)zero));
    }
}

public void deleteEdge(Node from, Node to){
    /*delete edge*/
    if(network.containsKey(from)){
        network.get(from).remove(to);
        if(network.get(from).size() == 0){ //если других рёбер больше
нет
            network.remove(from);
        }
    }
    if(reverseNetwork.containsKey(to)){
        reverseNetwork.get(to).remove(from);
        if(reverseNetwork.get(to).size() == 0){
            reverseNetwork.remove(to);
        }
    }
}

public void setSource(Node source){
    this.source = source;
}

public void setDestination(Node destination){
    this.destination = destination;
}

public Node getSource(){
    return source;
}

public Node getDestination(){
    return destination;
}

public ResidualNetwork<T> copy(){
    return new ResidualNetwork<T>(source, destination, network,
reverseNetwork, heights, surpluses);
}

public void printNetwork(){
    System.out.println("Edges:");
    for (Node from : network.keySet()) {
        for (Node to : network.get(from).keySet()) {
            System.out.println(from.getName()+" "+to.getName()+"
"+network.get(from).get(to).getCapacity()+"
"+network.get(from).get(to).getFlow());
        }
    }
    System.out.println("Reverse Edges:");
    for (Node from : reverseNetwork.keySet()) {
        for (Node to : reverseNetwork.get(from).keySet()) {

```

```

        System.out.println(from.getName()+" "+to.getName()+"
"+reverseNetwork.get(from).get(to).getCapacity()+"
"+reverseNetwork.get(from).get(to).getFlow());
    }
    }
    System.out.println();
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    ResidualNetwork<?> that = (ResidualNetwork<?>) o;
    return Objects.equals(network, that.network) &&
Objects.equals(reverseNetwork, that.reverseNetwork) && Objects.equals(surpluses,
that.surpluses) && Objects.equals(heights, that.heights) && Objects.equals(source,
that.source) && Objects.equals(destination, that.destination);
}

@Override
public int hashCode() {
    return Objects.hash(network, reverseNetwork, surpluses, heights,
source, destination);
}
}

```

## Файл HeightFunctionViewPainter.java:

```

package leti.practice.view;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.scene.transform.Affine;
import leti.practice.structures.graph.Node;
import leti.practice.structures.graph.ResidualNetwork;

import java.util.HashMap;
import java.util.HashSet;

public class HeightFunctionViewPainter extends ViewPainter {
    private final double heightMargin = 75.0;

    public HeightFunctionViewPainter() {
        nodeViewParameters = new HashMap<>();
    }

    @Override
    public void paint(ResidualNetwork<Double> network) {
        clearCanvas();

        if (network == null || !isCanvasSet()) {
            return;
        }

        calculateNodesParameters(network);

        GraphicsContext gc = getCanvas().getGraphicsContext2D();
        int nodeCount = nodeViewParameters.size();
        double arrowSize = 5;
    }
}

```

```

        double plotX = Math.max((nodeCount + 1) * heightMargin, 5 *
heightMargin);
        double plotY = Math.max((2 * nodeCount + 1) * heightMargin, 9 *
heightMargin);
        double originalLineWidth = gc.getLineWidth();
        TextAlignment originalTextAlignment = gc.getTextAlign();
        Affine originalAffineTransform = gc.getTransform();
        Font originalFont = gc.getFont();

        expandCanvas(heightMargin + plotX, heightMargin + plotY,
heightMargin);

        gc.strokeLine(heightMargin, heightMargin, heightMargin + plotX +
arrowSize, heightMargin);
        gc.strokeLine(heightMargin, heightMargin, heightMargin, heightMargin
+ plotY + arrowSize);
        gc.strokeLine(heightMargin + plotX + arrowSize, heightMargin,
heightMargin + plotX, heightMargin - arrowSize);
        gc.strokeLine(heightMargin + plotX + arrowSize, heightMargin,
heightMargin + plotX, heightMargin + arrowSize);
        gc.strokeLine(heightMargin, heightMargin + plotY + arrowSize,
heightMargin - arrowSize, heightMargin + plotY);
        gc.strokeLine(heightMargin, heightMargin + plotY + arrowSize,
heightMargin + arrowSize, heightMargin + plotY);

        gc.setFont(new Font("Arial", 14));
        gc.rotate(270);
        paintText(-150, heightMargin / 2, "NODE HEIGHT");
        gc.setTransform(originalAffineTransform);
        gc.setFont(originalFont);

        gc.setLineWidth(0.5);
        gc.setLineDashes(10);
        gc.setLineDashOffset(10);
        gc.setTextAlign(TextAlignment.LEFT);

        for (int i = 0; i <= Math.max(2 * nodeCount, 8); i++) {
            double y = heightMargin * (1 + i);
            gc.strokeLine(heightMargin, y, heightMargin + plotX, y);
            paintText(heightMargin - 15, y, String.valueOf(i));
        }

        gc.setLineDashes(0);
        gc.setLineWidth(originalLineWidth);
        gc.setTextAlign(originalTextAlignment);

        for (int i = 0; i <= Math.max(2 * nodeCount, 8); i++) {
            double y = heightMargin * (1 + i);
            gc.strokeLine(heightMargin - 3, y, heightMargin + 3, y);
        }

        paintNodes(network);
    }

    @Override
    protected void calculateNodesParameters(ResidualNetwork<Double> network)
    {
        nodeViewParameters.clear();

        int nodeIndex = 0;
        HashSet<Node> nodes = new HashSet<>(network.getNetworkNodes());
        nodes.addAll(network.getReverseNetworkNodes());
    }

```

```

        for (Node node : nodes) {
            Integer height = network.getHeights().get(node);

            if (height == null) {
                height = 0;
            }

            nodeViewParameters.put(node, new NodeViewParameters(node,
                heightMargin * (nodeIndex + 2),
                heightMargin * (height + 1), nodeIndex));
            nodeIndex++;
        }
    }
}

```

### Файл LineType.java:

```

package leti.practice.view;

public enum LineType {
    STRAIGHT,
    ARC
}

```

### Файл NetworkParametersFormatter.java:

```

package leti.practice.view;

import leti.practice.structures.graph.EdgeProperties;
import leti.practice.structures.graph.Node;
import leti.practice.structures.graph.ResidualNetwork;

import java.util.HashSet;

public class NetworkParametersFormatter {
    public static String getNetworkParameters(ResidualNetwork<Double>
network) {
        if (network == null) {
            return ""
                Nodes (<name> <surplus> <height>):

                Edges (<source> <destination> <flow> <capacity>):

                Reverse edges (<source> <destination> <flow>
<capacity>):
                """;
        }

        StringBuilder result = new StringBuilder();
        HashSet<Node> nodes = new HashSet<>(network.getNetworkNodes());

        nodes.addAll(network.getReverseNetworkNodes());
        result.append("Nodes (<name> <surplus> <height>):\n");

        for (Node node : nodes) {
            result.append(node.getName());
            result.append(" ");

            Double surplus = network.getSurpluses().get(node);
            if (surplus != null) {
                result.append(surplus);
            } else {
                result.append("-");
            }
        }
    }
}

```



```

        result.append(" ");

        Integer height = network.getHeights().get(node);
        if (height != null) {
            result.append(height);
        } else {
            result.append("-");
        }

        result.append("\n");
    }

    result.append("\nEdges    (<source>    <destination>    <flow>
<capacity>):\n");

    for (Node source : network.getNetworkNodes()) {
        for (Node destination: network.getNetworkEdges(source).keySet())
        {
            EdgeProperties<Double>    edgeParams    =
network.getNetworkEdges(source).get(destination);

            result.append(source.getName());
            result.append(" ");
            result.append(destination.getName());
            result.append(" ");
            result.append(edgeParams.getFlow());
            result.append(" ");
            result.append(edgeParams.getCapacity());
            result.append("\n");
        }
    }

    result.append("\nReverse    edges    (<source>    <destination>    <flow>
<capacity>):\n");

    for (Node source : network.getReverseNetworkNodes()) {
        for (Node    destination:
network.getReverseNetworkEdges(source).keySet()) {
            EdgeProperties<Double>    edgeParams    =
network.getReverseNetworkEdges(source).get(destination);

            result.append(source.getName());
            result.append(" ");
            result.append(destination.getName());
            result.append(" ");
            result.append(edgeParams.getFlow());
            result.append(" ");
            result.append(edgeParams.getCapacity());
            result.append("\n");
        }
    }

    return result.toString();
}
}

```

### Файл NetworkViewPainter.java:

```

package leti.practice.view;

import javafx.scene.paint.Color;
import javafx.scene.paint.Paint;
import leti.practice.structures.graph.EdgeProperties;
import leti.practice.structures.graph.Node;

```

```

import leti.practice.structures.graph.ResidualNetwork;

import java.util.*;

public abstract class NetworkViewPainter extends ViewPainter {
    protected ArrayList<Integer> layersNodeCount;
    private final double margin = 200;

    @Override
    protected void calculateNodesParameters(ResidualNetwork<Double> network)
    {
        nodeViewParameters.clear();

        if (network == null) {
            return;
        }

        Node startNode = network.getSource();
        HashSet<Node> undrawnNodes = new HashSet<>();
        undrawnNodes.addAll(network.getNetworkNodes());
        undrawnNodes.addAll(network.getReverseNetworkNodes());

        if (startNode == null || !undrawnNodes.contains(startNode)) {
            boolean fitNodeFound = false;

            for (Node node : undrawnNodes) {
                HashMap<Node, EdgeProperties<Double>> edges =
network.getNetworkEdges(node);

                if (edges != null && edges.size() > 0) {
                    startNode = node;
                    fitNodeFound = true;
                    break;
                }
            }

            if (!fitNodeFound) {
                for (Node node : undrawnNodes) {
                    startNode = node;
                    break;
                }
            }
        }

        while (!undrawnNodes.isEmpty()) {
            int layer = 0;

            NodeViewParameters currentNode = new
NodeViewParameters(startNode, margin, margin, layer);
            ArrayDeque<NodeViewParameters> frontier = new ArrayDeque<>(100);

            frontier.offer(currentNode);
            nodeViewParameters.put(currentNode.node, currentNode);
            undrawnNodes.remove(currentNode.node);

            while (!frontier.isEmpty()) {
                currentNode = frontier.poll();
                layer = currentNode.layer;

                if (layersNodeCount.size() <= layer) {
                    layersNodeCount.add(0);
                }

                currentNode.x = margin * (layer + 1);
            }
        }
    }
}

```

```

        currentNode.y = margin * (layersNodeCount.get(layer) + 1);
        layersNodeCount.set(layer, layersNodeCount.get(layer) + 1);

        HashMap<Node, EdgeProperties<Double>> edges =
network.getNetworkEdges(currentNode.node);

        if (edges != null) {
            for (Node node : edges.keySet()) {
                if (undrawnNodes.contains(node)) {
                    NodeViewParameters params = new
NodeViewParameters(node, 0, 0, layer + 1);
                    frontier.offer(params);
                    nodeViewParameters.put(node, params);
                    undrawnNodes.remove(node);
                }
            }
        }

        boolean fitNodeFound = false;

        for (Node node : undrawnNodes) {
            HashMap<Node, EdgeProperties<Double>> edges =
network.getNetworkEdges(node);

            if (edges != null && edges.size() > 0) {
                startNode = node;
                fitNodeFound = true;
                break;
            }
        }

        if (!fitNodeFound) {
            for (Node node : undrawnNodes) {
                startNode = node;
                break;
            }
        }
    }
}

```

### Файл NodeViewParameters.java:

```

package leti.practice.view;

import leti.practice.structures.graph.Node;

public class NodeViewParameters {
    public Node node;
    public double x, y;
    public int layer;

    NodeViewParameters(Node node, double x, double y, int layer) {
        this.node = node;
        this.x = x;
        this.y = y;
        this.layer = layer;
    }
}

```

### Файл OriginalNetworkViewPainter.java:

```

import leti.practice.structures.graph.ResidualNetwork;

import java.util.ArrayList;
import java.util.HashMap;

public class OriginalNetworkViewPainter extends NetworkViewPainter {
    public OriginalNetworkViewPainter() {
        nodeViewParameters = new HashMap<>();
        layersNodeCount = new ArrayList<Integer>();
    }

    @Override
    public void paint(ResidualNetwork<Double> network) {
        clearCanvas();

        if (network == null || !isCanvasSet()) {
            return;
        }

        if (needRecalculateNodesParameters) {
layersNodeCount.ensureCapacity(network.getNetworkNodes().size());
            calculateNodesParameters(network);
            layersNodeCount.clear();
        }

        paintEdges(network, false);
        paintNodes(network);
    }
}

```

### Файл ResidualNetworkViewPainter.java:

```

package leti.practice.view;

import leti.practice.structures.graph.ResidualNetwork;

import java.util.ArrayList;
import java.util.HashMap;

public class ResidualNetworkViewPainter extends NetworkViewPainter {
    public ResidualNetworkViewPainter() {
        nodeViewParameters = new HashMap<>();
        layersNodeCount = new ArrayList<Integer>();
    }

    @Override
    public void paint(ResidualNetwork<Double> network) {
        clearCanvas();

        if (network == null || !isCanvasSet()) {
            return;
        }

        if (needRecalculateNodesParameters) {
layersNodeCount.ensureCapacity(network.getNetworkNodes().size());
            calculateNodesParameters(network);
            layersNodeCount.clear();
            needRecalculateNodesParameters = false;
        }
    }
}

```

```

        paintEdges(network, true);
        paintNodes(network);
    }
}

```

## Файл ViewPainter.java:

```

package leti.practice.view;

import javafx.geometry.VPos;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.paint.Paint;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import leti.practice.structures.graph.EdgeProperties;
import leti.practice.structures.graph.Node;
import leti.practice.structures.graph.ResidualNetwork;

import java.util.HashMap;

public abstract class ViewPainter {
    private Canvas canvas = null;
    private final double nodeSize = 34;
    protected HashMap<Node, NodeViewParameters> nodeViewParameters;
    protected boolean needRecalculateNodesParameters = true;

    public abstract void paint(ResidualNetwork<Double> network);
    protected abstract void calculateNodesParameters(ResidualNetwork<Double>
network);

    public void setCanvas(Canvas canvas) {
        this.canvas = canvas;
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setLineWidth(2);
        gc.setTextAlign(TextAlignment.CENTER);
        gc.setTextBaseline(VPos.CENTER);
        gc.setFont(new Font("Arial", 10));
    }

    public Canvas getCanvas() {
        return canvas;
    }

    public boolean isCanvasSet() {
        return canvas != null;
    }

    public void expandCanvas(double x, double y, double size) {
        double additionalMargin = 100;

        if (canvas.getWidth() - (x + size) < 0) {
            canvas.setWidth(x + size + additionalMargin);
        }

        if (canvas.getHeight() - (y + size) < 0) {
            canvas.setHeight(y + size + additionalMargin);
        }
    }

    public void clearCanvas() {
        GraphicsContext gc = canvas.getGraphicsContext2D();

```

```

        gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
    }

    public void setNeedRecalculateNodesParameters(boolean value) {
        needRecalculateNodesParameters = value;
    }

    public boolean setNodePosition(String nodeName, double x, double y) {
        Node node = new Node(nodeName);

        if (nodeViewParameters.containsKey(node)) {
            NodeViewParameters nodeParams = nodeViewParameters.get(node);
            nodeParams.x = x;
            nodeParams.y = y;
            return true;
        }

        return false;
    }

    public String getNodeNameByPosition(double x, double y) {
        for (Node node : nodeViewParameters.keySet()) {
            NodeViewParameters nodeParams = nodeViewParameters.get(node);

            if (Math.abs(nodeParams.x - x) <= nodeSize &&
                Math.abs(nodeParams.y - y) <= nodeSize) {
                return node.getName();
            }
        }

        return null;
    }

    public void paintNode(double x, double y, String name) {
        paintNode(x, y, name, Color.BLACK);
    }

    public void paintNode(double x, double y, String name, Paint color) {
        expandCanvas(x, y, nodeSize);

        GraphicsContext gc = canvas.getGraphicsContext2D();
        Paint originalPaint = gc.getStroke();

        gc.setStroke(color);
        gc.setFill(Color.WHITE);
        gc.fillOval(x - nodeSize / 2, y - nodeSize / 2, nodeSize, nodeSize);
        gc.setFill(Color.BLACK);
        gc.strokeOval(x - nodeSize / 2, y - nodeSize / 2, nodeSize,
nodeSize);
        gc.fillText(name, x, y);
        gc.setStroke(originalPaint);
    }

    public void paintEdge(double sx, double sy, double dx, double dy, String
info, LineType lineType) {
        paintEdge(sx, sy, dx, dy, info, lineType, Color.BLACK);
    }

    public void paintEdge(double sx, double sy, double dx, double dy, String
info, LineType lineType, Paint color) {
        expandCanvas(Math.max(sx, dx), Math.max(sy, dy), 0);

        GraphicsContext gc = canvas.getGraphicsContext2D();

```

```

double indent = nodeSize / 4;
double distanceX = Math.abs(dx - sx);
double distanceY = Math.abs(dy - sy);
double distance = Math.sqrt(distanceX * distanceX + distanceY *
distanceY);
double sign_1 = (dy > sy)? -1 : 1;
double sign_2 = (sx > dx)? -1 : 1;
double deltaX = sign_1 * distanceY * indent / distance;
double deltaY = sign_2 * distanceX * indent / distance;

double new_sx, new_dx, new_sy, new_dy;

if (lineType == LineType.STRAIGHT) {
    new_sx = sx + deltaX;
    new_dx = dx + deltaX;
    new_sy = sy + deltaY;
    new_dy = dy + deltaY;
} else {
    new_sx = sx - deltaX;
    new_dx = dx - deltaX;
    new_sy = sy - deltaY;
    new_dy = dy - deltaY;
}

double centerX = (Math.max(new_dx, new_sx) + Math.min(new_dx,
new_sx)) / 2;
double centerY = (Math.max(new_dy, new_sy) + Math.min(new_dy,
new_sy)) / 2;

double arcCurvature = 6;
double arrowRatio = 0.5;

Paint originalPaint = gc.getStroke();
gc.setStroke(color);

if (lineType == LineType.STRAIGHT) {
    gc.strokeLine(new_sx, new_sy, new_dx, new_dy);
    gc.strokeLine(centerX, centerY,
        centerX + arrowRatio * (deltaX - deltaY), centerY +
arrowRatio * (deltaY + deltaX));
    gc.strokeLine(centerX, centerY,
        centerX + arrowRatio * (-deltaX - deltaY), centerY +
arrowRatio * (-deltaY + deltaX));
    paintText(centerX + deltaX, centerY + deltaY, info, Color.RED);
} else if (lineType == LineType.ARC) {
    gc.setLineDashes(5);
    gc.beginPath();
    gc.moveTo(new_sx, new_sy);
    gc.quadraticCurveTo(centerX - deltaX * arcCurvature, centerY -
deltaY * arcCurvature, new_dx, new_dy);
    gc.stroke();
    gc.setLineDashes(0);

    double arcCenterX = centerX - 3 * deltaX;
    double arcCenterY = centerY - 3 * deltaY;

    gc.strokeLine(arcCenterX, arcCenterY,
        arcCenterX + arrowRatio * (deltaX - deltaY), arcCenterY
+ arrowRatio * (deltaY + deltaX));
    gc.strokeLine(arcCenterX, arcCenterY, arcCenterX + arrowRatio *
(-deltaX - deltaY),
        arcCenterY + arrowRatio * (-deltaY + deltaX));
}

```

```

        paintText(centerX - deltaX * (arcCurvature - 1), centerY -
deltaY * (arcCurvature - 1), info, Color.RED);
    }

    gc.setStroke(originalPaint);
}

public void paintText(double x, double y, String text) {
    paintText(x, y, text, Color.BLACK);
}

public void paintText(double x, double y, String text, Paint paint) {
    GraphicsContext gc = canvas.getGraphicsContext2D();
    Paint originalPaint = gc.getFill();
    gc.setFill(paint);
    gc.fillText(text, x, y);
    gc.setFill(originalPaint);
}

protected void paintEdges(ResidualNetwork<Double> network, boolean
paintReverseEdges) {
    for (Node source : network.getNetworkNodes()) {
        HashMap<Node, EdgeProperties<Double>> edges =
network.getNetworkEdges(source);

        if (edges != null) {
            for (Node destination : edges.keySet()) {
                NodeViewParameters sourceParams =
nodeViewParameters.get(source);
                NodeViewParameters destParams =
nodeViewParameters.get(destination);
                EdgeProperties<Double> edgeParams =
edges.get(destination);
                String info = edgeParams.getFlow() + "/" +
edgeParams.getCapacity();

                if (sourceParams == null || destParams == null) {
                    continue;
                }

                if (edgeParams.getFlow() > 0) {
                    paintEdge(sourceParams.x, sourceParams.y,
destParams.x, destParams.y, info,
LineType.STRAIGHT, Color.BLUE);
                } else {
                    paintEdge(sourceParams.x, sourceParams.y,
destParams.x, destParams.y, info, LineType.STRAIGHT);
                }
            }
        }
    }

    if (!paintReverseEdges) {
        return;
    }

    for (Node source : network.getReverseNetworkNodes()) {
        HashMap<Node, EdgeProperties<Double>> edges =
network.getReverseNetworkEdges(source);

        if (edges != null) {
            for (Node destination : edges.keySet()) {

```



```

NodeViewParameters sourceParams =
nodeViewParameters.get(source);
NodeViewParameters destParams =
nodeViewParameters.get(destination);
EdgeProperties<Double> edgeParams =
edges.get(destination);
String info = edgeParams.getFlow() + "/" +
edgeParams.getCapacity();

        if (sourceParams == null || destParams == null) {
            continue;
        }

        paintEdge(sourceParams.x, sourceParams.y, destParams.x,
destParams.y, info, LineType.ARC);
    }
}

protected void paintNodes(ResidualNetwork<Double> network) {
    for (NodeViewParameters parameters : nodeViewParameters.values()) {
        Double height = network.getSurpluses().get(parameters.node);
        Paint color = Color.BLACK;

        if (network.getSource() != null &&
network.getSource().equals(parameters.node)) {
            color = Color.GREEN;
        } else if (network.getDestination() != null &&
network.getDestination().equals(parameters.node)) {
            color = Color.RED;
        } else if (height != null && height > 0) {
            color = Color.BLUE;
        }

        paintNode(parameters.x, parameters.y, parameters.node.getName(),
color);
    }
}
}

```

## Файл App.java:

```

package leti.practice;

import javafx.application.Application;
import javafx.stage.Stage;
import leti.practice.gui.MainWindow;
import leti.practice.logging.MessageFormatter;
import leti.practice.logging.MessageHandler;

import java.util.logging.Level;
import java.util.logging.Logger;

public class App extends Application {
    private static final Logger logger = Logger.getLogger("leti.practice");
    private static MessageHandler messageHandler;
    private MainWindow mainWindow;
    private Controller controller;

    public static void main(String[] args) {

```

```

        messageHandler = new MessageHandler();
        MessageFormatter formatter = new MessageFormatter();

        messageHandler.setLevel(Level.ALL);
        messageHandler.setFormatter(formatter);
        logger.setLevel(Level.ALL);
        logger.addHandler(messageHandler);
        logger.setUseParentHandlers(false);

        launch(args);
    }

    @Override
    public void start(Stage stage) {
        controller = new Controller();
        mainWindow = new MainWindow(stage, controller);
        messageHandler.setMainWindowController(mainWindow.getController());
    }
}

```

## Файл ConsoleController.java:

```

package leti.practice;

import leti.practice.algorithm.AlgorithmExecutor;
import leti.practice.files.NetworkLoader;
import leti.practice.logging.MessageFormatter;
import leti.practice.logging.MessageHandler;
import leti.practice.structures.graph.Node;
import leti.practice.structures.graph.ResidualNetwork;

import java.io.File;
import java.util.HashSet;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ConsoleController {
    private static final Logger logger = Logger.getLogger("leti.practice");
    private static MessageHandler messageHandler;
    private AlgorithmExecutor algorithmExecutor;
    private ResidualNetwork<Double> network;
    private boolean isNetworkCorrect;

    public ConsoleController(){
        algorithmExecutor = new AlgorithmExecutor();
        network = null;
        isNetworkCorrect = false;
        messageHandler = new MessageHandler();
        MessageFormatter formatter = new MessageFormatter();

        messageHandler.setLevel(Level.INFO);
        messageHandler.setFormatter(formatter);
        logger.setLevel(Level.ALL);
        logger.addHandler(messageHandler);
        logger.setUseParentHandlers(false);
    }

    public void run(String[] args){

```

```

        if (args.length == 1){
            printHelp();
            return;
        }

        for (int i = 0; i < args.length; ++i){
            if (args[i].equals("-f")) {
                if (i + 1 >= args.length) {
                    logger.log(Level.INFO, "The path to the file is not
specified.");
                    return;
                }
                File file = new File(args[i + 1]);
                isNetworkCorrect = loadNetworkFromFile(file);
                i++;
            } else if (args[i].equals("-h")) {
                printHelp();
            } else if (args[i].equals("-o")) {
                logger.log(Level.INFO, "The output of intermediate messages
is activated.");
                messageHandler.setLevel(Level.ALL);
            } else {
                if (!args[i].equals("-cli")) {
                    printHelp();
                    return;
                }
            }
        }

        if (isNetworkCorrect) {
            double result = runAlgorithm();
            logger.log(Level.INFO, "The maximum flow in the network is equal
to " + result);
        } else {
            logger.log(Level.INFO, "The network is not loaded.");
        }
    }

    private void printHelp(){
        System.out.println("""
        Available flags:
        -h          Print help
        -f <file>    Choose file with network
        -o          to enable intermediate outputs
        """);
    }

    private boolean isSourceAndDestinationCorrect() {
        if (network.getSource() == null || network.getDestination() == null)
        {
            return false;
        }

        HashSet<Node> nodes = new HashSet<>(network.getNetworkNodes());
        nodes.addAll(network.getReverseNetworkNodes());

        return nodes.contains(network.getSource()) &&
nodes.contains(network.getDestination());
    }

    private boolean loadNetworkFromFile(File file){
        ResidualNetwork<Double> network = NetworkLoader.loadNetwork(file);

```

```

        if (network != null) {
            this.network = network;
            return isSourceAndDestinationCorrect();
        } else {
            logger.log(Level.INFO, "Error loading the network from a
file.");
        }

        return false;
    }

    private double runAlgorithm(){
        double result = -1.0;

        try {
            result = algorithmExecutor.runAlgorithm(network);
        } catch (NullPointerException e) {
            logger.log(Level.INFO, "The network is incorrect.");
        }

        return result;
    }
}

```

### Файл Controller.java:

```

package leti.practice;

import javafx.scene.canvas.Canvas;
import leti.practice.algorithm.AlgorithmExecutor;
import leti.practice.commands.StepForwardResult;
import leti.practice.files.NetworkLoader;
import leti.practice.files.NetworkSaver;
import leti.practice.gui.MainWindow;
import leti.practice.gui.ViewType;
import leti.practice.structures.graph.EdgeProperties;
import leti.practice.structures.graph.Node;
import leti.practice.structures.graph.ResidualNetwork;
import leti.practice.view.*;

import java.io.File;
import java.util.HashSet;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Controller {
    private static final Logger logger =
Logger.getLogger(MainWindow.class.getName());
    private AlgorithmExecutor algorithmExecutor;
    private ResidualNetwork<Double> network;
    private ResidualNetwork<Double> initialNetwork;
    private NetworkViewPainter networkViewPainter;
    private ResidualNetworkViewPainter residualNetworkViewPainter;
    private HeightFunctionViewPainter heightFunctionViewPainter;
    private ViewPainter viewPainter;
    private boolean isAlgorithmRan;
    private String selectedNode;

    public Controller() {
        initialNetwork = new ResidualNetwork<Double>();
        networkViewPainter = new OriginalNetworkViewPainter();
        residualNetworkViewPainter = new ResidualNetworkViewPainter();
        heightFunctionViewPainter = new HeightFunctionViewPainter();
    }
}

```

```

        viewPainter = residualNetworkViewPainter;
        network = initialNetwork;
        algorithmExecutor = null;
        isAlgorithmRan = false;
        selectedNode = null;
    }

    public boolean loadNetwork(File file) {
        ResidualNetwork<Double> network = NetworkLoader.loadNetwork(file);

        if (network != null) {
            initialNetwork = network;
            resetAlgorithm();
            setNeedRecalculateNodesParameters(true);
            selectedNode = null;
            return true;
        }

        return false;
    }

    public boolean saveNetwork(File file) {
        return NetworkSaver.saveNetwork(file, initialNetwork);
    }

    public boolean setView(ViewType viewType) {
        if (viewType == ViewType.ORIGINAL_NETWORK) {
            viewPainter = networkViewPainter;
        } else if (viewType == ViewType.RESIDUAL_NETWORK) {
            viewPainter = residualNetworkViewPainter;
        } else if (viewType == ViewType.HEIGHT_FUNCTION) {
            viewPainter = heightFunctionViewPainter;
        } else {
            return false;
        }

        return true;
    }

    public StepForwardResult stepForward() {
        if (!isAlgorithmRan) {
            boolean isNetworkSet = false;

            initialNetwork = network.copy();
            algorithmExecutor = new AlgorithmExecutor();

            try {
                isNetworkSet = algorithmExecutor.setNetwork(network);
            } catch (NullPointerException e) {
                algorithmExecutor = null;
                return StepForwardResult.INCORRECT_NETWORK;
            }

            if (!isNetworkSet) {
                algorithmExecutor = null;
                return StepForwardResult.INCORRECT_NETWORK;
            }

            isAlgorithmRan = true;
        }

        if (algorithmExecutor != null) {
            boolean result = algorithmExecutor.nextStep();

```

```

        network = algorithmExecutor.getNetwork();

        if (result) {
            logger.log(Level.INFO, "Step Forward Command completed.");
            return StepForwardResult.SUCCESS;
        } else {
            return StepForwardResult.END_ALGORITHM;
        }
    }

    return StepForwardResult.ERROR;
}

public void setSourceAndDestination(String source, String destination) {
    if (source != null && destination != null) {
        initialNetwork.setSource(new Node(source));
        initialNetwork.setDestination(new Node(destination));
        resetAlgorithm();
        setNeedRecalculateNodesParameters(true);
    }
}

public boolean isSourceAndDestinationCorrect() {
    if (initialNetwork.getSource() == null ||
    initialNetwork.getDestination() == null) {
        return false;
    }

    HashSet<Node> nodes = new
    HashSet<>(initialNetwork.getNetworkNodes());
    nodes.addAll(initialNetwork.getReverseNetworkNodes());

    return nodes.contains(network.getSource()) &&
    nodes.contains(network.getDestination());
}

public boolean stepBackward() {
    if (!isAlgorithmRan) {
        return false;
    }

    if (algorithmExecutor != null) {
        boolean result = algorithmExecutor.previousStep();
        network = algorithmExecutor.getNetwork();

        if (result) {
            logger.log(Level.INFO, "Step Backward Command executed.");
        }

        return result;
    }

    return false;
}

public boolean addEdge(String source, String destination, Double
capacity) {
    if (source != null && destination != null && capacity != null) {
        resetAlgorithm();
        initialNetwork.addEdge(new Node(source), new Node(destination),
new EdgeProperties<Double>(capacity, 0.0));
        setNeedRecalculateNodesParameters(true);
        selectedNode = null;
    }
}

```

```

        return true;
    }

    return false;
}

public boolean removeEdge(String source, String destination) {
    if (source != null && destination != null) {
        resetAlgorithm();
        initialNetwork.deleteEdge(new Node(source), new
Node(destination));
        setNeedRecalculateNodesParameters(true);
        selectedNode = null;
        return true;
    }

    return false;
}

public void resetAlgorithm() {
    network = initialNetwork;
    algorithmExecutor = null;
    isAlgorithmRan = false;
}

public void clearNetwork() {
    logger.log(Level.INFO, "Clear Network Command completed.");

    if (viewPainter.isCanvasSet()) {
        viewPainter.clearCanvas();
        setNeedRecalculateNodesParameters(true);
        selectedNode = null;
    }

    initialNetwork = new ResidualNetwork<Double>();
    network = initialNetwork;
    algorithmExecutor = null;
    isAlgorithmRan = false;
}

public void paintView(Canvas canvas) {
    if (viewPainter != null && canvas != null) {
        viewPainter.setCanvas(canvas);
        viewPainter.paint(network);
    }
}

public String getNetworkParameters() {
    return NetworkParametersFormatter.getNetworkParameters(network);
}

public double getNetworkMaxFlow() {
    if (algorithmExecutor != null) {
        return algorithmExecutor.getMaxFlow();
    }

    return 0.0;
}

public boolean selectAndMoveNetworkNode(double x, double y) {
    if (selectedNode != null) {
        boolean result = viewPainter.setNodePosition(selectedNode, x,
y);

```

```

        if (result) {
            logger.log(Level.INFO, "Set node '" + selectedNode + "'
position to X=" + x + "; Y=" + y);
        }

        selectedNode = null;
        return result;
    } else {
        selectedNode = viewPainter.getNodeNameByPosition(x, y);

        if (selectedNode != null) {
            logger.log(Level.INFO, "Node '" + selectedNode + "'
selected.");
        }

        return selectedNode == null;
    }
}

private void setNeedRecalculateNodesParameters(boolean value) {
    networkViewPainter.setNeedRecalculateNodesParameters(value);
    residualNetworkViewPainter.setNeedRecalculateNodesParameters(value);
}
}

```

### Файл Launcher.java:

```

package leti.practice;

public class Launcher {
    public static void main(String[] args) {
        boolean isConsoleLineInterface = false;

        for (String arg : args) {
            if (arg.equals("-cli")) {
                isConsoleLineInterface = true;
                break;
            }
        }

        if (isConsoleLineInterface) {
            ConsoleController consoleController = new ConsoleController();
            consoleController.run(args);
        } else {
            try {
                App.main(args);
            } catch (NoClassDefFoundError e) {
                System.out.println("If you want to call the CLI, add the
'-cli' flag.");
            }
        }
    }
}

```



}

}