

**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: 00.06.2021

Дата защиты отчета: 00.06.2021

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
1.2.	Уточнение требований после сдачи прототипа	0
1.3.	Уточнение требований после сдачи 1-ой версии	0
1.4.	Уточнение требований после сдачи 2-ой версии	0
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.	Структура проекта	0
3.2.		0
3.3.		0
4.	Тестирование	0
4.1.	Тестирование графического интерфейса	0
4.2.	Тестирование кода алгоритма	0
4.3.	...	0
	Заключение	0
	Список использованных источников	0
	Приложение А. Исходный код программы	0

## **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 вводу исходных данных*

Пользователь должен иметь возможность задавать сеть через:

- Интерактивное добавление, изменение или удаление взвешенных ребер;
- Загрузку файла с сетью.

### *1.1.2 Требования к визуализации*

Пользователю должно быть доступно графическое представление сети и ее параметров на каждом шаге алгоритма.

Пользователь должен иметь возможность применить алгоритм Гольдберга, а также включить или выключить вывод промежуточных данных алгоритма. Должна быть возможность выполнять алгоритм пошагово (то есть сделать одну итерацию алгоритма вперед или вернуться на одну итерацию назад) или выполнять алгоритм сразу до завершения его работы.

## **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;
- 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 приложения.

### 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* - интерфейс для классов *NetworkViewPainter*, *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.

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.

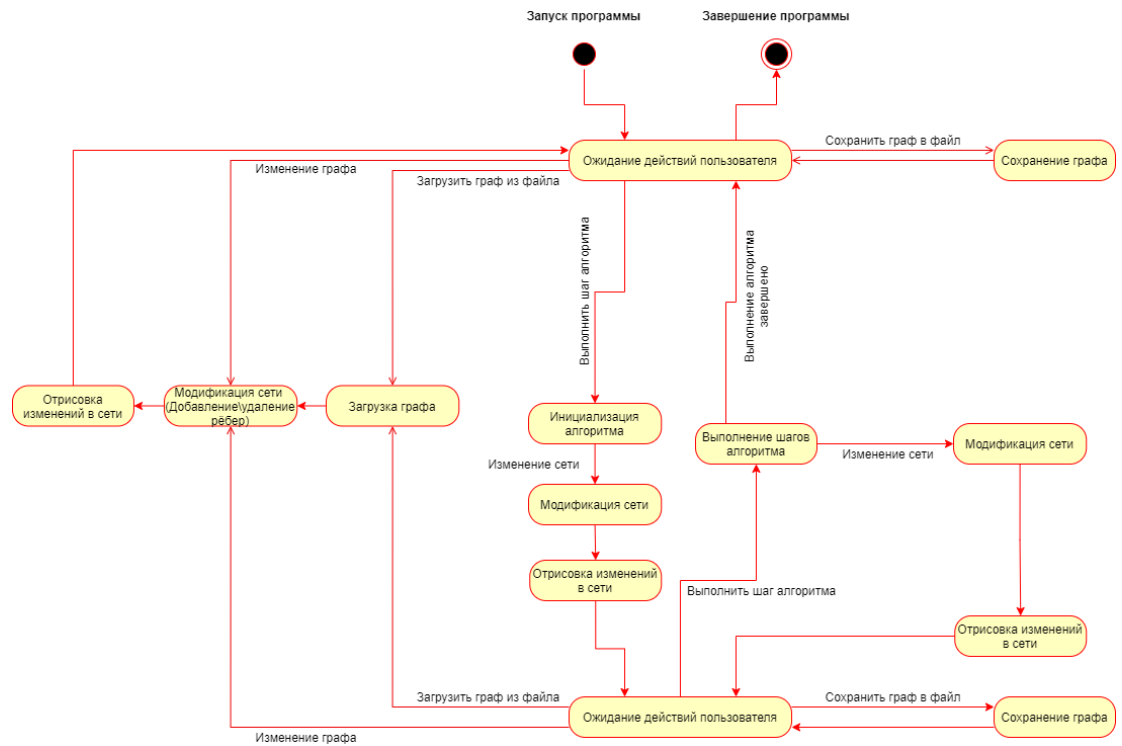


```

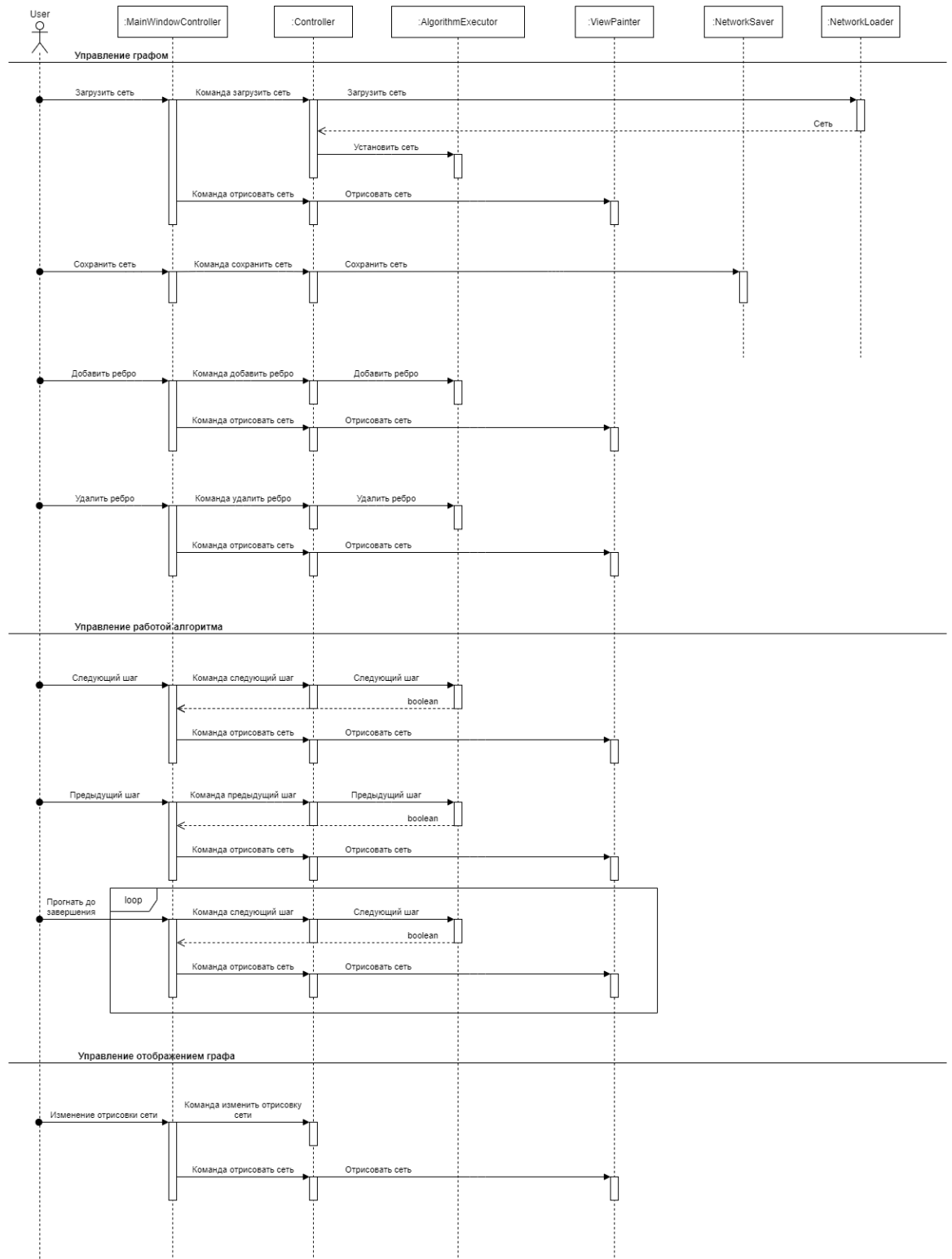
classDiagram
    class Application {
        <<javafx: application>>
    }
    class App {
        logger: Logger
        mainWindow: MainWindow
        main(String[] args)
        start(Stage stage)
    }
    class MainWindow {
        logger: Logger
        primaryStage: Stage
        showErrorMessage(String contentType, String title, String header, String content)
        showErrorMessage(String contentType)
        endApplication()
    }
    class MainWindowController {
        mainWindow: MainWindow
        viewType: ViewType
        viewModel: ResidualNetworkController
        commands: HashMap<CommandType, Command>
        console: TextArea
        initialize()
        buttonClickedPressed()
        buttonSavePressed()
        buttonLoadPressed()
        radioButtonOriginalGraphSelected()
        radioButtonResidualNetworkSelected()
        radioButtonHeightFunctionSelected()
        radioButtonMessagesSelected()
        buttonForwardPressed()
        buttonBackwardPressed()
        buttonPaintViewPressed()
        buttonAddEdgePressed()
        buttonRemoveEdgePressed()
        buttonClearNetworkPressed()
        setMainWindow()
        printMessageToConsole()
    }
    class LoadNetworkCommand {
        controller: Controller
        + LoadNetworkCommand(controller)
        + execute()
    }
    class SaveNetworkCommand {
        controller: Controller
        + SaveNetworkCommand(controller)
        + execute()
    }
    class SetViewCommand {
        controller: Controller
        viewType: ViewType
        + SetViewCommand(controller, viewType)
        + execute()
    }
    class StepForwardCommand {
        controller: Controller
        + StepForwardCommand(controller)
        + execute()
    }
    class StepBackwardCommand {
        controller: Controller
        + StepBackwardCommand(controller)
        + execute()
    }
    class AddEdgeCommand {
        controller: Controller
        sourceNode: Node
        destinationNode: Node
        capacity: Double
        + AddEdgeCommand(controller)
        + setSourceNode(node)
        + setDestinationNode(node)
        + setCapacity(double capacity)
        + execute()
    }
    class RemoveEdgeCommand {
        controller: Controller
        sourceNode: Node
        destinationNode: Node
        + RemoveEdgeCommand(controller)
        + setSourceNode(node)
        + setDestinationNode(node)
        + execute()
    }
    class PaintViewCommand {
        controller: Controller
        canvas: Canvas
        + PaintViewCommand(controller, canvas)
        + execute()
    }
    class ClearNetworkCommand {
        controller: Controller
        + ClearNetworkCommand(controller)
        + execute()
    }
    class NetworkViewPainter {
        <<javafx: javafx.scene.painter>>
        + paint(Canvas canvas, ResidualNetwork-T? network)
    }
    class ResidualNetworkViewPainter {
        + paint(Canvas canvas, ResidualNetwork-T? network)
    }
    class HeightFunctionViewPainter {
        + paint(Canvas canvas, ResidualNetwork-T? network)
    }
    class NetworkLoader {
        + load(path: String, ResidualNetwork-T)
    }
    class NetworkSaver {
        + save(ResidualNetwork-T, path: String, boolean)
    }
    class ResidualNetwork_T_Extends_Number {
        network: HashMap<Node, HashMap<Node, EdgeProperties-T>>>
        reverseNetwork: HashMap<Node, HashMap<Node, EdgeProperties-T>>>
        sources: HashMap<Node, T>
        heights: HashMap<Node, Integer>
        sourceNode: Node
        destinationNode: Node
        + ResidualNetwork()=Graph: HashMap<Node, EdgeProperties-T>=, source: String, destination: String)
        + getReverseEdges() HashMap<Node, EdgeProperties-T>
        + getReverseNetworkEdges() HashMap<Node, EdgeProperties-T>
        + getReverseNodes() Set<Node>
        + getReverseNetworkNodes() Set<Node>
        + addEdge(String source, String destination, EdgeProperties-T): void
        + removeEdge(String source, String destination): void
        + setSourceNode(String): void
        + setDestinationNode(String): void
        + copy() ResidualNetwork-T
    }
    class EdgeProperties_T_Extends_Number {
        flow: T
        capacity: T
        + EdgeProperties()=capacity: T)
        + getFlow() T
        + setFlow(flow: T): void
    }
    class Node {
        name: String
        nodeName: String
        + getName() String
    }

    Application --|> App
    App --> MainWindow
    App --> MainWindowController
    MainWindow --> MainWindowController
    MainWindowController --> NetworkViewPainter
    MainWindowController --> ResidualNetworkViewPainter
    MainWindowController --> HeightFunctionViewPainter
    MainWindowController --> NetworkLoader
    MainWindowController --> NetworkSaver
    App --> LoadNetworkCommand
    App --> SaveNetworkCommand
    App --> SetViewCommand
    App --> StepForwardCommand
    App --> StepBackwardCommand
    App --> AddEdgeCommand
    App --> RemoveEdgeCommand
    App --> PaintViewCommand
    App --> ClearNetworkCommand
    LoadNetworkCommand --> App
    SaveNetworkCommand --> App
    SetViewCommand --> App
    StepForwardCommand --> App
    StepBackwardCommand --> App
    AddEdgeCommand --> App
    RemoveEdgeCommand --> App
    PaintViewCommand --> App
    ClearNetworkCommand --> App
    NetworkViewPainter --> ResidualNetworkViewPainter
    ResidualNetworkViewPainter --> HeightFunctionViewPainter
    NetworkLoader --> NetworkSaver
    ResidualNetwork_T_Extends_Number --> EdgeProperties_T_Extends_Number
    EdgeProperties_T_Extends_Number --> Node
  
```

### 3.1.2 UML-диаграмма состояний



### 3.1.3 UML-диаграмма последовательностей



## **4. ТЕСТИРОВАНИЕ**

### **4.1. План тестирования программы.**

Для проведения тестирования планируется использовать библиотеку Junit. Необходимо протестировать корректность работы алгоритма Гольдберга (класс *AlgorithmExecutor*).

### **4.2. Результаты тестирования.**

## **ЗАКЛЮЧЕНИЕ**

## 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: 00.07.2021).
2. Java. Базовый курс // Stepik. URL: <https://stepik.org/course/187/info> (дата обращения: 04.07.2021).
3. JavaFX Reference Documentation // JavaFX. URL: <https://openjfx.io/> (дата обращения: 00.07.2021).
4. Учебник по JavaFX (Русский) // code.makery. URL: <https://code.makery.ch/ru/library/javafx-tutorial/> (дата обращения: 00.07.2021).
5. Руководства JavaFX // betacode. URL: <https://betacode.net/11009/javafx> (дата обращения: 00.07.2021).

**ПРИЛОЖЕНИЕ А**  
**ИСХОДНЫЙ КОД ПРОГРАММЫ**