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

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 выводов);
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 алгоритма 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 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;
- 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 выполнение 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.

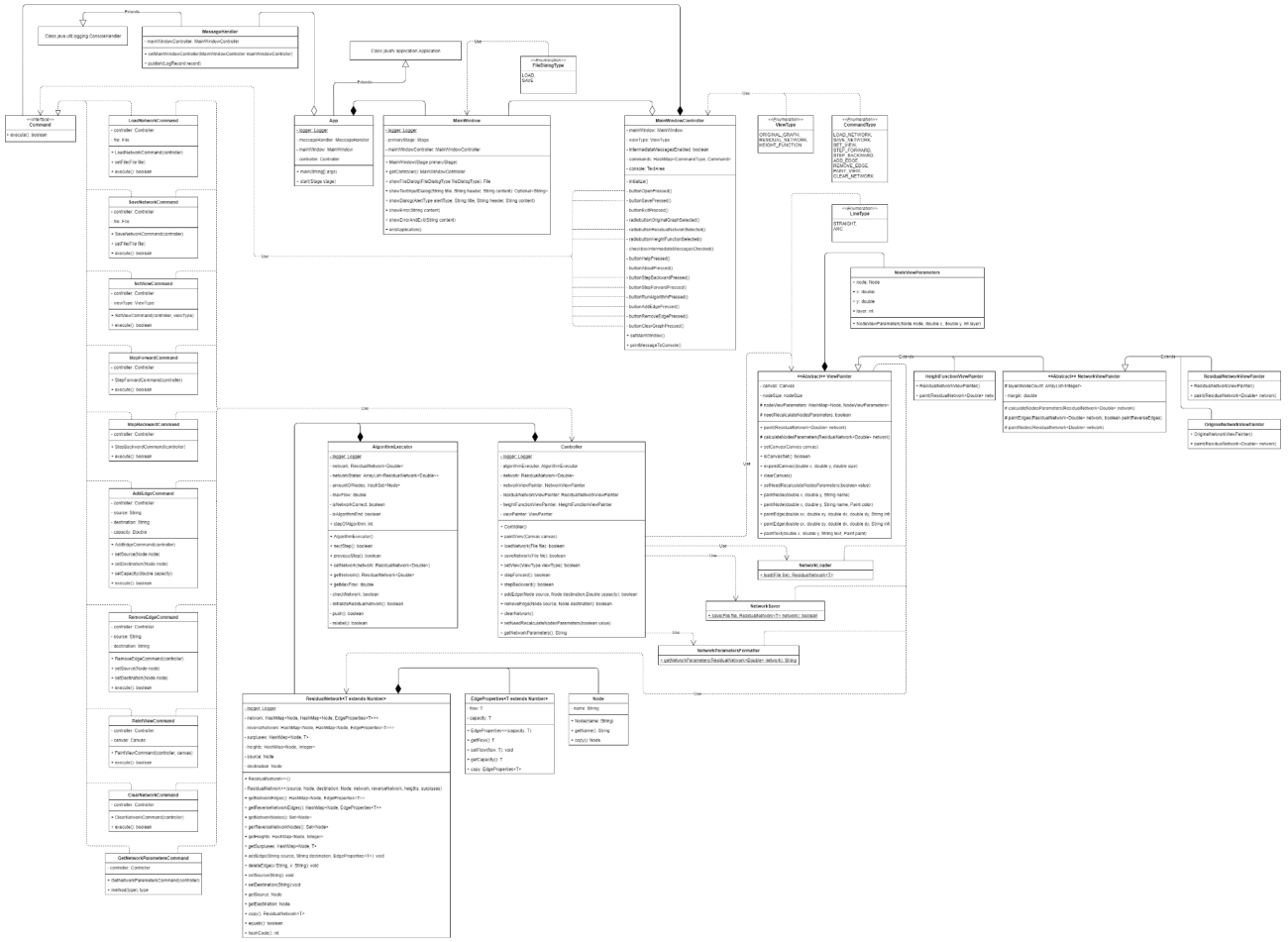


Рисунок 1 - Диаграмма классов.

3.1.2 UML-диаграмма состояний

На рисунке 2 представлена UML-диаграмма состояний.

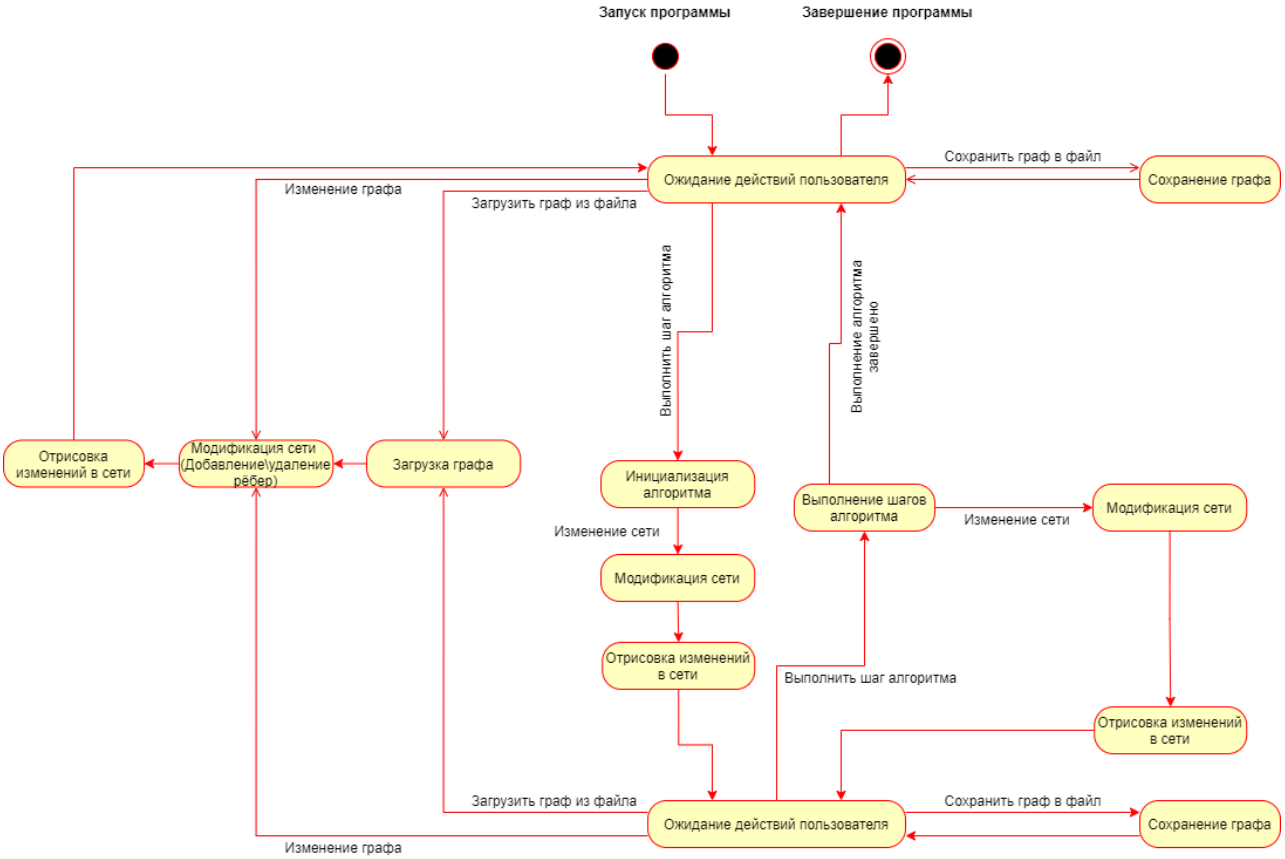


Рисунок 2 - Диаграмма состояний.

3.1.3 UML-диаграмма последовательностей

На рисунке 3 представлена UML-диаграмма последовательностей.

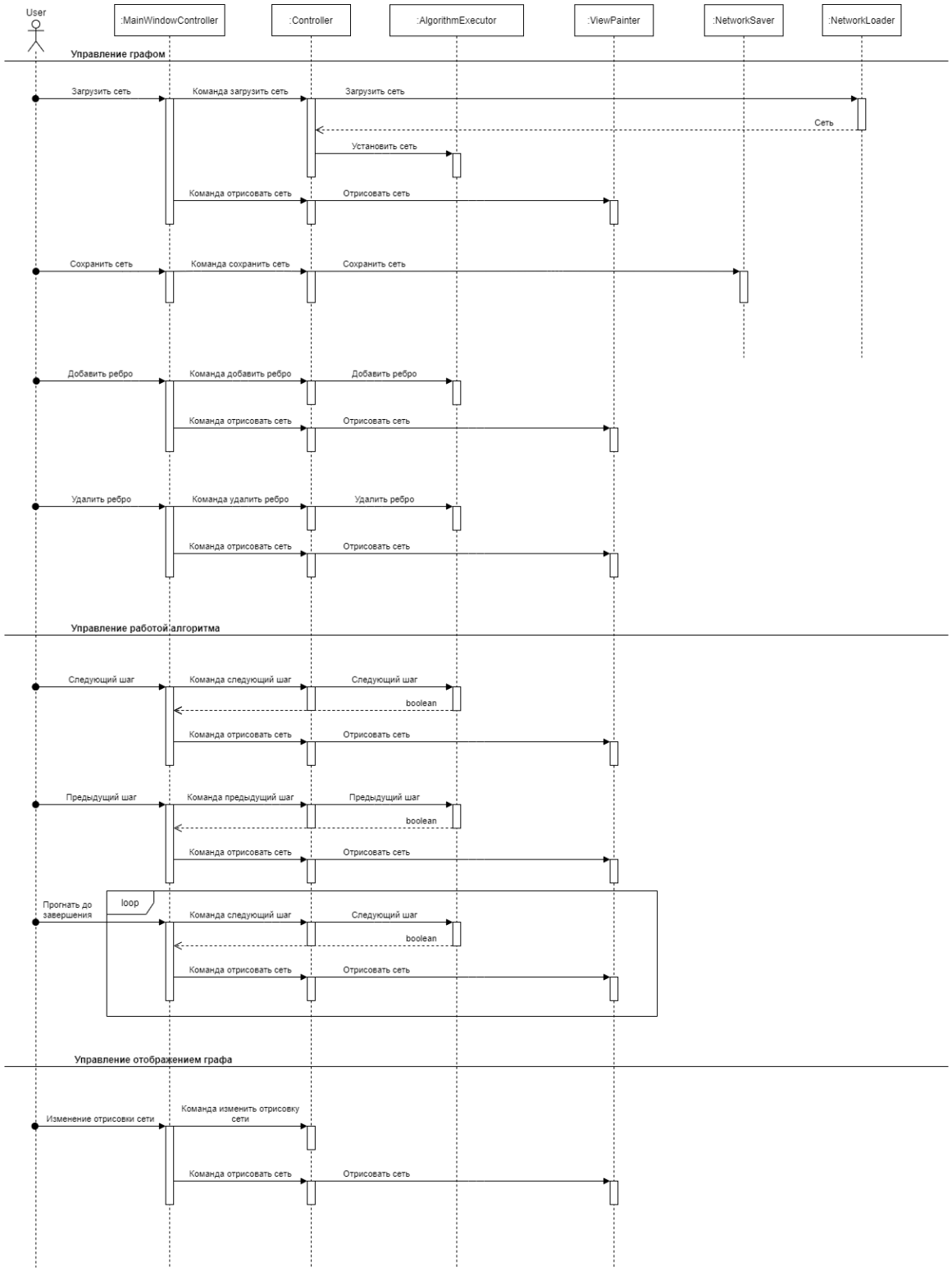


Рисунок 3 - Диаграмма последовательностей.

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 $f: V \times V \to R$, удовлетворяющую следующим свойствам:

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

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:

- $h(source) = |V|$
- $h(destination) = 0$
- 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
- $\forall (u,v) \in E\ h(u) \leqslant h(v)$ 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 самый 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 потока 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* - исток.
- *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

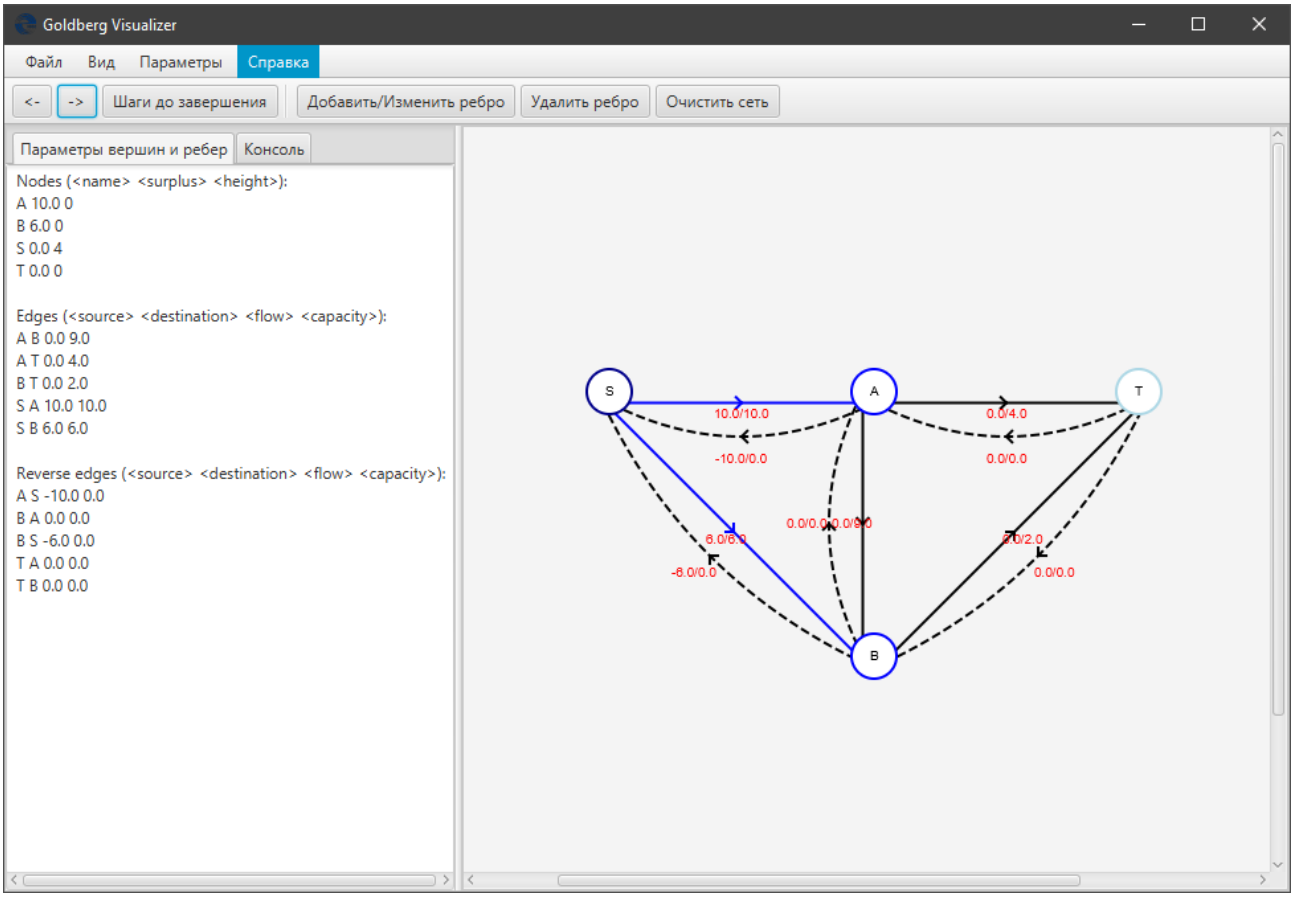
3.4.1 Описание графического интерфейса программы

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);
- 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).

Во вкладке ’Консоль’ отображается 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:



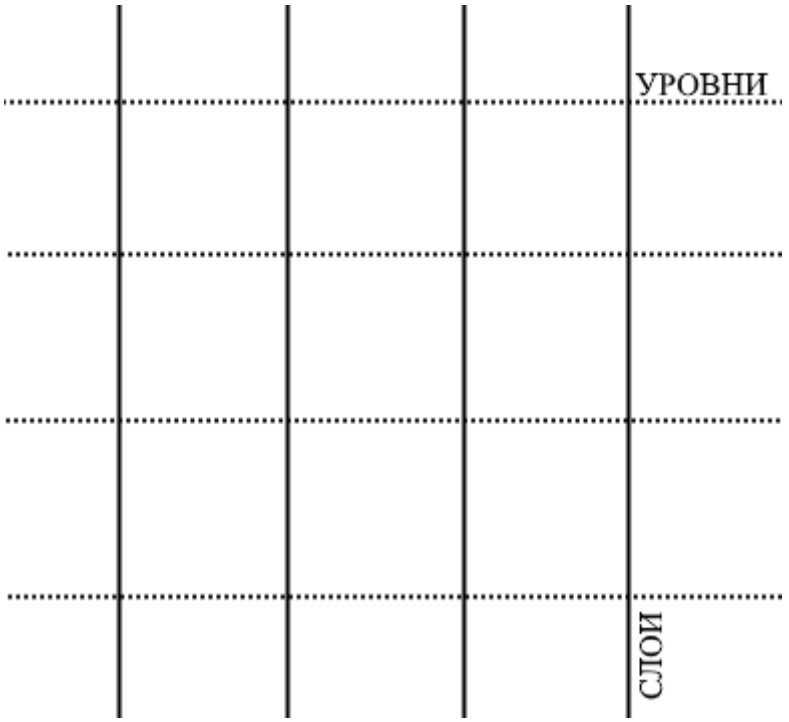
Вершины отображаются в виде окружностей, в центре которых отображаются имена вершин. Ребра исходной сети отображаются сплошной линией, а ребра обратной сети - пунктирной линией. Исток и сток окрашиваются в темно-синий и светло-синий соответственно. Ребра, через которые на данный момент проходит какой-либо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

4. ТЕСТИРОВАНИЕ

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

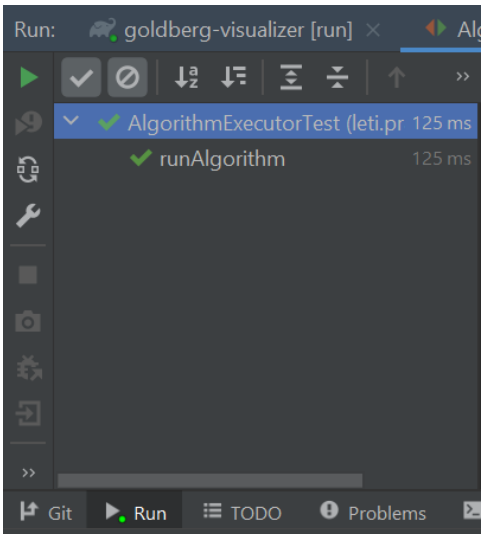
		b d 3 c d 2 d c 2	
9	Симметричный граф	a g a b 4 a c 4 b d 3 b e 6 c d 3 c f 6 d e 3 d f 3 e g 5 f g 5	8
10	Все ребра имеют одинаковый вес	a f a b 5 a c 5 b d 5 c f 5 d e 5 d f 5 e c 5	10
11	Двусторонний граф	16 a e a b 20 b a 20 a d 10 d a 10 a c 30 c a 30 b c 40 c b 40 c d 10 d c 10 c e 20 e c 20 b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

		e c -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%.

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: 07.07.2021).

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: 07.07.2021).

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 (дата обращения: 07.07.2021)

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ