

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

**ОТЧЕТ**  
**по учебной практике**  
**Тема: “Разработка визуализатора алгоритма Косарайю-Шарира”**

Студенты гр. 9381	_____	Судаков Е.В.
Студенты гр. 9381	_____	Фоминенко А.Н.
Студенты гр. 9381	_____	Авдеев И.
Руководитель	_____	Жангиров Т.Р.

Санкт-Петербург  
2021

## ЗАДАНИЕ НА ЛЕТНЮЮ ПРАКТИКУ

Студент Группы 9381 Судаков Е ,

Студент Группы 9381 Фоминенко А.

Студент Группы 9381 Авдеев И.

Тема практики: Разработка визуализатора алгоритма

Задание на практику:

Разработка визуализатора алгоритма Косарайю-Шарира на ЯП Java или Kotlin.

Алгоритм: Косарайю-Шарира

Сроки прохождения практики: 01.07.2021 - 14.07.2021

Дата сдачи отчета: 12.07.2021

Дата защиты отчета: 12.07.2021

Студенты гр. 9381

\_\_\_\_\_

Судаков Е.В.

Студенты гр. 9381

\_\_\_\_\_

Фоминенко А.Н.

Студенты гр. 9381

\_\_\_\_\_

Авдеев И.

Руководитель

\_\_\_\_\_

Жангиров Т.Р.

## **АННОТАЦИЯ**

Задача летней практики состоит в реализации бригадой из трех человек, визуализатора алгоритма на ЯП Java или Kotlin. В качестве алгоритма для визуализации был выбран алгоритм поиска компонент связности Косарайю-Шарира. Был выбран ЯП Java. В реализации интерфейса был использован JavaFX xfml.

В ходе работы была разработана программа с графическим и консольным интерфейсом для построения, сохранения графа, а также нахождения в нем компонент сильной связности.

## **SUMMARY**

The task of the summer practice is to implement an algorithm visualizer in Java or Kotlin by a team of three people. As an algorithm for visualization, the Kosarayu-Sharir connectivity component search algorithm was chosen. The Java YAP was selected. JavaFX xfml was used in the implementation of the interface.

The summer practice report consists of a specification, a development plan, and the application itself.

In the course of the work, a program was developed with a graphical and console interface for building, saving a graph, as well as finding connectivity components in it.

## СОДЕРЖАНИЕ

	Введение	6
1.	Требования к программе	7
1.1	Исходные требования к программе	7
1.1.1.	<i>Требования к вводу исходных данных</i>	7
1.1.2.	<i>Требования к визуализации</i>	7
2.	План разработки и распределение ролей в команде	8
2.1	План разработки	8
2.2	Распределение ролей в бригаде	8
3.	Особенности реализации	9
3.1	Архитектура программы	9
4.	Описание алгоритма	12
4.1.	Общее описание	12
4.2.	Шаг первый	12
4.3.	Шаг второй	12
4.4.	Шаг третий	12
5.	Описание структур данных и методов	13
5.1.	Класс APP	13
5.2.	Класс CLI	13
5.3.	Класс Graph	14
5.3.1.	<i>Класс Snapshot</i>	17
5.4.	Класс GraphFacade	18
5.5.	Класс GraphParams	19
5.6.	Класс GraphHistory	19
6.	Описание интерфейса пользователя	21
7.	Тестирование	24
7.1	План тестирования программы	24
7.1.1	<i>Объект тестирования</i>	24

7.1.2	<i>Тестируемый функционал</i>	24
7.1.3	<i>Подход к тестированию</i>	24
7.1.4	<i>Критерии прекращения тестирования</i>	24
7.2	Тестовые случаи	24
7.3	Результаты тестирования	24
	Заключение	25
	Список использованных источников	26

## **ВВЕДЕНИЕ**

### **Цель работы**

Реализация визуализатора алгоритма Косарайю-Шарира.

### **Задачи**

- Изучение ЯП Java.
- Изучение алгоритмов работающих с графами
- Получение навыков работы в команде
- Написание исходного кода программы
- Сборка программы
- Тестирование программы

### **Основные теоретические положения**

Алгоритм Косарайю-Шарира - алгоритм поиска областей сильной связности в ориентированном графе. Чтобы найти области сильной связности, сначала выполняется поиск в глубину (DFS) на обращении исходного графа (то есть против дуг), вычисляя порядок выхода из вершин. Затем мы используем обращение этого порядка, чтобы выполнить поиск в глубину на исходном графе (в очередной раз берём вершину с максимальным номером, полученным при обратном проходе). Деревья в лесе DFS, которые выбираются в результате, представляют собой сильные компоненты.

## 1. ТРЕБОВАНИЯ К ПРОГРАММЕ

### 1.1. Исходные требования к программе

Программа должна предоставлять интерфейс для пошаговой визуализации алгоритма Косарайю-Шарира.

#### *1.1.1 Требования к вводу исходных данных*

Пользователь должен иметь возможность задавать граф через:

- Интерактивное добавление, изменение или удаление взвешенных ребер при помощи диалогового ввода в GUI;
- Загрузку файла с графом.

#### *1.1.2 Требования к визуализации*

Пользователю должно быть доступно графическое представление графа и статуса выполнения алгоритма на каждом шаге.

Пользователь должен иметь возможность применить алгоритм Косарайю-Шарира, 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**

- 1.07: Распределение ролей в бригаде;
- 5.07: Создание репозитория для проекта и настройка системы автоматической сборки, подготовка отчета, разработка архитектуры и выполнение других заданий, необходимых для сдачи первой итерации;
- 8.07: Реализация алгоритма Косарайю-Шарира, реализация частичного функционала GUI и создание архитектуры для будущего тестирования;
- 11.07: Реализация взаимодействия с алгоритмом, полностью рабочий GUI и CLI;
- 13.07: Реализация дополнительного функционала.

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

Судаков Е.В. – сборка, разработка архитектуры, разработка GUI программы и визуализация алгоритма;

Фоминенко А.Н. – Реализация алгоритма Косарайю-Шарира, разработка архитектуры, реализация CLI, разработка тестов;

Авдеев И. – Тестирование приложения и помощь при подготовке документации и архитектуры



### 3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

#### 3.1. Архитектура программы

В основе архитектуры приложения взята модель MVC, обеспечивающая высокий уровень абстрагирования, а следовательно надежности, удобочитаемости и простоты кода и разработки. Ниже представлены некоторые uml-диаграммы, характеризующие логику и архитектуру построения приложения.

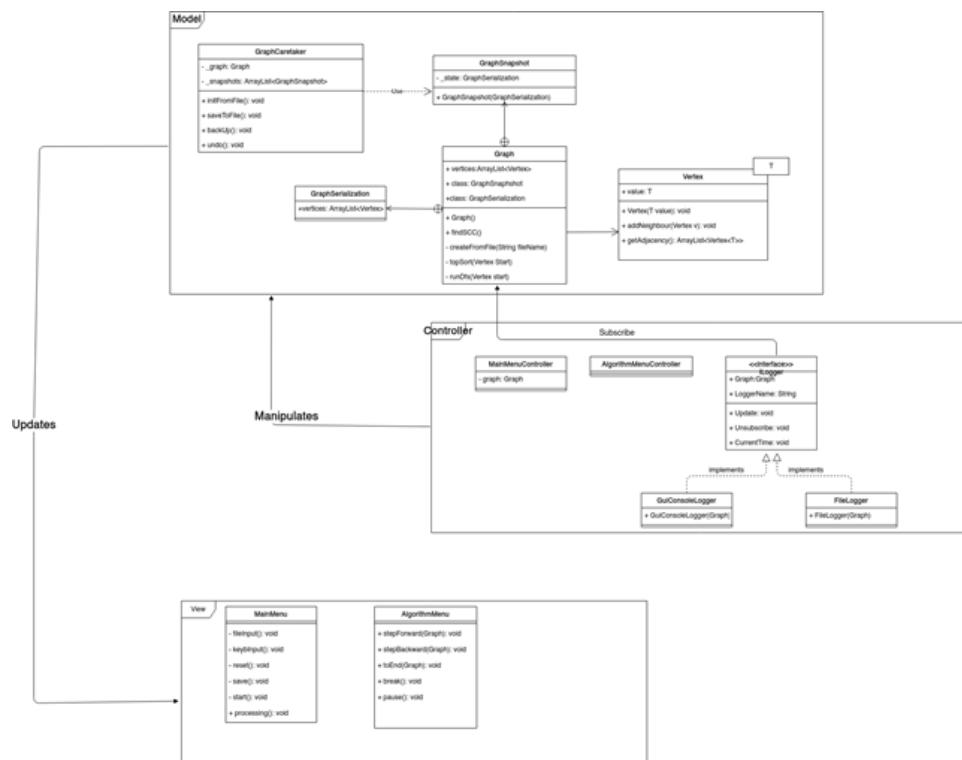


Рис 1. Uml - диаграмма классов

Диаграмма состояний

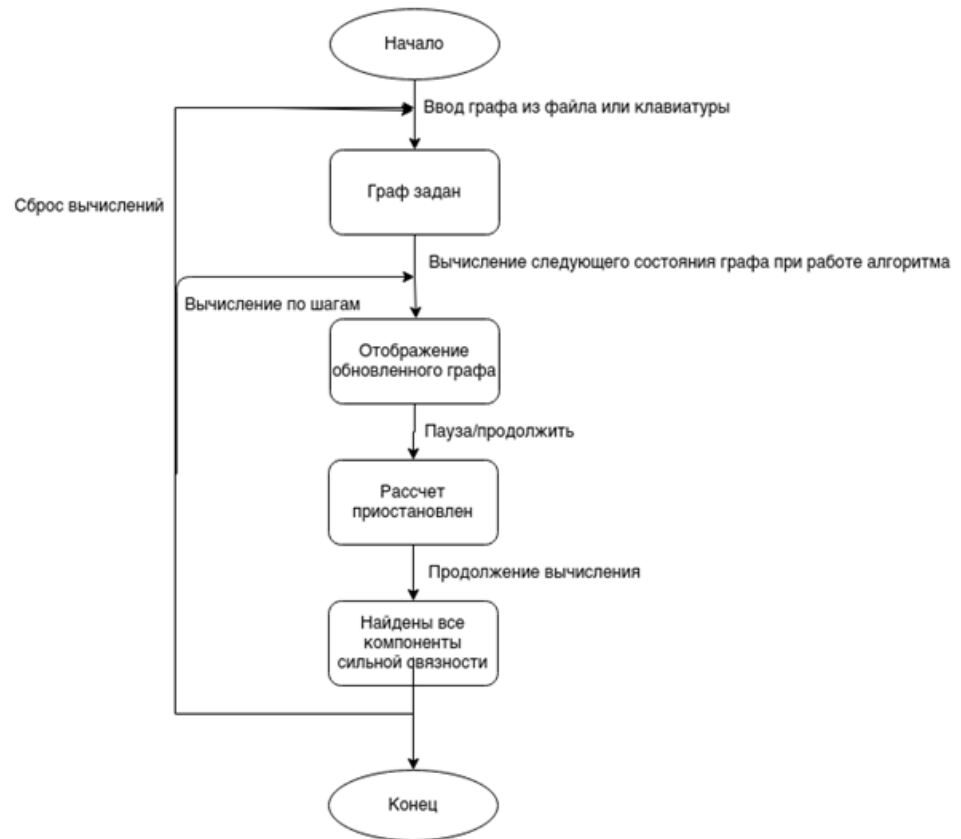


Рис 2. Uml - диаграмма состояний

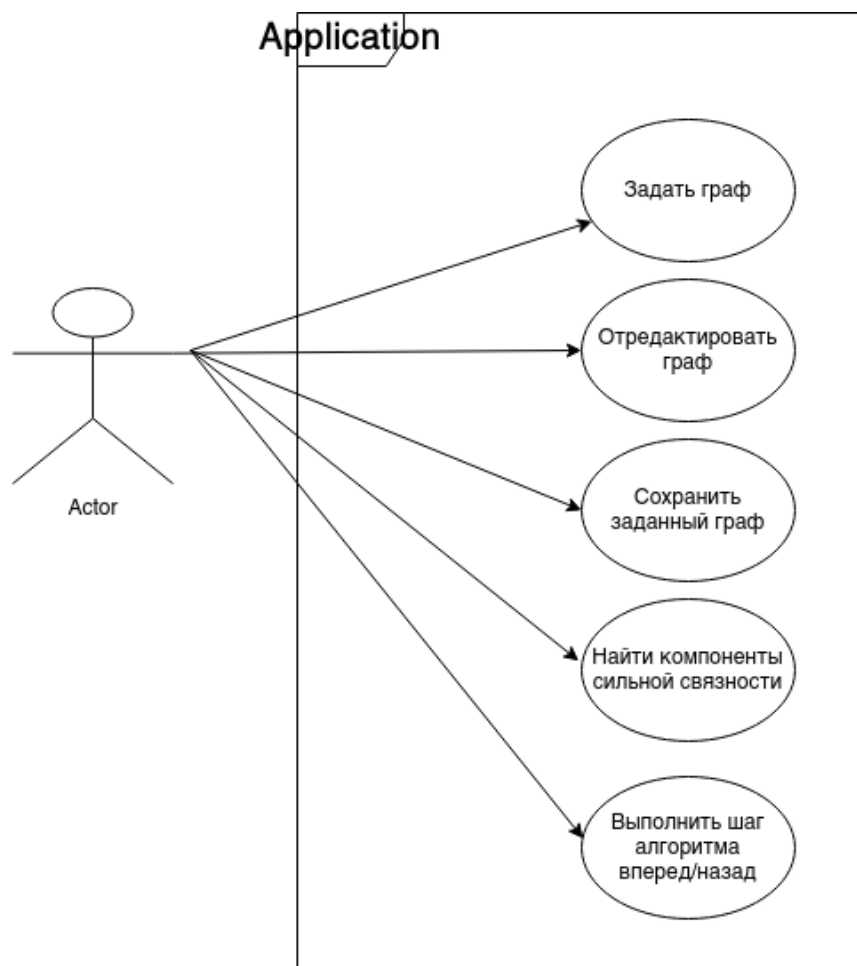


Рис 3. Uml - диаграмма возможностей

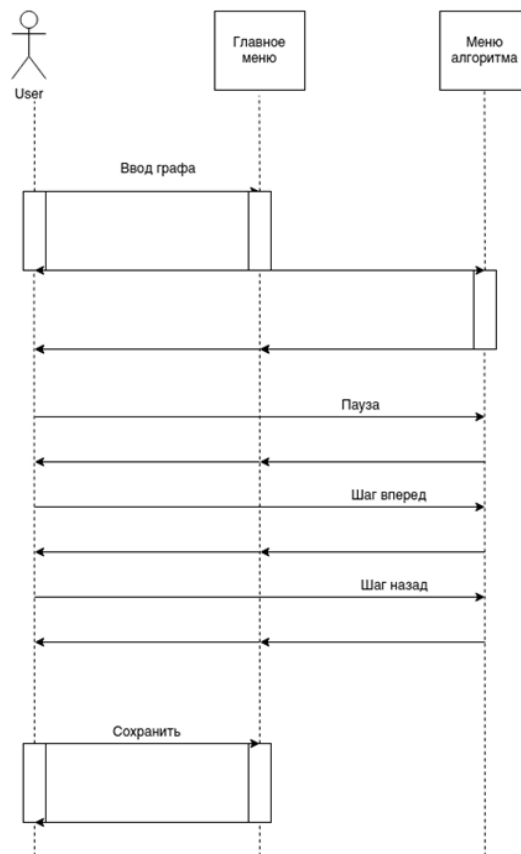


Рис 4. Uml - диаграмма последовательности

## **4. ОПИСАНИЕ АЛГОРИТМА**

### **4.1. Общее описание**

Операции графа, которые использует алгоритм, заключаются в переборе вершин графа, хранении данных вершины (если не в самой структуре данных графа, то в некоторой таблице, которая может использовать вершины в качестве индексов), переборе внешних соседей (пересекающие ребра в прямом направлении) и внутренних соседей вершины (пересекающие ребра в обратном направлении); однако можно обойтись без последнего, ценой построения транспонированного графа во время фазы прямого обхода. Единственная дополнительная структура данных, необходимая алгоритму, - это упорядоченный список  $L$  вершин графа, который будет расти, чтобы содержать каждую вершину один раз.

### **4.2. Шаг Первый**

Отметить каждую вершину как не посещенную. Список  $L$  должен быть пустым.

### **4.3. Шаг Второй**

Посещать каждую вершину, если вершина не помечена как посещенная, то отметить ее как посещенную, рекурсивно посетить внешних соседей затем добавить ее в  $L$ . Иначе ничего не делать.

### **4.4. Шаг Третий**

Для каждой вершины из  $L$  выполнять рекурсивно: Если вершина не назначена к компоненте, то назначить ее принадлежащей текущей компоненте. Иначе ничего не делать.

## 5. ОПИСАНИЕ СТРУКТУР ДАННЫХ И МЕТОДОВ

### 5.1. Класс APP

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
<i>public</i>	<i>void</i>	<i>start(Stage stage)</i>
<i>public</i>	<i>void</i>	<i>main(String[] args)</i>

#### Метод **app::start.**

Запускает GUI.

#### Метод **app::main.**

Обработывает выбор между CLI и GUI. Запуская CLI или GUI в зависимости от выбора.

### 5.2. Класс CLI

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
<i>private</i>	<i>Graph&lt;?&gt;</i>	<i>graphInput()</i>
<i>public</i>	<i>void</i>	<i>mainCLI()</i>

#### Метод **CLI::graphInput.**

Ничего не принимает. Обработывает консольный ввод типа названий вершин графа, названия вершин графа и названия ребер графа. Возвращает введенный граф.

#### Метод **CLI::mainCLI.**

Ничего не принимает. Запускает консольный ввод графа, затем применяет к нему алгоритм Косарайю-Шарира.

### 5.3. Класс Graph<>

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
<i>public</i>	<i>Integer numberOfEdges</i>	Хранит количество ребер в графе	0
<i>public</i>	<i>GraphHistory history</i>	Хранит состояния графа на предыдущих шагах	-
<i>private</i>	<i>Map&lt;T, ArrayList&lt;T&gt;&gt; graphAdjacencyList</i>	Список смежности графа	-
<i>private</i>	<i>Map&lt;T, ArrayList&lt;T&gt;&gt; graphTransposedAdjacencyList</i>	Список смежности транспонированного графа	-
<i>private</i>	<i>Map&lt;T, Boolean&gt; used</i>	Список Маркеров посещенности вершин	-
<i>private</i>	<i>Map&lt;T, GraphParams&gt; params</i>	Параметры вершин	-
<i>private</i>	<i>ArrayList&lt;T&gt; allVertex</i>	Список всех вершин	-
<i>private</i>	<i>ArrayList&lt;T&gt; order</i>	Порядок обхода	-
<i>private</i>	<i>ArrayList&lt;T&gt; component</i>	Список компонент связностей	-

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
<i>private</i>		Graph(Map<T, ArrayList<T>>, Map<T, ArrayList<T>>, Map<T, Boolean>, Map<T, GraphParams>, ArrayList<T>, ArrayList<T>, ArrayList<T>)
<i>public</i>		Graph(Graph<T>)
<i>public</i>		Graph()
<i>public</i>	<i>Snapshot</i>	createSnapshot()
<i>public</i>	<i>Map&lt;T, ArrayList&lt;T&gt;&gt;</i>	getGraphAdjacencyList()
<i>public</i>	<i>Map&lt;T, ArrayList&lt;T&gt;&gt;</i>	getGraphTransposedAdjacencyList()

<i>public</i>	<i>Map&lt;T, Boolean&gt;</i>	<i>getUsed()</i>
<i>public</i>	<i>Map&lt;T, GraphParams&gt;</i>	<i>getParams()</i>
<i>public</i>	<i>ArrayList&lt;T&gt;</i>	<i>getAllVertex()</i>
<i>public</i>	<i>ArrayList&lt;T&gt;</i>	<i>getOrder()</i>
<i>public</i>	<i>ArrayList&lt;T&gt;</i>	<i>getComponent()</i>
<i>public</i>	<i>void</i>	<i>setGraphAdjacencyList( Map&lt;T, ArrayList&lt;T&gt;&gt;)</i>
<i>public</i>	<i>void</i>	<i>setGraphTransposedAdjacencyList( Map&lt;T, ArrayList&lt;T&gt;&gt;)</i>
<i>public</i>	<i>void</i>	<i>setUsed(Map&lt;T, Boolean&gt;)</i>
<i>public</i>	<i>void</i>	<i>setParams(Map&lt;T, GraphParams&gt;)</i>
<i>public</i>	<i>void</i>	<i>setAllVertex(ArrayList&lt;T&gt;)</i>
<i>public</i>	<i>void</i>	<i>setOrder(ArrayList&lt;T&gt;)</i>
<i>public</i>	<i>void</i>	<i>setComponent(ArrayList&lt;T&gt;)</i>
<i>public</i>	<i>Graph</i>	<i>getInstance()</i>
<i>public</i>	<i>void</i>	<i>addVertex(T)</i>
<i>public</i>	<i>int</i>	<i>getNum(T)</i>
<i>public</i>	<i>void</i>	<i>addEdge(T, T)</i>
<i>public</i>	<i>boolean</i>	<i>hasVertex(T)</i>
<i>public</i>	<i>boolean</i>	<i>hasEdge(T, T)</i>
<i>public</i>	<i>String</i>	<i>toString()</i>
<i>private</i>	<i>void</i>	<i>dfs1(T)</i>
<i>private</i>	<i>void</i>	<i>dfs2(T)</i>
<i>private</i>	<i>void</i>	<i>printComponent(int)</i>
<i>public</i>	<i>void</i>	<i>mainAlgo()</i>

**Метод Graph<>:: Graph(Map<T, ArrayList<T>>,  
Map<T, ArrayList<T>> ,  
Map<T, Boolean>,  
Map<T, GraphParams>,  
ArrayList<T>,  
ArrayList<T>,  
ArrayList<T>).**

Конструктор, который создает граф из отправленных в аргументы параметров.

**Метод Graph<>:: Graph(Graph<T> graph).**

Конструктор, который создает граф копируя граф отправленный в аргументы



**Метод `Graph<>::Graph()`.**

Конструктор.

**Метод `Graph<>::createSnapshot`.**

Создает снимок графа на шаге. Снимок храниться в объекте класс `Snapshot`.

Возвращает этот объект.

**Метод `Graph<>::setGraphAdjacencyList`.**

Задает или меняет Список смежности графа.

**Метод `Graph<>::setGraphTransposedAdjacencyList`.**

Задает или меняет список смежности транспонированного графа.

**Метод `Graph<>::setUsed`.**

Задает или меняет маркер посещенности вершины.

**Метод `Graph<>::setParams`.**

**Метод `Graph<>::setAllVertex`.**

Задает или изменяет коллекцию всех вершин графа.

**Метод `Graph<>::setOrder`.**

Позволяет задать или изменить порядок обхода графа.

**Метод `Graph<>::setComponent`.**

Позволяет задать или изменить компоненты связности.

**Метод `Graph<>::getInstance`.**

Позволяет получить граф. Возвращает объект класса графа.

**Метод Graph<>:: addVertex.**

Позволяет добавить вершину в граф.

**Метод Graph<>:: getNum**

Позволяет получить время захода. Возвращает целое число

**Метод Graph<>:: hasVertex.**

Позволяет проверить наличие в списке смежности вершины

**Метод Graph<>:: hasEdge.**

Позволяет проверить наличие в списке смежности ребро

**Метод Graph<>:: toString.**

Позволяет вывести граф в виде строки.

**Метод Graph<>:: dfs1.**

Обход в глубину графа

**Метод Graph<>:: dfs2.**

Обход в глубину транспонированного графа.

**Метод Graph<>:: printComponent.**

Выводит в консоль компоненты сильной связности

**Метод Graph<>:: mainAlgo.**

Алгоритм поиска компонент сильной связности в графе.

**5.3.1 Класс Snapshot**

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
<i>private final</i>	<i>Graph&lt;T&gt; graph</i>	Хранит граф	-
<i>private final</i>	<i>Map&lt;T, ArrayList&lt;T&gt;&gt; graphAdjacencyList</i>	Хранит список смежности графа	-
<i>private final</i>	<i>Map&lt;T, ArrayList&lt;T&gt;&gt; graphTransposedAdjacencyList</i>	Хранит список смежности транспонированного графа	-
<i>private final</i>	<i>Map&lt;T, Boolean&gt; used</i>	Карта маркеров посещенности вершин	-
<i>private final</i>	<i>Map&lt;T, GraphParams&gt; params</i>		-
<i>private final</i>	<i>ArrayList&lt;T&gt; allVertex</i>	Список всех вершин графа	-
<i>private final</i>	<i>ArrayList&lt;T&gt; order</i>	Направление сторон	-
<i>private final</i>	<i>ArrayList&lt;T&gt; component</i>	Список компонент сильной связности	-

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
<i>public</i>		Snapshot(Graph<T>, Map<T, ArrayList<T>>, Map<T, ArrayList<T>>, Map<T, Boolean> used, Map<T, GraphParams>, ArrayList<T>, ArrayList<T>, ArrayList<T>)
<i>public</i>	<i>void</i>	<i>restore()</i>

### Метод **Snapshot::Snapshot.**

Конструктор, который создает снимок графа.

### Метод **Snapshot::restore.**

Позволяет обновить снимок графа, заменив его на текущее состояние графа.

#### 5.4 Класс GraphFacade

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
<i>public</i>	<i>Graph graph</i>	Хранит граф	-

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
<i>public</i>		GraphFacade( BufferedReader graphBR)

**Метод GraphFacade::GraphFacade.**

Конструктор.

#### 5.5 Класс GraphParams

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
<i>public</i>	<i>Int num</i>	Тестовое поле	<i>-1</i>

#### 5.6 Класс GraphHistory

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
<i>private</i>	<i>ArrayList&lt;Graph.Snapshot&gt; snapshots</i>	Список снимков графа	-
<i>private</i>	<i>Graph graph</i>	Хранит граф	-

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
<i>public</i>	<i>int</i>	getSize()
<i>public</i>		GraphHistory(Graph )
<i>public</i>	<i>void</i>	backUp()

<i>public</i>	<i>Graph.Snapshot</i>	getSnapshot(int )
---------------	-----------------------	-------------------

### **Метод GraphHistory:: GraphHistory.**

Конструктор. Делает снимок текущего состояния графа и сохраняет его в переменной graph

### **Метод GraphHistory::getSize.**

Позволяет получить размер списка снимков графа. Возвращает размер списка.

### **Метод GraphHistory:: backUp.**

Добавляет текущий снимок графа в список снимков графа.

### **Метод GraphHistory:: getSnapshot.**

Позволяет получить снимок из списка снимков по индексу. Возвращает объект snapshot.

## 6. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

На вход программе подается ориентированный граф, который задается в текстовом виде через диалоговое окно или из файла. После завершения алгоритма полученные компоненты сильной связности отображаются на главном окне, также присутствует возможность записи полученного результата в текстовый файл. В ходе выполнения алгоритма пользователь может видеть предыдущее и текущее состояние программы.

Ниже на рисунке представлено графический интерфейс пользователя с интуитивно понятным интерфейсом.

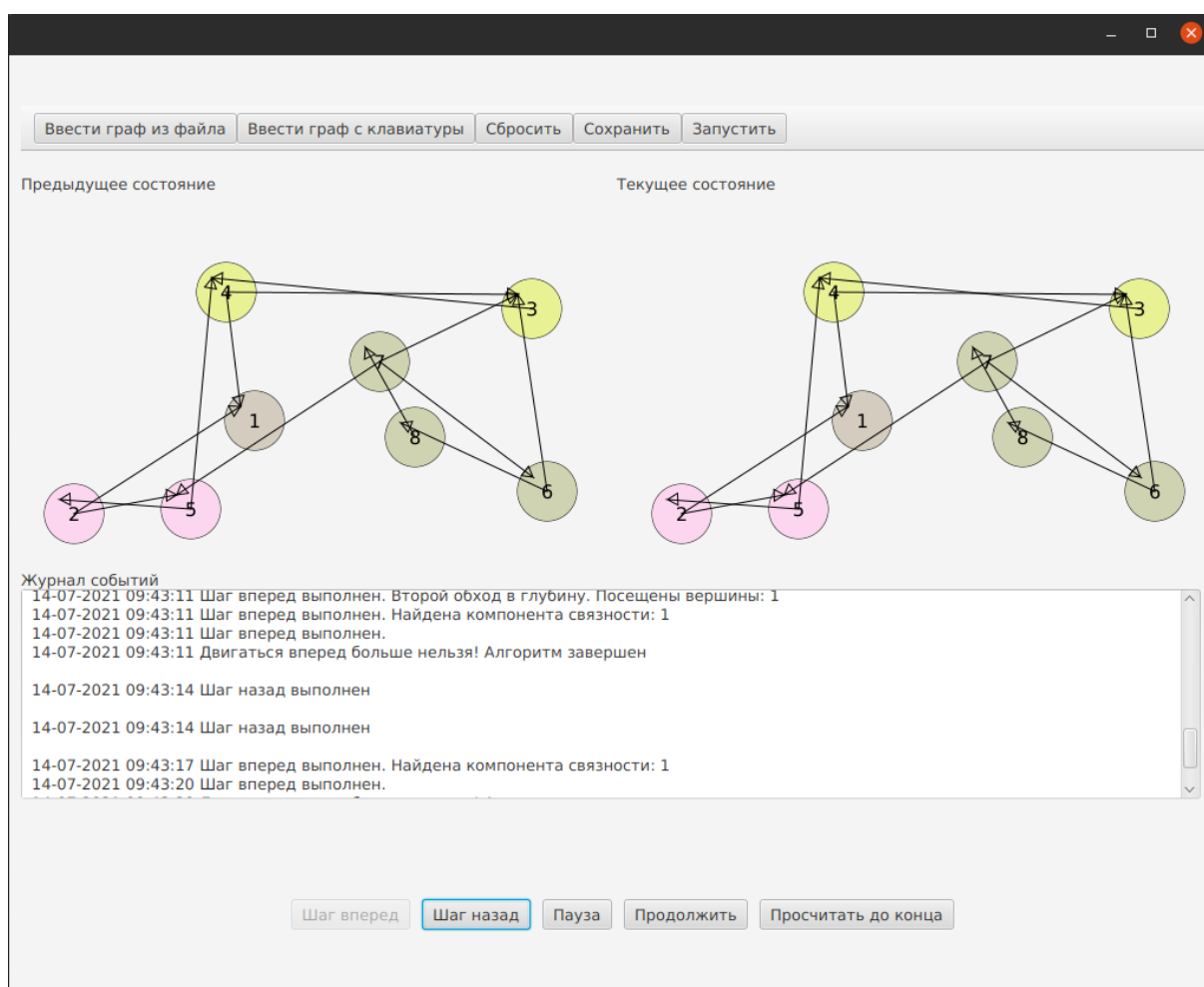


Рис 5. Реализованное окно исполнения алгоритма



## 7. ТЕСТИРОВАНИЕ

### 7.1. План тестирования программы.

#### 7.1.1. Объект тестирования

Объектом тестирования является программа для визуализации работы алгоритма Косарайю-Шарира.

#### 7.1.2. Тестируемый функционал.

Необходимо протестировать метод `mainAlgo()` класса `Graph`, так как в этом методе происходит полное выполнение алгоритма.

#### 7.1.3. Подход к тестированию.

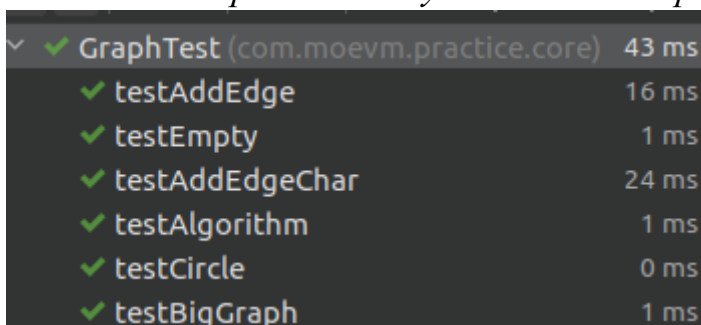
Тестирование будет проводиться на модульном уровне при помощи фреймворка автоматического тестирования JUnit.

#### 7.1.4. Критерии прекращения тестирования.

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

### 7.3. Результаты тестирования.

#### 7.3.1. Скриншот запуска unit-тестирования:

A screenshot of a JUnit test runner window showing the results of a test run for the class GraphTest. The window has a dark background with green checkmarks indicating successful tests. The tests listed are testAddEdge, testEmpty, testAddEdgeChar, testAlgorithm, testCircle, and testBigGraph, each with its execution time in milliseconds.

✓ GraphTest (com.moevm.practice.core)	43 ms
✓ testAddEdge	16 ms
✓ testEmpty	1 ms
✓ testAddEdgeChar	24 ms
✓ testAlgorithm	1 ms
✓ testCircle	0 ms
✓ testBigGraph	1 ms

Таким образом, тестирование можно считать пройденным, так как фактические и ожидаемые результаты всех запланированных тестов совпали.



## **ЗАКЛЮЧЕНИЕ**

В ходе работы над поставленным заданием был изучен такой ЯП как Java, получены навыки разработки проекта в команде. Получено понимание работы алгоритмов, а также навыки разработки алгоритмов обработки графов. Были реализованы классы для ввода, вывода, сохранения и демонстрации класса, а также его преобразования. Был изучен алгоритм поиска сильных компонент связностей - Косарайю-Шарира. Программа была успешно протестирована на работоспособность.

## 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. Поиск компонент сильной связности, построение конденсации графа // e-maxx. URL: [https://e-maxx.ru/algo/strong\\_connected\\_components](https://e-maxx.ru/algo/strong_connected_components) (дата обращения: 08.07.2021).