**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по учебной практике**

Тема: Сильно связные компоненты орграфа

|  |  |  |
| --- | --- | --- |
| Студент гр. 1384 |  | Степаненко Д.В. |
| Студент гр. 1381 |  | Возмитель В.Е. |
| Студент гр. 1381 |  | Тарасов К.O. |
| Руководитель |  | Фирсов М.А. |

Санкт-Петербург

2023г.

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент Степаненко Д.В. группы 1384 | | |
| Студент Возмитель В.Е. группы 1381 | | |
| Студент Тарасов К.O. группы 1381  Тема практики: сильно связные компоненты орграфа | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на ЯП Kotlin с графическим интерфейсом.  Алгоритм: Косарайю и Шарира. | | |
| Сроки прохождения практики: 30.06.2020 – 13.07.2023 | | |
| Дата сдачи отчета: 10.07.2023 | | |
| Дата защиты отчета: 10.07.2023 | | |
|  | | |
| Студент |  | Степаненко Д.В. |
| Студент |  | Возмитель В.Е. |
| Студент |  | Тарасов К.O. |
| Руководитель |  | Фирсов М.А. |

**Аннотация**

Целью проекта является разработка итеративного визуализатора алгоритма Косарайю и Шарира с графическим интерфейсом на ЯП Kotlin. Выполнение работы состоит из пяти этапов: создание спецификации и плана разработки; написания кода, реализующего алгоритм; визуализации алгоритма; тестирования созданного приложения; написания отчета. Разработка осуществляется в команде, каждый участник которой выполняет свою роль. Итогом работы приложения считается поэтапное выполнение алгоритма для нахождения компонент сильной связности в направленном графе, т.е. построение графа Герца. Входные данные могут задаваться тремя путями: граф генерируется самой программой, считывается из файла, либо создается самим пользователем с использованием программных инструментов. Проект предлагает разработку программы на ЯП Kotlin 231-1.8.21-IJ9161.38 c GUI, реализованного с помощью JavaFX. Для сборки проекта используется Maven, программа покрывается тестами JUnit 5.

**Summary**

The aim of the project is to develop an interactive visualizer of the Kosaraju and Sharir algorithm with a graphical interface on the Kotlin YAP. The work consists of five stages: creating a specification and a development plan; writing code implementing the algorithm; visualizing the algorithm; testing the created application; writing a report. Development is carried out in a team, each member of which performs its own role. The result of the application is considered to be the step-by-step execution of the algorithm for finding the components of strong connectivity in a directed graph, i.e. the construction of a Hertz graph. Input data can be set in three ways: the graph is generated by the program itself, read from a file, or created by the user himself using software tools. The project offers the development of a program on Kotlin 231-1.8.21-IJ9161.38 with a GUI implemented using JavaFX. Maven is used to build the project; the program is covered by JUnit 5 tests.

# Оглавление

[Оглавление 5](#_Toc139878126)

[Введение 7](#_Toc139878127)

[1. Требования к программе 8](#_Toc139878128)

[1.1 Требования к вводу исходных данных 8](#_Toc139878129)

[1.2 Требования к визуализации 8](#_Toc139878130)

[2. План разработки и распределение ролей в бригаде 11](#_Toc139878131)

[2.1. План разработки 11](#_Toc139878132)

[2.2. Распределение ролей в бригаде 11](#_Toc139878133)

[3.1. Структуры данных 12](#_Toc139878134)

[3.2. Основные методы 13](#_Toc139878135)

[4. Тестирование 16](#_Toc139878136)

[4.1. Тестирование методов класса Algorithm 16](#_Toc139878137)

[4.1.1 Тестирование метода start() 16](#_Toc139878138)

[4.1.2. Тестирование метода dfs 18](#_Toc139878139)

[4.2. Тестирование случайной генерации графа 21](#_Toc139878140)

[4.3 Тестирование класса OrientedGraph 23](#_Toc139878141)

[4.3.1 Тестирование метода fillGraph 23](#_Toc139878142)

[4.3.2 Тестирование метода clearGraph 25](#_Toc139878143)

[4.4 Тестирование интерфейса 25](#_Toc139878144)

[Заключение 36](#_Toc139878145)

[Список использованных источников 37](#_Toc139878146)

[Приложение А 38](#_Toc139878147)

# Введение

Целью проекта является изучение возможностей языка Kotlin, GUI с использованием JavaFX, а также итеративная разработка приложения с графическим интерфейсом на Kotlin, предоставляющего пользователю инструменты для взаимодействия с ним. Приложение должно поэтапно визуализировать алгоритм Косарайю и Шарира, который находит компоненты сильной связности в ориентированном графе.

# 1. Требования к программе

## 1.1 Требования к вводу исходных данных

Входные данные могут быть заданы тремя способами:

Загружены из файла формата json. Граф хранится в виде: {"<номер исходящей вершины>":[<номер входящей вершины>,...],...,"<номер вершины>":[x,y],...,"NodeNumber":<количество вершин>}. После загрузки воссоздается программой графическое представление графа.

Сгенерированы самой программой. Общие данные о графе задаются пользователем в отдельном окне (количество вершин и ребер в графе через пробел), остальное генерируется случайным образом и визуализируются программой. Полученный граф должен быть ориентированным.

## 1.2 Требования к визуализации

Графический интерфейс должен содержать панель управления и холст, на котором будет представлена работа алгоритма.

На панели управления расположены несколько кнопок:

1. Кнопка “Save” – сохраняет построенный граф в файл json.
2. Кнопка “Load” - загружает граф из файла и строит его на холсте.
3. Кнопка “Generate” – очищает холст, запускает генерацию графа и добавляет граф на холст.
4. Кнопка “Algorithm” – показывает результат выполгения алгоритма для нахождения компонент сильной связности графа, выделяя их отличающимися цветами.
5. Кнопка “Delete” – включает режим удаления вершин и ребер с холста при нажатии на них.
6. Кнопка “Move” – включает режим перемещения вершин графа на хосте, перемещая их при выборе и нажатии на свободное место холста.
7. Кнопка “Clear” – очищает холст.
8. Кнопка "Step by step" - запускает пошаговую реацию алгоритма с указанием скорости выполнения в сплывающем окне.

Также для генерации графа должны быть создано окно, в котором пользователь может указать количество узлов и ребер через пробел, опираясь на которые программа сгенерирует граф.

После каждой итерации алгоритма должен отображаться текст, указывающий на изменения в графе ("DFS первоначальный: рассмотрена вершина №x", "Определение компонент СС: раскрашена вершина №y"). О выполнении основных этапов алгоритма сигнализируется сообщением:

1) "Завершен DFS по графу"

2) "Завершено инвертирование графа"

3) "Завершен DFS по инвертированному графу"

4) "Завершено раскрашивание графа по компонентам"

Просмотренные вершины и пройденные ребра в первом этапе алгоритма будут окрашиваться в серый цвет. Далее граф инвертируется и ребра меняют свое направление. В конечном шаге цвет вершины будет завесить от компонент сильной связности, ребра в графе Герца должны выделяться жирным шрифтом. Скорость визуализации итераций алгоритма должна быть задана автоматически (т.е. будет фиксированной).

Диаграмма сценариев представлена на рисунке 1, макет интерфейса – на рисунке 2.

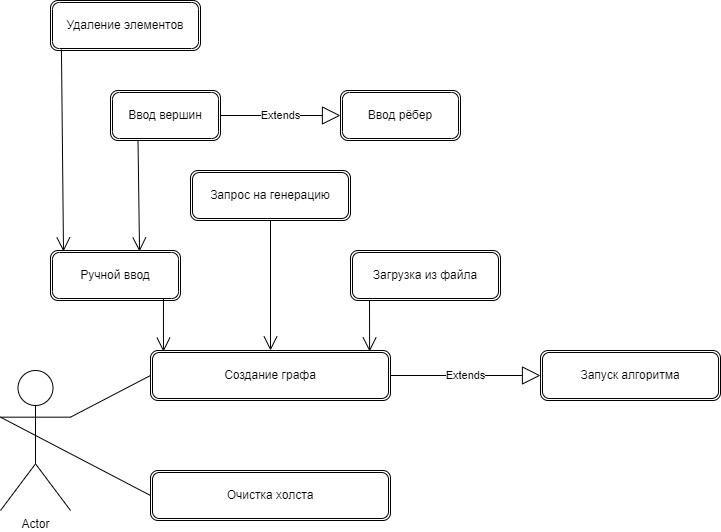


Рис. 1 – диаграмма сценариев использования приложения

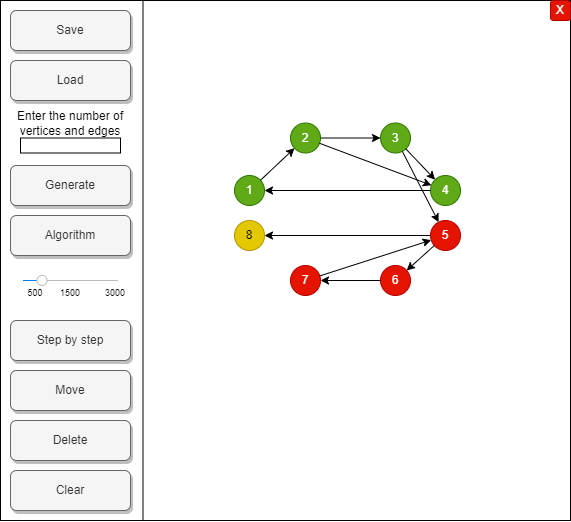


Рис. 2 – макет графического интерфейса приложения

# 2. **План разработки и распределение ролей в бригаде**

## 2.1. План разработки

03.07 – Начало работы с отчетом, написание раздела спецификаций, описание ролей участников команды. Создание диаграммы состояний для описания процесса пошагового исполнения алгоритма. Согласование спецификации. Создание Maven-проекта.

05.07. – Создание прототипа приложения: реализация структур данных и алгоритма, тестов для структур данных и алгоритма, набросков интерфейса. Создание диаграмм классов и описание сущностей. Описание тестовых случаев.

07.07 – Создание первой версии приложения: реализация генерации данных, выполнение и отображение результата работы алгоритма. Описание интерфейса взаимодействия с выполнением алгоритма.

10.07 – Создание второй версии приложения: реализация корректной работы кнопки, отвечающей за пошаговое выполнение алгоритма, и кнопки, отвечающей за удаление объектов с холста; реализация структур данных, отвечающих за пошаговое выполнение алгоритма; реализация тестов для этих структур. Оптимизация кода.

12.07 – Сборка проекта в jar-архив, предоставление итогового отчета.

## 2.2. Распределение ролей в бригаде

Степаненко Денис – покрытие программ тестами, связь интерфейса и внутренних структур данных.

Возмитель Влас – визуализация, создание интерфейса программы.

Тарасов Константин - реализация алгоритма и структур данных.3. Особенности реализации

## 3.1. Структуры данных

1) *class Node(val name: Int, var x: Double, var y: Double, var r: Double) -* класс, имитирующий вершину графа. Структура класса хранит необходимые для описания и отрисовки вершины поля и методы. Каждый объект данного класса хранит смежные вершины.

2) *interface Graph* - интерфейс, описывающее стандартные методы и поведение графов.

3) *class OrientedGraph: Graph -* класс, наследуемый от интерфейса *Graph,* содержит реализацию методов, описанных в интейрфесе, хранит набор веришн графа.

4) *class Drawablegraph(var FrontPane: AnchorPane, n: Int, m: Int, var graph: OrientedGraph) -* класс, содержащий методы для визуализации графа.

5) *class GraphGenerator -* класс, случайным образом генерирующий ориентированный граф. Класс имеет основной метод *generateGraph,* который получает на вход количество вершин и ребер, возвращая ориентированный граф.

6) *interface Algorithm -* интерфейс, описывающий поведение и стандарные методы алгоритмов для графа.

7) *class Kesarajo: Algorithm -* класс, наследуемый от интерфейса *Algotithm.* В данном классе реализованы необходимые методы для решения задачи поиска сильных компонент связности.

8) *class Saver -* класс, созданный для реализации сохранения текущего графа, в целях получения для пользователя возможности загрузить сохранённый граф из файла в любой момент.

9) *class Loader -* класс, реализующий загрузку сохранённого графа из файла в любой момент.

10) *class MainController -* класс, являющийся главным инструментом управления для объектов, расположенных на отображаемом окне приложения. Здесь имеются поля, представляющие объекты интерфейса, а также объект graph класса *DrawableGraph,* необходимый для отрисовки и изменения графа.

UML-диаграмма представлена на рис. 3.

## 3.2. Основные методы

1) *class Node(val name: Int, var x: Double, var y: Double, var r: Double) -* класс, имитирующий вершину графа. Структура класса хранит необходимые для описания и отрисовки вершины поля и методы. Каждый объект данного класса хранит смежные вершины.

2) *interface Graph* - интерфейс, описывающее стандартные методы и поведение графов.

3) *class OrientedGraph: Graph -* класс, наследуемый от интерфейса *Graph,* содержит реализацию методов, описанных в интейрфесе, хранит набор веришн графа.

4) *class Drawablegraph(var FrontPane: AnchorPane, n: Int, m: Int, var graph: OrientedGraph) -* класс, содержащий методы для визуализации графа.

5) *class GraphGenerator -* класс, случайным образом генерирующий ориентированный граф. Класс имеет основной метод *generateGraph,* который получает на вход количество вершин и ребер, возвращая ориентированный граф.

6) *interface Algorithm -* интерфейс, описывающий поведение и стандарные методы алгоритмов для графа.

7) *class Kesarajo: Algorithm -* класс, наследуемый от интерфейса *Algotithm.* В данном классе реализованы необходимые методы для решения задачи поиска сильных компонент связности.

8) *class Saver -* класс, созданный для реализации сохранения текущего графа, в целях получения для пользователя возможности загрузить сохранённый граф из файла в любой момент.

9) *class Loader -* класс, реализующий загрузку сохранённого графа из файла в любой момент.

10) *class MainController -* класс, являющийся главным инструментом управления для объектов, расположенных на отображаемом окне приложения. Здесь имеются поля, представляющие объекты интерфейса, а также объект graph класса *DrawableGraph,* необходимый для отрисовки и изменения графа.

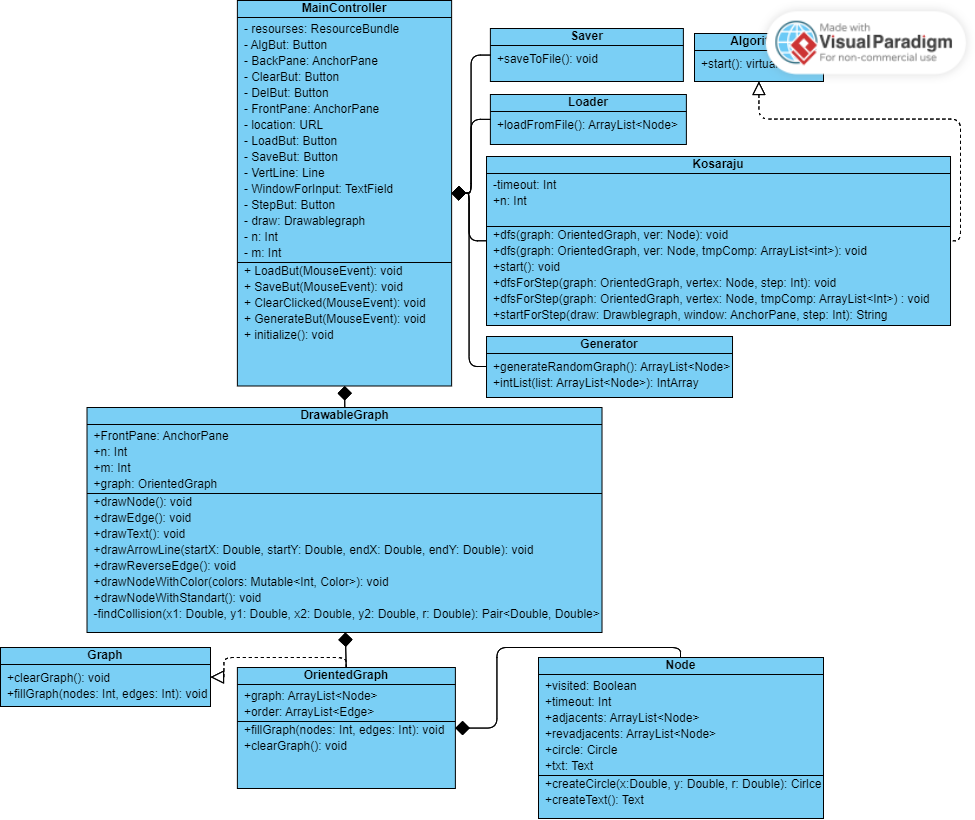


Рис. 3 – UML-диаграмма.

# 4. Тестирование

## 4.1. Тестирование методов класса Algorithm

### 4.1.1 Тестирование метода start()

Для тестирования реализации алгоритма был использован фреймворк JUnut5. Тесты были разделены на следующие группы:

1. Граничные случаи
   1. Граф, состоящий из одной вершины
   2. 2-полный граф
   3. 3-полный граф
   4. Несвязный граф из 3 вершин (3 вершины, 0 ребер)
2. Позитивные тесты
   1. Обычный связный граф (8 вершин и 13 ребер)
   2. 1-связный граф c одной компонентой (3 вершины, 3 ребра)
   3. 2-связный граф с одной компонентой (3 вершины, 4 ребра)
   4. 2-связный граф с двумя компонентами (3 вершины, 2 ребра)
   5. 2-связный граф с двумя компонентами (4 вершины, 4 ребра)
   6. 1-связный граф с тремя компонентами (3 вершины, 2 ребра)
   7. 1-связный граф с тремя компонентами (3 вершины, 1 ребро)
3. Негативные тесты
   1. Граф с нулевым количеством вершин
   2. Граф, у которого количество ребер превышает максимально возможное

Таблица 1 - Примеры тестовых случаев для метода start.

| Название | Входной граф (<номер вершины>: <инцидентные ей вершины>) | Проверки (<номер компоненты>:[<номера вершин>]) | Комментарий |
| --- | --- | --- | --- |
| Kosaraju: Graph with one vertex | 1: [] | 1:[1] | Работает верно! |
| Kosaraju: 2-full graph | 1:[2], 2:[1] | 1: [1, 2] | Работает верно! |
| Kosaraju: 3-full graph | 1: [2,3], 2: [1,3], 3: [1,2] | 1: [1, 2, 3] | Работает верно! |
| Kosaraju: Disconnected graph with three components | 1: [], 2: [], 3: [] | 1: [3] 2: [2] 3: [1] | Работает верно! |
| Kosaraju: Ordinary Graph | 1: [2], 2: [3,5,6], 3: [4,7], 4: [3,8], 5: [1,6], 6: [7], 7: [6,8], 8: [] | 1: [1, 5, 2] 2: [3, 4] 3: [7, 6] 4: [8] | Работает верно! |
| Kosaraju: Graph with one component (3 vertexes, 3 ages) | 1: [2], 2: [3], 3: [2] | 1: [1, 2, 3] | Работает верно! |
| Kosaraju: Graph with one component (3 vertexes, 4 ages) | 1: [2], 2: [1,3], 3: [2] | 1: [1, 2, 3] | Работает верно! |
| Kosaraju: Graph with two components (3 vertexes, 2 ages) | 1: [2], 2: [1], 3: [] | 1: [3] 2: [1, 2] | Работает верно! |
| Kosaraju: Graph with three components (3 vertexes, 2 ages) | 1: [2], 2: [3], 3: [] | 1: [1] 2: [2] 3: [3] | Работает верно! |
| Kosaraju: Graph with three components (3 vertexes, 1 age) | 1: [2], 2: [], 3: [] | 1: [3] 2: [1] 3: [2] | Работает верно! |
| Kosaraju: Graph with 0 vertex | - | - | Функция не считывает ни одной вершины. Возвращаемый ответ пуст. Работает верно! |
| Kosaraju: Graph with n vertexes & n(n-1)+1 ages | 1: [1,2,3], 2: [1,3], 3: [1,2] | 1: [1, 2, 3] | Функция считывания интерпретирует граф как полный. Работает верно! |

### 4.1.2. Тестирование метода dfs

1. Граничные случаи
   1. Граф, состоящий из одной вершины
   2. 2-полный граф
   3. 3-полный граф
   4. Несвязный граф из 3 вершин (3 вершины, 0 ребер)
2. Позитивные тесты
   1. Обычный связный граф (8 вершин и 13 ребер)
   2. 1-связный граф c одной компонентой (3 вершины, 3 ребра)
   3. 2-связный граф с одной компонентой (3 вершины, 4 ребра)
   4. 2-связный граф с двумя компонентами (3 вершины, 2 ребра)
   5. 1-связный граф с тремя компонентами (3 вершины, 2 ребра)
   6. 1-связный граф с тремя компонентами (3 вершины, 1 ребро)
3. Негативные тесты
   1. Граф с нулевым количеством вершин
   2. Граф, у которого количество ребер превышает максимально возможное

Таблица 2 - Примеры тестовых случаев для метода dfs.

| Название | Входной граф (<номер вершины>: <инцидентные ей вершины>) | Проверки (<время выхода вершин графа по индексам>]) | Комментарий |
| --- | --- | --- | --- |
| DFS: Graph with one vertex | 1: [] | 2 | Работает верно! |
| DFS: 2-full graph | 1:[2], 2:[1] | 4 3 | Работает верно! |
| DFS: 3-full graph | 1: [2,3], 2: [1,3], 3: [1,2] | 6 5 4 | Работает верно! |
| DFS: Disconnected graph with three components | 1: [], 2: [], 3: [] | 2 4 6 | Работает верно! |
| DFS: Ordinary Graph | 1: [2], 2: [3,5,6], 3: [4,7], 4: [3,8], 5: [1,6], 6: [7], 7: [6,8], 8: [] | 16 15 12 7 14 10 11 6 | Работает верно! |
| DFS: Graph with one component (3 vertexes, 3 ages) | 1: [2], 2: [3], 3: [2] | 6 5 4 | Работает верно! |
| DFS: Graph with one component (3 vertexes, 4 ages) | 1: [2], 2: [1,3], 3: [2] | 6 5 4 | Работает верно! |
| DFS: Graph with two components (3 vertexes, 2 ages) | 1: [2], 2: [1], 3: [] | 4 3 6 | Работает верно! |
| DFS: Graph with three components (3 vertexes, 2 ages) | 1: [2], 2: [3], 3: [] | 6 5 4 | Работает верно! |
| DFS: Graph with three components (3 vertexes, 1 age) | 1: [2], 2: [], 3: [] | 4 3 6 | Работает верно! |
| DFS: Graph with 0 vertex | - | 0 | Функция не считывает ни одной вершины. Время возврата равно нулю. Работает верно! |
| DFS: Graph with n vertexes & n(n-1)+1 ages | 1: [1,2,3], 2: [1,3], 3: [1,2] | 6 5 4 | Функция считывания интерпретирует граф как полный. Работает верно! |

## 4.2. Тестирование случайной генерации графа

1. Граничные случаи
   1. Граф, состоящий из одной вершины
   2. Несвязный граф
2. Позитивные тесты (граф с положительным количеством вершин N и ребер не более чем N(N-1))
   1. 2-полный граф
   2. Граф с 5 вершинами и 12 ребрами
3. Негативные тесты
   1. Граф, с количеством вершин ≤ 0
   2. Граф, с количеством ребер < 0
   3. Граф, у которого количество ребер > максимально возможного

Таблица 3 - Примеры тестовых случаев для метода generateGraph.

| Название | Входные данные (<количество вершин> <количество ребер>) | Проверки (итоговое <количество вершин> <количество ребер>) | Комментарий |
| --- | --- | --- | --- |
| Graph with one vertex | 1 0 | 1 0 | Работает верно! |
| Discharged graph with five vertexes | 5 0 | 5 0 | Работает верно! |
| 2-full graph | 2 2 | 2 2 | Работает верно! |
| Check node & edges amount | 5 12 | 5 12 | Работает верно! |
| Graph with 0 vertex | 0 0 | 0 0 | Работает верно! |
| Graph with a negative number of vertexes | -10 0 | 0 0 | Так как были введены некорректные данные, алгоритм не создал граф. Работает верно! |
| Graph with a negative number of vertexes & 3 ages | -10 3 | 0 0 | Так как были введены некорректные данные, алгоритм не добавил вершин и ребер. Работает верно! |
| Graph with a negative number of ages | 3 -10 | 3 0 | Так как были введены некорректные данные, алгоритм не добавил ребра. Работает верно! |
| Graph with a negative number of vertexes and ages | -10 -10 | 0 0 | Так как были введены некорректные данные, алгоритм не создал граф. Работает верно! |
| Graph with 1 vertex and 12 ages | 1 12 | 1 0 | Так как были введены некорректные данные, алгоритм откинул превышающее количество ребер. Работает верно! |
| Graph with 2 vertexes and 12 ages | 2 12 | 2 2 | Так как были введены некорректные данные, алгоритм откинул превышающее количество ребер. Работает верно! |

## 4.3 Тестирование класса OrientedGraph

### 4.3.1 Тестирование метода fillGraph

1. Заполнить граф, состоящий из одной вершины
2. Заполнить несвязный граф
3. Заполнить 2-полный граф
4. Заполнить граф с 5 вершинами и 12 ребрами
5. Заполнить граф, с количеством вершин ≤ 0
6. Заполнить граф, с количеством ребер < 0
7. Заполнить граф, у которого количество ребер > максимально возможного

Таблица 4 - Примеры тестовых случаев для метода fillGraph.

| Название | Входные данные (<количество вершин> <количество ребер>) | Проверки (итоговое <количество вершин> <количество ребер>) | Комментарий |
| --- | --- | --- | --- |
| Fill Graph with one vertex | 1 0 | 1 0 | Работает верно! |
| Fill disconnected graph (3 vertexes) | 3 0 | 3 0 | Работает верно! |
| Fill 2-full graph | 2 2 | 2 2 | Работает верно! |
| Fill ordinary graph (5 vertexes, 12 ages) | 5 12 | 5 12 | Работает верно! |
| Fill graph with a 0 number of vertexes (0 vertexes, 0 ages) | 0 0 | 0 0 | Работает верно! |
| Fill graph with a negative number of vertexes (-5 vertexes, 0 ages) | -5 0 | 0 0 | Так как были введены некорректные данные, алгоритм не создал граф. Работает верно! |
| Fill graph with a negative number of ages (5 vertexes, -10 ages) | 5 -10 | 5 0 | Так как были введены некорректные данные, алгоритм не добавил ребер. Работает верно! |
| Fill graph with a negative number of vertexes and ages (-5 vertexes, -10 ages) | -5 -10 | 0 0 | Так как были введены некорректные данные, алгоритм не добавил вершин и ребер. Работает верно! |
| Fill graph with a maximum+1 number of ages (5 vertexes, 21 ages) | 5 21 | 5 20 | Так как были введены некорректные данные, алгоритм откинул превышающее количество ребер. Работает верно! |

### 4.3.2 Тестирование метода clearGraph

1. Очистить пустой граф
2. Очистить полный граф
3. Очистить граф из одной вершины
4. Очистить несвязный граф

Таблица 5 - Примеры тестовых случаев для метода clearGraph.

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные данные (<количество вершин> <количество ребер>) | Проверки (итоговое <количество вершин> <количество ребер>) | Комментарий |
| Clear empty Graph | 0 0 | 0 0 | Работает верно! |
| Clear 2-full graph | 2 2 | 0 0 | Работает верно! |
| Clear graph with one vertex | 1 0 | 0 0 | Работает верно! |
| Clear disconnected Graph | 5 0 | 0 0 | Работает верно! |

## 4**.4 Тестирование интерфейса**

1. Тестирование кнопки “Save”
   1. Сохранить сгенерированный граф
   2. Сохранить граф с удаленным ребром
   3. Сохранить граф с удаленной вершиной
   4. Сохранить полностью удаленный граф
   5. Сохранить граф в момент визуализации
   6. Сохранить граф в режиме перемещения
   7. Сохранить граф в режиме удаления
2. Тестирование кнопки “Load”
   1. Загрузить пустой граф
   2. Загрузить граф с одной вершиной
   3. Загрузить 2-полный граф
   4. Загрузить 3-полный граф
   5. Загрузить граф с 10 вершинами и 10 ребрами
   6. Загрузить граф в момент визуализации
   7. Повторное нажатие кнопки
   8. Нажать на кнопку в режиме перемещения
   9. Нажать на кнопку в режиме удаления
3. Тестирование кнопки “Generate”
   1. Ввод графа с 10 вершинами и 10 ребрами (через пробел)
   2. Ввод графа с 7 вершинами и 10 ребрами (через пробел)
   3. Ввод графа с 1 вершиной и 0 ребер
   4. Ввод графа с 0 вершинами
   5. Ввод 2-полного графа
   6. Ввод неверных данных (количество ребер больше максимального) (3 вершины 10 ребер)
   7. Ввод неверных данных (-10 вершин -10 ребер)
   8. Ввод неверных данных (-10 вершин 5 ребер)
   9. Ввод неверных данных (5 вершин -10 ребер)
   10. Ввод неверных данных (0 вершин -10 ребер)
   11. Ввод неверных данных (-10 вершин 0 ребер)
   12. Ввод данных через запятую
   13. Ввод данных через двойной пробел
   14. Генерировать без ввода данных
   15. Генерировать граф в момент визуализации
   16. Использовать при вводе символы
   17. При вводе перед цифрами указать несколько 0
   18. Ввод 3 и более значений вместо 2
   19. Нажать на кнопку в режиме перемещения
   20. Нажать на кнопку в режиме удаления
4. Тестирование кнопки “Algorithm”
5. Запуск алгоритма для графа с 10 вершинами и 10 ребрами
6. Запуск алгоритма для графа из 1 вершины
7. Запуск алгоритма для пустого графа
8. Запустить алгоритм в момент визуализации
9. Повторное нажатие на кнопку
10. Нажать на кнопку в режиме перемещения
11. Нажать на кнопку в режиме удаления
12. Тестирование кнопки “Step by step”
    1. Нажатие кнопки без ввода скорости
    2. Нажатие кнопки со скоростью 3000
    3. Нажатие кнопки с вводом скорости в момент выполнения визуализации
    4. Использование кнопки после удаления ребра
    5. Использование кнопки после удаления вершины
    6. Использование кнопки для пустого графа
    7. Использования кнопки для графа с 1 вершиной
    8. Нажать на кнопку в режиме перемещения
    9. Нажать на кнопку в режиме удаления
13. Тестирование кнопки “Delete”
    1. Удаление вершины с исходящими ребрами в 3-полном графе
    2. Удаление всех ребер в 3-полном графе
    3. Удаление вершины в несвязном графе
    4. Удаление вершины в графе из одной вершины
    5. Удалить вершину в момент визуализации
    6. Удалить ребро в момент визуализации
    7. Удалить вершину уже у раскрашенного графа
    8. Удалить ребро уже у раскрашенного графа
    9. Повторное нажатие на кнопку
    10. Нажать на кнопку в режиме перемещения
    11. Нажатие на свободные координаты холста в режиме удаления.
    12. Сохранить и загрузить граф с удаленным ребром и вершиной
14. Тестирование кнопки “Clear”
    1. Очистить холст с 3-полным графом
    2. Очистить холст с пустым графом
    3. Очистить пустой холст
    4. Очистить холст в момент визуализации
    5. Повторное нажатие кнопки
    6. Очистить холст с раскрашенным графом
    7. Очистить холст после удаления вершины
    8. Очистить холст после удаления ребра
    9. Очистить холст в режиме удаления
    10. Очистить холст в режиме перемещения
15. Тестирование кнопки закрытия, сворачивания и оконного режима программы
    1. Закрыть программу
    2. Свернуть программу
    3. Развернуть программу
    4. Открыть программу в полное окно
    5. Уменьшить окно программы
16. Тестирование кнопки “Move”
    1. Перемещение вершины в 3-полном графе
    2. Перемежение всех вершин в 3-полном графе
    3. Перемещение вершины в несвязном графе
    4. Перемещение вершины в графе из одной вершины
    5. Перемещение вершину в момент визуализации
    6. Перемещение вершины уже у раскрашенного графа
    7. Повторное нажатие на кнопку
    8. Клик на холст в режиме перемещения
    9. Нажать на кнопку в режиме удаления
    10. Сохранить и загрузить граф с перемещенной вершиной
    11. После выбора вершины нажимать на занятые координаты холста

Таблица 6 - Примеры тестов интерфейса.

|  |  |
| --- | --- |
| Номер теста | Комментарий |
| 1.а | Граф сохраняется в виде файла формата .json. Работает верно! |
| 1.b | Сохраняется граф с удаленным ребром. Работает верно! |
| 1.c | Сохраняется граф с удаленной вершиной. Работает верно! |
| 1.d | Сохраняется пустой граф. Работает верно! |
| 1.e | Визуализация приостанавливается, граф сохраняется. При повторном запуске алгоритм начинается сначала. Цвета вершин меняются. Работает верно! |
| 1.f | Режим перемещения выключается, граф сохраняется. Работает верно! |
| 1.g | Режим удаления выключается, граф сохраняется. Работает верно! |
| 2.a | Загружается пустой граф. Работает верно! |
| 2.b | Загружается граф с одной вершиной. Работает верно! |
| 2.c | Загружается 2-полный граф. Работает верно! |
| 2.d | Загружается 3-полный граф. Работает верно! |
| 2.e | Загружается указанный в фале граф. Работает верно! |
| 2.f | Визуализация останавливается, загружается новый граф. Работает верно! |
| 2.g | При повторном нажатии кнопки ничего не происходит. Работает верно! |
| 2.h | Режим перемещения выключается, загружается сохраненный граф. Работает верно! |
| 2.i | Режим удаления выключается, загружается сохраненный граф. Работает верно! |
| 3.a | Создается граф с верным количеством вершин и ребер. Работает верно! |
| 3.b | Создается граф с верным количеством вершин и ребер. Работает верно! |
| 3.c | Создается граф с 1 вершиной. Работает верно! |
| 3.d | Создается пустой граф. Всплывает соответствующее уведомление. Работает верно! |
| 3.e | Создается 2-полный граф. Работает верно! |
| 3.f | Создается 3-полный граф, количество ребер меняется на максимально возможное. Всплывает соответствующее уведомление. Работает верно! |
| 3.g | При вводе отрицательных ребер граф не создается, выводится сообщение о неверном вводе данных. Работает верно! |
| 3.h | При вводе отрицательных вершин и положительных ребер граф не создается, выводится сообщение о неверном вводе данных. Работает верно! |
| 3.i | При вводе положительного числа вершин и отрицательного числа ребер граф не создается, выводится сообщение о неверном вводе данных. Работает верно! |
| 3.j | При вводе 0 вершин и отрицательного числа ребер граф не создается, выводится сообщение о неверном вводе данных. Работает верно! |
| 3.k | При вводе отрицательного числа вершин и 0 ребер граф не создается, выводится сообщение о неверном вводе данных. Работает верно! |
| 3.l | При вводе данных через запятую граф не создается, выводится сообщение о неверном вводе данных. Работает верно! |
| 3.m | При вводе данных через двойной пробел запятую граф не создается, выводится сообщение о неверном вводе данных. Работает верно! |
| 3.n | При попытке генерации графа без данных: 1) после запуска программы создается граф из 5 вершин и 7 ребер; 2) после корректного использования кнопки строит граф с предыдущими параметрами. |
| 3.o | Визуализации прекращается, строится новый граф. Работает верно! |
| 3.p | При вводе символов в строку граф не создается, выводится сообщение о неверном вводе данных. Работает верно! |
| 3.q | При попытке ввода нулей перед числами всплывает сообщение о некорректном вводе. Работает верно! |
| 3.r | При вводе 3 значений в окно всплывает сообщение о некорректном вводе. Работает верно! |
| 3.s | Режим перемещения отключается. Выполняется генерация графа. Работает верно! |
| 3.t | Режим удаления отключается. Выполняется генерация графа. Работает верно! |
| 4.a | Раскраска графа выполняется верно! |
| 4.b | Вершина раскрашивается в цвет. Работает верно! |
| 4.c | Ничего не происходит. Работает верно! |
| 4.f | Режим перемещения отключается. Выполняется раскраска графа. Работает верно! |
| 4.g | Режим удаления отключается. Выполняется раскраска графа. Работает верно! |
| 4.d | Визуализация останавливается. На холсте появляется раскраска графа. Работает верно! |
| 4.e | Граф раскрашивается в другие цвета, при этом раскраска остается верной. Работает верно! |
| 5.a | При запуске алгоритма стандартная скорость = 500 мс. Работает верно! |
| 5.b | Скорость выполнения алгоритма замедляется до 3000 мс. Работает верно! |
| 5.c | При повторном нажатии кнопки алгоритм начнется заново, причем с новой установленной скоростью. Работает верно! |
| 5.d | Алгоритм выполняется стандартно. Работает верно! |
| 5.e | Алгоритм выполняется стандартно. Работает верно! |
| 5.f | Ничего не происходит, т.к. граф пустой. Работает верно! |
| 5.g | Алгоритм раскрашивает вершину. Работает верно! |
| 5.h | Режим перемещения отключается. Запускается алгоритм. Работает верно! |
| 5.i | Режим удаления отключается. Запускается алгоритм. Работает верно! |
| 6.a | Удаляется вершина и инцидентные ей ребра. Номер вершины смещается при необходимости. Работает верно! |
| 6.b | Все ребра удаляются последовательно. Одно нажатие – одно ребро. Работает верно! |
| 6.c | Вершины удаляются корректно. |
| 6.d | Вершина удаляется, образуется пустой граф. |
| 6.e | Визуализация приостанавливается. Включается режим удаления. Получается успешно удалить вершину. |
| 6.f | Визуализация приостанавливается. Включается режим удаления. Получается успешно удалить ребро. |
| 6.g | Раскраска графа пропадет, включится режим удаления. Вершина успешно удалится. |
| 6.h | Раскраска графа пропадет, включится режим удаления. Ребро успешно удалится. |
| 6.i | При повторном нажатии на кнопку ничего не происходит, т.к. кнопка зажата, и пользователь находится в режиме удаления. Работает верно! |
| 6.j | Происходит переключение на режим удаления. Работает верно! |
| 6.k | При нажатии на холст в режиме удаления происходит выход из режима. Работает верно! |
| 6.l | Граф с удаленным ребром и вершиной успешно сохраняется и загружается. |
| 7.a | Холст очищается. Работает верно! |
| 7.b | Ничего не происходит. Работает верно! |
| 7.c | Ничего не происходит. Работает верно! |
| 7.d | Выполнение пошаговой визуализации останавливается. Холст очищается. Работает верно! |
| 7.e | Повторное нажатие кнопки ни к чему не приводит. Работает верно! |
| 7.f | Холст очищается. Работает верно! |
| 7.g | Холст очищается. Работает верно! |
| 7.h | Холст очищается. Работает верно! |
| 7.i | Холст очищается. Работает верно! |
| 7.j | Холст очищается. Работает верно! |
| 8.a | При нажатии на крестик приложение закрывается. Работает верно! |
| 8.b | Приложение сворачивается. Работает верно! |
| 8.c | Приложение разворачивается. Работает верно! |
| 8.d | Приложение открывается в полноэкранном режиме. Элементы интерфейса не подстраиваются под него. |
| 8.e | Окно приложения можно уменьшить. Элементы интерфейса не подстраиваются под него. |
| 9.a | Перемещение выполняется после активации режима перемещения после нажатия на кнопку. Далее выбранная кликом вершина перемещается на указанной вторым кликом место. Работает верно! |
| 9.b | Все вершины перемещаются. Работает верно! |
| 9.c | Все вершины перемещаются. Работает верно! |
| 9.d | Вершина перемещается корректно. |
| 9.e | Визуализация продолжается, вершина перемещается. Работает верно! |
| 9.f | Раскраска сохраняется, вершина перемещается. Работает верно! |
| 9.g | Невозможно нажать на кнопку, т.к. она зажата (пользователь находится в режиме перемещения). Работает верно! |
| 9.h | Вершина для перемещения не выбирается. Выход из режима перемещения. Работает верно! |
| 9.i | Режим удаления переключается на режим перемещения. Работает верно! |
| 9.j | Измененный граф сохраняется и загружается верно! |
| 9.k | Происходит переопределение перемещаемой вершины. Работает верно! |

# Заключение

В ходе выполнения проекта было разработано графическое приложение на языке Kotlin с использованием таких фреймворков как: JavaFX, Maven, Junit5, Json и др. Приложение полностью соответствует спецификации: позволяет ввести граф двумя способами (генерацией или загрузкой из файла формата .json), реализует все поставленные графические задачи. Также были написаны тесты для проверки работы алгоритмов и структур. Результат работы алгоритма Косарайю и Шарира представлен раскраской вершин графа.

# Список использованных источников

1. Рыбин С. В. Дискретная математика и информатика: Лань, 2022. 748 с.
2. Poul Klausen Java 14: Development of applications with JavaFX: Software Development: bookboon.com, 2018. 179 c.
3. Руководство пользователя JUnit 5 // junit.org. URL: <https://junit.org/junit5/docs/current/user-guide/> (дата обращения: 04. 07.2023)
4. Документация ЯП Kotlin // kotlinlang.org. URL: <https://kotlinlang.org/docs/home.html> (дата обращения: 03. 07.2023)
5. Документация JavaFX // docs.oracle.com. URL: <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm> (дата обращения: 04. 07.2023)
6. Спецификация XML Schema language // [www.w3.org](http://www.w3.org) URL: <https://www.w3.org/TR/xmlschema-2/#string> (дата обращения: 07. 07.2023)
7. Статьи для изучения ЯП Kotlin // baeldung.com. URL: <https://www.baeldung.com/kotlin/> (дата обращения: 08. 07.2023)
8. Леонид Старцев Kotlin Serialization Guide // github.com. URL: <https://github.com/Kotlin/kotlinx.serialization/blob/master/docs/serialization-guide.md> (дата обращения: 05.07.2023)
9. Курс “Введение в Kotlin” // stepik.org. URL: <https://stepik.org/course/5448/info> (дата обращения: 01. 07.2023)

# Приложение А

#### Код приложения

**Файл Algorithm.kt**

package com.example.practice

import javafx.event.EventHandler

import javafx.scene.control.Label

import javafx.scene.layout.AnchorPane

import javafx.scene.paint.Color

import javafx.scene.text.Font

import java.util.\*

import kotlinx.coroutines.\*

import java.lang.Thread.sleep

import java.time.Duration

import kotlin.coroutines.\*

import kotlin.time.DurationUnit

import kotlin.time.toDuration

import kotlinx.coroutines.javafx.JavaFx as Main

interface Algorithm{

fun start(): String

}

class Kosaraju(var downlabel: Label = Label(""), var label: Label = Label(""), var graph: OrientedGraph = OrientedGraph()): Algorithm {

private var timeout: Int = 0

var n = graph.graph.size

private var job: Job? = null

fun dfs(graph: OrientedGraph, vertex: Node) {

timeout += 1

vertex.visited = true

for (node in vertex.adjacents) {

if (!node.visited) {

dfs(graph, node)

}

}

timeout += 1

vertex.timeout = timeout

graph.order.add(vertex)

}

fun dfs(graph: OrientedGraph, vertex: Node, tmpComp: ArrayList<Int>) {

vertex.visited = true

tmpComp.add(vertex.name - 1)

for (node in vertex.revadjacents) {

if (!node.visited) {

dfs(graph, node, tmpComp)

}

}

}

override fun start(): String {

var result = ""

for (vertex in 0 until n) {

if (!graph.graph[vertex].visited) {

dfs(graph, graph.graph[vertex])

}

}

if (graph.order.size > 1)

graph.order = graph.order.reversed() as ArrayList<Node>

else if (graph.order.size == 1)

// return "1: [1] "

else

return ""

graph.graph.forEach { it.visited = false }

var i = 0

for (vertex in graph.order) {

if (!vertex.visited) {

val tmpComp = ArrayList<Int>()

dfs(graph, vertex, tmpComp)

val r = (0..255).random()

val g = (0..255).random()

val b = (0..255).random()

for (an in 0 until tmpComp.size) {

tmpComp[an] = tmpComp[an] + 1

graph.graph[tmpComp[an] - 1].circle.fill = Color.rgb(r, g, b)

}

i++

result = "$result$i: $tmpComp "

}

}

println(result)

graph.graph.forEach { it.visited = false }

graph.order.clear()

label.text = " - The algorithm is completed. The components are built.\n"

label.text += " - Amount: $i\n"

find\_bridges()

return result

}

suspend fun dfsForStep(graph: OrientedGraph, vertex: Node, step: Int) {

vertex.visited = true

for (node in vertex.adjacents) {

if (!node.visited) {

node.circle.fill = Color.MEDIUMSEAGREEN

downlabel.text = " - Vertex ${node.name - 1} is considered"

delay(step.toDuration(DurationUnit.MILLISECONDS))

dfsForStep(graph, node, step)

}

}

vertex.timeout = timeout

graph.order.add(vertex)

timeout += 1

}

suspend fun dfsForStep(graph: OrientedGraph, vertex: Node, tmpComp: ArrayList<Int>) {

vertex.visited = true

tmpComp.add(vertex.name - 1)

for (node in vertex.revadjacents) {

if (!node.visited) {

dfsForStep(graph, node, tmpComp)

}

}

}

suspend fun startForStep(draw: DrawableGraph, window: AnchorPane, step: Int): String{

val result = ""

delay(step.toDuration(DurationUnit.MILLISECONDS))

label.text += " - The first DFS is starting...\n"

for (vertex in 0 until n) {

if (!graph.graph[vertex].visited) {

downlabel.text = " - Vertex ${graph.graph[vertex].name - 1} is considered"

graph.graph[vertex].circle.fill = Color.MEDIUMSEAGREEN

delay(step.toDuration(DurationUnit.MILLISECONDS))

dfsForStep(graph, graph.graph[vertex], step)

}

}

label.text += " - The first DFS has completed its work.\n"

if (graph.order.size > 1)

graph.order = graph.order.reversed() as ArrayList<Node>

else if (graph.order.size == 1)

// return "1: [] "

else

return ""

graph.graph.forEach { it.visited = false }

var i = 0

delay(step.toDuration(DurationUnit.MILLISECONDS)/2) // для корректной работы логгера

window.children.clear()

draw.drawNodeWithStandart()

draw.drawText()

draw.drawReverseEdge()

delay(step.toDuration(DurationUnit.MILLISECONDS)/2)

label.text += " - The edges have changed direction in the graph.\n"

delay(step.toDuration(DurationUnit.MILLISECONDS))

label.text += " - The second DFS is starting...\n"

delay(step.toDuration(DurationUnit.MILLISECONDS))

val colors = mutableMapOf<Int, Color>()

for (vertex in graph.order) {

if (!vertex.visited) {

val tmpComp = ArrayList<Int>()

dfsForStep(graph, vertex, tmpComp)

val r = (0..255).random()

val g = (0..255).random()

val b = (0..255).random()

downlabel.text = "Component definition $i: "

delay(step.toDuration(DurationUnit.MILLISECONDS))

for (an in 0 until tmpComp.size) {

tmpComp[an] = tmpComp[an] + 1

graph.graph[tmpComp[an] - 1].circle.fill = Color.rgb(r, g, b)

downlabel.text = "Vertex ${graph.graph[tmpComp[an] - 1].name - 1} is colored."

colors.put(tmpComp[an] - 1, Color.rgb(r, g, b))

delay(step.toDuration(DurationUnit.MILLISECONDS))

}

i++

}

}

label.text += " - The components of strong connectivity are constructed. \n"

label.text += " - Amount: $i\n"

delay(step.toDuration(DurationUnit.MILLISECONDS)/2)

graph.graph.forEach { it.visited = false }

window.children.clear()

draw.drawNodeWithColor(colors)

draw.drawText()

draw.drawEdge()

label.text += " - The edges have original direction in the graph.\n"

downlabel.text = ""

find\_bridges()

return result

}

fun find\_bridges(){

for(node in graph.graph){

for(close in node.adjacents){

if(node.circle.fill != close.circle.fill){

for(lines in node.List\_of\_Lines){

if(lines.second == close){

lines.first.strokeWidth = 3.5

}

}

}

}

}

}

}

//Для проверки работы алгоритма запускаем

fun main() {

}

Файл DrawableGraph.kt

package com.example.practice

import javafx.scene.layout.AnchorPane

import javafx.scene.paint.Color

import javafx.scene.shape.Line

import javafx.scene.text.Font

import kotlin.math.\*

class DrawableGraph(var FrontPane: AnchorPane, n: Int = 5, m: Int = 7, var graph: OrientedGraph = OrientedGraph()){

init{

graph.fillGraph(n, m)

}

fun drawNode(){

val increment = 360.0/graph.graph.size

for(i in 0 until graph.graph.size){

val y = FrontPane.height / 2 + 200 \* sin(Math.toRadians((increment\*i)))

val x = FrontPane.width / 2 + 200 \* cos(Math.toRadians((increment\*i)))

val circle = graph.graph[i].createCircle(x, y, 20.0)

circle.fill = Color.BLACK

circle.fill

FrontPane.children.add(circle)

}

}

fun drawNodeWithColor(colors: MutableMap<Int, Color>){

for(i in 0 until graph.graph.size){

val circle = graph.graph[i].circle

circle.fill = colors[i]

FrontPane.children.add(circle)

}

}

fun drawNodeWithStandart(){

for(i in 0 until graph.graph.size){

val circle = graph.graph[i].circle

circle.fill = Color.BLACK

FrontPane.children.add(circle)

}

}

fun drawEdge(){

graph.graph.forEach {

for(elem in it.adjacents){

val slope = (it.circle.centerY - elem.circle.centerY) / (it.circle.centerX - elem.circle.centerX)

val lineAngle = atan(slope)

val arrowAngle = if (it.circle.centerX >= elem.circle.centerX) Math.toRadians(11.0) else -Math.toRadians(168.0)

val line: Line

val arrow1 = Line()

val arrow2 = Line()

val arrowLength = 200 / 10

val arg1 = findCollision(it.circle.centerX, it.circle.centerY, elem.circle.centerX, elem.circle.centerY, elem.circle.radius)

val arg2 = findCollision(elem.circle.centerX, elem.circle.centerY, it.circle.centerX, it.circle.centerY, it.circle.radius)

val ax = arg1.first

val ay = arg1.second

val bx = arg2.first

val by = arg2.second

line = Line(bx + it.circle.centerX,by + it.circle.centerY,ax + elem.circle.centerX,ay + elem.circle.centerY)

arrow1.startX = ax + elem.circle.centerX

arrow1.startY = ay + elem.circle.centerY

arrow1.endX = ax + elem.circle.centerX + arrowLength \* cos(lineAngle - arrowAngle)

arrow1.endY = ay + elem.circle.centerY + arrowLength \* sin(lineAngle - arrowAngle)

arrow2.startX = ax + elem.circle.centerX

arrow2.startY = ay + elem.circle.centerY

arrow2.endX = ax + elem.circle.centerX + arrowLength \* cos(lineAngle + arrowAngle)

arrow2.endY = ay + elem.circle.centerY + arrowLength \* sin(lineAngle + arrowAngle)

line.strokeWidth = 2.0

arrow1.strokeWidth = 2.0

arrow2.strokeWidth = 2.0

it.List\_of\_Lines.add(Pair(line, elem))

FrontPane.children.addAll(line, arrow1, arrow2)

}

}

}

fun drawReverseEdge(){

graph.graph.forEach {

for(elem in it.revadjacents){

val slope = (it.circle.centerY - elem.circle.centerY) / (it.circle.centerX - elem.circle.centerX)

val lineAngle = atan(slope)

val arrowAngle = if (it.circle.centerX >= elem.circle.centerX) Math.toRadians(11.0) else -Math.toRadians(168.0)

val line: Line

val arrow1 = Line()

val arrow2 = Line()

val arrowLength = 200 / 10

val arg1 = findCollision(it.circle.centerX, it.circle.centerY, elem.circle.centerX, elem.circle.centerY, elem.circle.radius)

val arg2 = findCollision(elem.circle.centerX, elem.circle.centerY, it.circle.centerX, it.circle.centerY, it.circle.radius)

val ax = arg1.first

val ay = arg1.second

val bx = arg2.first

val by = arg2.second

line = Line(bx + it.circle.centerX,by + it.circle.centerY,ax + elem.circle.centerX,ay + elem.circle.centerY)

arrow1.startX = ax + elem.circle.centerX

arrow1.startY = ay + elem.circle.centerY

arrow1.endX = ax + elem.circle.centerX + arrowLength \* cos(lineAngle - arrowAngle)

arrow1.endY = ay + elem.circle.centerY + arrowLength \* sin(lineAngle - arrowAngle)

arrow2.startX = ax + elem.circle.centerX

arrow2.startY = ay + elem.circle.centerY

arrow2.endX = ax + elem.circle.centerX + arrowLength \* cos(lineAngle + arrowAngle)

arrow2.endY = ay + elem.circle.centerY + arrowLength \* sin(lineAngle + arrowAngle)

line.strokeWidth = 2.0

arrow1.strokeWidth = 2.0

arrow2.strokeWidth = 2.0

FrontPane.children.addAll(line, arrow1, arrow2)

}

}

}

fun drawText(){

for(i in 0 until graph.graph.size){

val x = graph.graph[i].circle.centerX

val y = graph.graph[i].circle.centerY

val text = graph.graph[i].createText(i, x, y)

text.font = Font(16.0)

text.fill = Color.WHITE

FrontPane.children.add(text)

}

}

/\*\*

\* Функция находящая точку соприкосновнения прямой и окружности

\* @param x1 Точка через которую проходит прямая

\* @param y1 Точка через которую проходит прямая

\* @param x2 Точка через которую проходит прямая и точка центра окружности

\* @param y2 Точка через которую проходит прямая и точка центра окружности

\* @param r Радиус окружности

\* @return Возвращает точку соприкосновения прямой и окружности в виде: Pair<Double, Double>

\*/

private fun findCollision(x1: Double, y1: Double, x2: Double, y2: Double, r: Double): Pair<Double, Double>{

val k = (y1 - y2) / (x1 - x2)

val a = -k

val b = 1

val c = 0

val x0 = -a \* c / (a \* a + b \* b)

val y0 = -b \* c / (a \* a + b \* b)

if (abs(c \* c - r \* r \* (a \* a + b \* b)) < 0.01) {

return Pair(x0,y0)

}

else{

val d = r \* r - c \* c / (a \* a + b \* b)

val mult = sqrt(d / (a \* a + b \* b))

val ax: Double

val ay: Double

if (x1 < x2){

ax = x0 - b \* mult

ay = y0 + a \* mult

}

else {

ax = x0 + b \* mult

ay = y0 - a \* mult

}

return Pair(ax, ay)

}

}

}

**Файл Graph**

package com.example.practice

//Интерфейс, который задаёт поведение для графов

interface Graph {

fun fillGraph(nodes: Int,edges: Int)

fun clearGraph()

}

**Файл GraphGenerator**

package com.example.practice

class GraphGenerator {

//Генерация графа

fun generateGraph(n: Int, m: Int): ArrayList<Node> {

val graph = ArrayList<Node>()

var edges = m

var nodes = n

if (m > n \* (n - 1)) {

println("Invalid data! It will be change to n\*(n-1) & n.")

edges = n \* (n - 1)

nodes = n

}

if (n < 0) {

edges = 0

nodes = 0

}

//если количество узлов/мостов отрицательно или равно 0 -> в циклы не входим

for (i in 1..nodes) {

graph.add(Node(i))

}

//Повторяем кол-во вершин раз

repeat(edges) {

var node1: Int

var node2: Int

do {

node1 = (0 until nodes).random()

node2 = (0 until nodes).random()

} while (node1 == node2 || (intList(graph[node1].adjacents).contains(node2 + 1)))

graph[node1].adjacents.add(graph[node2])

graph[node2].revadjacents.add(graph[node1])

}

return graph

}

//Вспомогательная функция для представления названий вершин в виде IntArray

public fun intList(list: java.util.ArrayList<Node>): IntArray {

val arr = IntArray(list.size)

var i = 0

for (item in list) {

arr[i] = item.name

i++

}

return arr

}

}

**Файл Loader**

package com.example.practice

import org.json.JSONException

import java.io.File

import org.json.JSONObject

class Loader {

fun loadFromFile(filename: String): ArrayList<Node>{

var graph = ArrayList<Node>()

val json = File(filename).readText()

val jsonObj = JSONObject(json.substring(json.indexOf("{"), json.lastIndexOf("}") + 1))

val nodenumb = jsonObj.getInt("NodeNumber")

for (i in 1..nodenumb){

graph.add(Node(i))

}

for (i in 1..nodenumb){

var arr = jsonObj.getJSONArray("$i")

for (item in arr){

graph[i - 1].adjacents.add(graph[item.toString().toInt() - 1])

graph[item.toString().toInt() - 1].revadjacents.add(graph[i - 1])

}

try {

arr = jsonObj.getJSONArray("${i}.")

graph[i - 1].createCircle(arr[0].toString().toDouble(), arr[1].toString().toDouble(), 20.0)

}catch (error: JSONException){

}

}

return graph

}

}

**Файл Main.kt**

package com.example.practice

import javafx.application.Application

import javafx.fxml.FXMLLoader

import javafx.scene.Scene

import javafx.stage.Stage

import kotlinx.coroutines.\*

class HelloApplication : Application() {

override fun start(stage: Stage) {

val fxmlLoader = FXMLLoader(HelloApplication::class.java.getResource("sample.fxml"))

val scene = Scene(fxmlLoader.load(), 1200.0, 820.0)

stage.title = "Kosaraju"

stage.scene = scene

stage.show()

}

}

fun main(){

Application.launch(HelloApplication::class.java)

}

**Файл MainController**

package com.example.practice

import com.jfoenix.controls.JFXButton

import com.jfoenix.controls.JFXSlider

import javafx.fxml.FXML

import javafx.geometry.Point2D

import javafx.scene.control.Button

import javafx.scene.control.Label

import javafx.scene.control.TextField

import javafx.scene.input.MouseEvent

import javafx.scene.layout.AnchorPane

import javafx.scene.shape.Circle

import javafx.scene.shape.Line

import kotlinx.coroutines.\*

import java.awt.MouseInfo

import java.net.URL

import java.util.\*

import kotlin.coroutines.cancellation.CancellationException

class MainController {

@FXML

private lateinit var resources: ResourceBundle

@FXML

private lateinit var location: URL

@FXML

private lateinit var Downlabel: Label

@FXML

private lateinit var AlgBut: Button

@FXML

private lateinit var BackPane: AnchorPane

@FXML

private lateinit var ClearBut: Button

@FXML

private lateinit var DelBut: Button

@FXML

private lateinit var FrontPane: AnchorPane

@FXML

private lateinit var LoadBut: Button

@FXML

private lateinit var SaveBut: Button

@FXML

private lateinit var slider: JFXSlider

@FXML

private lateinit var VertLine: Line

@FXML

private lateinit var WindowForInput: TextField

@FXML

private lateinit var label: Label

@FXML

private lateinit var StepBut: Button

@FXML

private lateinit var GenBut: JFXButton

@FXML

private lateinit var MoveBut: JFXButton

private lateinit var draw: DrawableGraph

private lateinit var obj: Kosaraju

private val list\_lines = ArrayList<Triple<Line, Node, Node>>()

private var job: Job = Job()

private var job2: Job = Job()

private var n: Int = 5

private var m: Int = 7

@FXML

fun LoadBut(event: MouseEvent?) {

MoveBut.isDisable = false

DelBut.isDisable = false

label.text = ""

Downlabel.text = ""

if (job.isActive)

job.cancel()

draw = DrawableGraph(FrontPane)

FrontPane.children.clear()

val loader = Loader()

draw.graph.clearGraph()

draw.graph.graph = loader.loadFromFile("saved\_graph.json")

obj = Kosaraju(Downlabel, label, draw.graph)

if (draw.graph.graph.size > 0 && draw.graph.graph[0].circle.centerX.toInt() == 0 && draw.graph.graph[1].circle.centerY.toInt() == 0)

draw.drawNode()

else

draw.drawNodeWithStandart()

draw.drawEdge()

draw.drawText()

Downlabel.text = " - The graph from the file is loaded."

}

@FXML

fun SaveBut(event: MouseEvent?) {

label.text = ""

Downlabel.text = ""

MoveBut.isDisable = false

DelBut.isDisable = false

if (job.isActive) {

job.cancel()

FrontPane.children.clear()

draw.drawNode()

draw.drawEdge()

draw.drawText()

}

if (draw.graph.graph.size > 0) {

val saver = Saver()

saver.saveToFile(draw.graph.graph)

Downlabel.text = " - The current graph is saved."

}

}

@FXML

fun ClearClicked(event: MouseEvent?) {

DelBut.isDisable = false

MoveBut.isDisable = false

label.text = ""

Downlabel.text = ""

if (job.isActive)

job.cancel()

//obj = Kosaraju(draw.graph)

try {

draw.graph.clearGraph()

} catch (er: kotlin.UninitializedPropertyAccessException) {

return

}

FrontPane.children.clear()

}

@FXML

fun GenerateBut(event: MouseEvent) {

DelBut.isDisable = false

MoveBut.isDisable = false

draw = DrawableGraph(FrontPane, n, m)

Downlabel.text = " - The graph with $n vertexes and $m ages was built."

label.text = ""

if (job.isActive)

job.cancel()

if (WindowForInput.text != "") {

val Node\_Edge = WindowForInput.text

val regex = "((\\d?)|([1-9]\\d\*)) ((\\d?)|([1-9]\\d\*))".toRegex()

if (regex.matches(Node\_Edge)) {

this.n = Node\_Edge.split(' ')[0].toInt()

this.m = Node\_Edge.split(' ')[1].toInt()

draw = DrawableGraph(FrontPane, n, m)

Downlabel.text = ""

if (n == 0)

Downlabel.text = " - Empty graph was built."

else if (m > n \* (n - 1))

Downlabel.text = " - Invalid data! It has been change to \n n vertexes and n \* (n-1) ages."

else

Downlabel.text = " - The graph with $n vertexes and $m ages was built."

WindowForInput.clear()

} else {

Downlabel.text = " - Incorrect input!"

WindowForInput.clear()

return

}

}

obj = Kosaraju(Downlabel, label, draw.graph)

FrontPane.children.clear()

this.draw.drawNode()

this.draw.drawEdge()

this.draw.drawText()

}

@FXML

fun StartAlgorithm(event: MouseEvent?) {

label.text = ""

Downlabel.text = ""

MoveBut.isDisable = false

DelBut.isDisable = false

try {

obj.graph.graph.forEach { it.visited = false }

if (job.isActive) {

job.cancel()

obj.graph.order.clear()

println("job is done!")

}

obj = Kosaraju(Downlabel, label, draw.graph)

obj.start()

} catch (er: kotlin.UninitializedPropertyAccessException) {

return

}

}

@FXML

fun DeleteClick(event: MouseEvent?) {

label.text = ""

Downlabel.text = ""

MoveBut.isDisable = false

if (job.isActive)

job.cancel()

if (draw.graph.graph.size > 0){

DelBut.isDisable = true

try {

FrontPane.children.clear()

draw.drawNodeWithStandart()

draw.drawEdge()

draw.drawText()

for (node in draw.graph.graph) {

for (line in node.List\_of\_Lines) {

list\_lines.add(Triple(line.first, node, line.second))

}

}

for (line in list\_lines) {

line.first.setOnMouseClicked {

(run {

FrontPane.children.clear()

line.second.adjacents.remove(line.third)

line.third.revadjacents.remove(line.second)

line.second.List\_of\_Lines.remove(Pair(line.first, line.third))

for (el in draw.graph.graph) {

FrontPane.children.add(el.circle)

}

draw.drawEdge()

draw.drawText()

for (line2 in list\_lines) {

line2.first.setOnMouseClicked {}

}

})

}

}

for (node in draw.graph.graph) {

node.circle.setOnMouseClicked {

(run {

FrontPane.children.clear()

for (close1 in node.revadjacents) {

if (node in close1.adjacents) {

close1.adjacents.remove(node)

}

}

for (close2 in node.adjacents) {

if (node in close2.revadjacents) {

close2.revadjacents.remove(node)

}

}

if (node in draw.graph.order) {

draw.graph.order.remove(node)

}

node.adjacents.clear()

node.revadjacents.clear()

val ind = draw.graph.graph.indexOf(node)

draw.graph.graph.remove(node)

for (i in ind until draw.graph.graph.size) {

draw.graph.graph[i].name -= 1

}

for (el in draw.graph.graph) {

FrontPane.children.add(el.circle)

}

draw.drawEdge()

draw.drawText()

for (node2 in draw.graph.graph) {

node2.circle.setOnMouseClicked {}

}

})

}

node.txt.setOnMouseClicked {

(run {

FrontPane.children.clear()

for (close1 in node.revadjacents) {

if (node in close1.adjacents) {

close1.adjacents.remove(node)

}

}

for (close2 in node.adjacents) {

if (node in close2.revadjacents) {

close2.revadjacents.remove(node)

}

}

if (node in draw.graph.order) {

draw.graph.order.remove(node)

}

node.adjacents.clear()

node.revadjacents.clear()

val ind = draw.graph.graph.indexOf(node)

draw.graph.graph.remove(node)

for (i in ind until draw.graph.graph.size) {

draw.graph.graph[i].name -= 1

}

for (el in draw.graph.graph) {

FrontPane.children.add(el.circle)

}

draw.drawEdge()

draw.drawText()

for (node2 in draw.graph.graph) {

node2.txt.setOnMouseClicked {}

}

})

}

}

} catch (er: kotlin.UninitializedPropertyAccessException) {

return

}

}

}

fun stepBut(event: MouseEvent?) {

DelBut.isDisable = false

MoveBut.isDisable = false

label.text = ""

Downlabel.text = ""

val step: Int = slider.value.toInt()

if (job.isActive)

job.cancel()

FrontPane.children.clear()

try {

this.draw.drawNodeWithStandart()

this.draw.drawEdge()

this.draw.drawText()

obj = Kosaraju(Downlabel, label, draw.graph)

job = GlobalScope.launch(Dispatchers.Main) {

try {

if (draw.graph.graph.size > 0) {

obj.startForStep(draw, FrontPane, step)

}

} catch (er: CancellationException) {

println("Step by step is cancelled")

obj.graph.graph.forEach { it.visited = false }

obj.graph.order.clear()

}

}.also {

it.invokeOnCompletion { }

}

} catch (er: kotlin.UninitializedPropertyAccessException) {

return

}

}

@FXML

fun initialize() {

// this.draw = Drawablegraph(FrontPane, n, m)

}

var tmp\_circle: Circle? = Circle(0.0, 0.0, 20.0)

var x: Double = 0.0

var y: Double = 0.0

@FXML

fun TestForPane(event: MouseEvent) {

DelBut.isDisable = false

for (node in draw.graph.graph) {

node.circle.setOnMouseClicked {}

node.txt.setOnMouseClicked {}

}

for (line in list\_lines) {

line.first.setOnMouseClicked {}

}

if (MoveBut.isDisable) {

val p = MouseInfo.getPointerInfo().location

val point2D: Point2D = Point2D(event.x, event.y)

if (Circle(this.x, this.y, 20.0).contains(point2D) || MoveBut.contains(point2D)) {

return

}

FrontPane.children.clear()

tmp\_circle?.centerX = event.x

tmp\_circle?.centerY = event.y

for (el in draw.graph.graph) {

FrontPane.children.add(el.circle)

}

this.draw.drawEdge()

this.draw.drawText()

}

tmp\_circle = null

MoveBut.isDisable = false

}

@FXML

fun MoveClicked(event: MouseEvent) {

DelBut.isDisable = false

try {

if (draw.graph.graph.size > 0) {

MoveBut.isDisable = true

for (node in draw.graph.graph) {

node.circle.setOnMouseClicked { mouseEvent ->

(run {

this.x = node.circle.centerX

this.y = node.circle.centerY

this.tmp\_circle = node.circle

})

}

node.txt.setOnMouseClicked { mouseEvent ->

(run {

this.x = node.circle.centerX

this.y = node.circle.centerY

this.tmp\_circle = node.circle

})

}

}

}

} catch (er: kotlin.UninitializedPropertyAccessException) {

return

}

}

}

**Файл Node**

package com.example.practice

import javafx.scene.shape.Circle

import javafx.scene.shape.Line

import javafx.scene.text.Text

class Node(var name: Int, var x: Double = 0.0, var y: Double = 0.0, var r: Double = 0.0){

var visited: Boolean = false

var timeout: Int = 0

var adjacents = ArrayList<Node>()

var revadjacents = ArrayList<Node>()

var circle = Circle(x, y, r)

var txt = Text(x, y, "$name")

var List\_of\_Lines = ArrayList<Pair<Line, Node>>()

fun createCircle(x: Double, y: Double, r: Double): Circle{

this.circle = Circle(x, y, r)

return this.circle

}

fun createText(name: Int, x: Double, y: Double): Text{

this.txt = Text(x, y, "$name")

return this.txt

}

}

**Файл OrientedGraph**

package com.example.practice

import java.util.ArrayList

class OrientedGraph: Graph {

var order = ArrayList<Node>()

var graph = ArrayList<Node>()

// Генерирует граф с заданным количеством рёбер и вершин(сделать проверку при вводе на невозможное кол-во вершин)

override fun fillGraph(nodes: Int, edges: Int){

graph.clear()

order.clear()

graph = GraphGenerator().generateGraph(nodes, edges)

}

override fun clearGraph() {

graph.clear()

order.clear()

}

}

**Файл Saver**

package com.example.practice

import org.json.JSONException

import org.json.JSONObject

import java.io.FileWriter

import java.io.PrintWriter

import java.nio.charset.Charset

class Saver {

fun saveToFile(graph: ArrayList<Node>){

val path = "saved\_graph.json"

val json = JSONObject()

try{

json.put("NodeNumber", graph.size)

for (node in graph) {

json.put("${node.name}", GraphGenerator().intList(node.adjacents))

}

for (node in graph) {

json.put("${node.name}.", mutableListOf(node.circle.centerX, node.circle.centerY))

}

} catch (e: JSONException){

e.printStackTrace()

}

try {

PrintWriter(FileWriter(path, Charset.defaultCharset()))

.use { it.write(json.toString()) }

} catch (e: Exception) {

e.printStackTrace()

}

}

}

**Файл Saver**

package com.example.practice

import org.json.JSONException

import org.json.JSONObject

import java.io.FileWriter

import java.io.PrintWriter

import java.nio.charset.Charset

class Saver {

fun saveToFile(graph: ArrayList<Node>){

val path = "saved\_graph.json"

val json = JSONObject()

try{

json.put("NodeNumber", graph.size)

for (node in graph) {

json.put("${node.name}", GraphGenerator().intList(node.adjacents))

}

for (node in graph) {

json.put("${node.name}.", mutableListOf(node.circle.centerX, node.circle.centerY))

}

} catch (e: JSONException){

e.printStackTrace()

}

try {

PrintWriter(FileWriter(path, Charset.defaultCharset()))

.use { it.write(json.toString()) }

} catch (e: Exception) {

e.printStackTrace()

}

}

}

**Файл GraphGeneratorTest**

import org.junit.jupiter.api.Test

import com.example.practice.GraphGenerator

import org.junit.jupiter.api.Assertions.\*

import org.junit.jupiter.api.DisplayName

class GraphGeneratorTest {

@Test

@DisplayName("Discharged graph with five vertexes")

fun generateGraphTest1() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(5, 0)

val expected = "1: {}; 2: {}; 3: {}; 4: {}; 5: {}; "

var actual = ""

for(vertex in nodeArray){

actual += "${vertex.name}: {"

for(i in vertex.adjacents)

actual += "${i.name} "

actual += "}; "

}

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("2-full graph")

fun generateGraphTest2() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(2, 2)

val expected = "1: { 2 }; 2: { 1 }; "

var actual = ""

for(vertex in nodeArray){

actual += "${vertex.name}: {"

for(i in vertex.adjacents)

actual += " ${i.name}"

actual += " }; "

}

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with 1 vertex and 12 ages")

fun generateGraphTest3() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(1, 12)

var actual = ""

val expected = "1: { }; "

for(vertex in nodeArray){

actual += "${vertex.name}: {"

for(i in vertex.adjacents)

actual += " ${i.name}"

actual += " }; "

}

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with 5 vertexes and 12 ages")

fun generateGraphTest4() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(5, 12)

val nodeAmount = nodeArray.size

var vertexAmount = 0

var actual = ""

val expected = "5 12"

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with 0 vertex")

fun generateGraphTest6() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(0, 0)

val nodeAmount = nodeArray.size

var vertexAmount = 0

var actual = ""

val expected = "0 0"

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with a negative number of vertexes")

fun generateGraphTest5() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(-10, 0)

val nodeAmount = nodeArray.size

var vertexAmount = 0

var actual = ""

val expected = "0 0"

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with one vertex")

fun generateGraphTest7() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(1, 0)

val expected = "1: {}; "

var actual = ""

for(vertex in nodeArray){

actual += "${vertex.name}: {"

for(i in vertex.adjacents)

actual += "${i.name} "

actual += "}; "

}

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with a negative number of ages")

fun generateGraphTest8() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(3, -10)

val nodeAmount = nodeArray.size

var vertexAmount = 0

var actual = ""

val expected = "3 0"

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with a negative number of vertexes and ages")

fun generateGraphTest9() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(-10, -10)

val nodeAmount = nodeArray.size

var vertexAmount = 0

var actual = ""

val expected = "0 0"

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with a negative number of vertexes & 3 ages")

fun generateGraphTest10() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(-10, 3)

val nodeAmount = nodeArray.size

var vertexAmount = 0

var actual = ""

val expected = "0 0"

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("Graph with a 2 vertexes and 12 ages")

fun generateGraphTest11() {

var generator = GraphGenerator()

val nodeArray = generator.generateGraph(2, 12)

val nodeAmount = nodeArray.size

var vertexAmount = 0

var actual = ""

val expected = "2 2"

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

}

**Файл OrientedGraphTest**

package com.example.**practice**

import org.junit.jupiter.api.Assertions.\*

import org.junit.jupiter.api.DisplayName

import org.junit.jupiter.api.Tag

import org.junit.jupiter.api.Test

class OrientedGraphTest {

//сравниваю количество вершин и ребер (как проверка генератора)

@Test

@DisplayName("fillGraph: Fill Graph with one vertex")

fun fillGraphTest1() {

var graph = OrientedGraph()

graph.fillGraph(1,0)

val nodeArray = graph.graph

val expected = "1 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("fillGraph: Fill disconnected graph (3 vertexes)")

fun fillGraphTest2() {

var graph = OrientedGraph()

graph.fillGraph(3,0)

val nodeArray = graph.graph

val expected = "3 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("fillGraph: Fill 2-full graph")

fun fillGraphTest3() {

var graph = OrientedGraph()

graph.fillGraph(2,2)

val nodeArray = graph.graph

val expected = "2 2"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("fillGraph: Fill ordinary graph (5 vertexes, 12 ages)")

fun fillGraphTest4() {

var graph = OrientedGraph()

graph.fillGraph(5,12)

val nodeArray = graph.graph

val expected = "5 12"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("fillGraph: Fill graph with a negative number of vertexes (-5 vertexes, 0 ages)")

fun fillGraphTest5() {

var graph = OrientedGraph()

graph.fillGraph(-5,0)

val nodeArray = graph.graph

val expected = "0 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("fillGraph: Fill graph with a 0 number of vertexes (0 vertexes, 0 ages)")

fun fillGraphTest6() {

var graph = OrientedGraph()

graph.fillGraph(-5,0)

val nodeArray = graph.graph

val expected = "0 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("fillGraph: Fill graph with a negative number of ages (5 vertexes, -10 ages)")

fun fillGraphTest7() {

var graph = OrientedGraph()

graph.fillGraph(5,-10)

val nodeArray = graph.graph

val expected = "5 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("fillGraph: Fill graph with a negative number of vertexes and ages (-5 vertexes, -10 ages)")

fun fillGraphTest8() {

var graph = OrientedGraph()

graph.fillGraph(-5,-10)

val nodeArray = graph.graph

val expected = "0 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("fillGraph: Fill graph with a maximum+1 number of ages (5 vertexes, 21 ages)")

fun fillGraphTest9() {

var graph = OrientedGraph()

graph.fillGraph(5,21)

val nodeArray = graph.graph

val expected = "5 20"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("clearGraph: Clear empty Graph")

fun clearGraphTest1() {

var graph = OrientedGraph()

graph.fillGraph(0,0)

graph.clearGraph()

val nodeArray = graph.graph

val expected = "0 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("clearGraph: Clear 2-full graph")

fun clearGraphTest2() {

var graph = OrientedGraph()

graph.fillGraph(2,2)

graph.clearGraph()

val nodeArray = graph.graph

val expected = "0 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("clearGraph: Clear graph with one vertex")

fun clearGraphTest3() {

var graph = OrientedGraph()

graph.fillGraph(1,0)

graph.clearGraph()

val nodeArray = graph.graph

val expected = "0 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

@Test

@DisplayName("clearGraph: Clear disconnected Graph")

fun clearGraphTest4() {

var graph = OrientedGraph()

graph.fillGraph(5,0)

graph.clearGraph()

val nodeArray = graph.graph

val expected = "0 0"

var actual = ""

val nodeAmount = nodeArray.size

var vertexAmount = 0

for(vertex in nodeArray){

for(i in vertex.adjacents)

vertexAmount += 1

}

actual = "$nodeAmount $vertexAmount"

println(actual)

assertEquals(expected, actual)

}

}

**Файл KosarajuTest**

import com.example.practice.Kosaraju

import com.example.practice.Loader

import com.example.practice.Node

import de.saxsys.javafx.test.JfxRunner

import javafx.scene.control.Label

import org.junit.jupiter.api.Assertions.\*

import org.junit.jupiter.api.DisplayName

import org.junit.jupiter.api.Test

class KosarajuTest {

@Test

@DisplayName("Kosaraju: Graph with one vertex")

fun startTest1() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_1.json")

var first\_label = Label()

var second\_label = Label()

var kosaraju: Kosaraju = Kosaraju(first\_label, second\_label)

kosaraju.n = 1

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [1] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: 2-full graph")

fun startTest2() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_2.json")

kosaraju.n = 2

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [1, 2] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: 3-full graph")

fun startTest3() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_3.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [1, 2, 3] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Disconnected graph with three component")

fun startTest4() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_4.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [3] 2: [2] 3: [1] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Ordinary Graph")

fun startTest5() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_5.json")

kosaraju.n = 8

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [1, 5, 2] 2: [3, 4] 3: [7, 6] 4: [8] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Graph with one component (3 vertexes, 3 ages)")

fun startTest6() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_6.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [1, 3, 2] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Graph with one component (3 vertexes, 4 ages)")

fun startTest7() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_7.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [1, 2, 3] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Graph with two components (3 vertexes, 2 ages)")

fun startTest8() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_8.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [3] 2: [1, 2] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Graph with three components (3 vertexes, 2 ages)")

fun startTest9() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_9.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [1] 2: [2] 3: [3] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Graph with three components (3 vertexes, 1 age)")

fun startTest10() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_10.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [3] 2: [1] 3: [2] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Graph with 0 vertex")

fun startTest11() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_6.json")

kosaraju.n = 0

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = ""

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Graph with n vertexes & n(n-1)+1 ages")

fun startTest12() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_12.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [1, 2, 3] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("Kosaraju: Graph with two components (4 vertexes, 4 ages)")

fun startTest13() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_13.json")

kosaraju.n = 4

kosaraju.graph.graph = node\_list

val actual = kosaraju.start()

val expected = "1: [3, 4] 2: [1, 2] "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with one vertex")

fun DFSTest1() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_1.json")

kosaraju.n = 1

kosaraju.graph.graph = node\_list

kosaraju.dfs(kosaraju.graph, node\_list[0])

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "2 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: 2-full graph")

fun DFSTest2() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_2.json")

kosaraju.n = 2

kosaraju.graph.graph = node\_list

kosaraju.dfs(kosaraju.graph, node\_list[0])

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "4 3 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: 3-full graph")

fun DFSTest3() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_3.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

kosaraju.dfs(kosaraju.graph, node\_list[0])

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "6 5 4 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Disconnected graph with three components")

fun DFSTest4() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_4.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "2 4 6 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Ordinary Graph")

fun DFSTest5() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_5.json")

kosaraju.n = 8

kosaraju.graph.graph = node\_list

kosaraju.dfs(kosaraju.graph, node\_list[0])

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "16 15 12 7 14 10 11 6 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with one component (3 vertexes, 3 ages)")

fun DFSTest6() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_6.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "6 5 4 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with one component (3 vertexes, 4 ages)")

fun DFSTest7() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_7.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "6 5 4 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with two components (3 vertexes, 2 ages)")

fun DFSTest8() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_8.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "4 3 6 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with three components (3 vertexes, 2 ages)")

fun DFSTest9() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_9.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "6 5 4 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with three components (3 vertexes, 1 age)")

fun DFSTest10() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_10.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "4 3 6 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with 0 vertex")

fun DFSTest11() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_11.json")

kosaraju.n = 0

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "0 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with n vertexes & n(n-1)+1 ages")

fun DFSTest12() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_12.json")

kosaraju.n = 3

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "6 5 4 "

assertEquals(expected, actual)

}

@org.junit.jupiter.api.Test

@org.junit.jupiter.api.DisplayName("DFS: Graph with two components (4 vertexes, 4 ages)")

fun DFSTest13() {

var node\_list: ArrayList<Node> = Loader().loadFromFile("test\_graph\_13.json")

kosaraju.n = 4

kosaraju.graph.graph = node\_list

for (vertex in 0 until kosaraju.n) {

if (!kosaraju.graph.graph[vertex].visited) {

kosaraju.dfs(kosaraju.graph, kosaraju.graph.graph[vertex])

}

}

var actual = ""

for(i in 0 until kosaraju.graph.graph.size){

actual += "${kosaraju.graph.graph[i].timeout} "

}

println(actual)

val expected = "4 3 8 7 "

assertEquals(expected, actual)

}

}

**Файл test\_graph\_1.json**

{"1":[],"NodeNumber":1}

**Файл test\_graph\_2.json**

{"1":[2],"2":[1],"NodeNumber":2}

**Файл test\_graph\_3.json**

{"1":[2,3],"2":[1,3],"3":[1,2],"NodeNumber":3}

**Файл test\_graph\_4.json**

{"1":[],"2":[],"3":[],"NodeNumber":3}

**Файл test\_graph\_5.json**

{"1":[2],"2":[3,5,6],"3":[4,7],"4":[3,8],"5":[1,6],"6":[7],"7":[6,8],"8":[],"NodeNumber":8}

**Файл test\_graph\_6.json**

{"1":[2],"2":[3],"3":[1],"NodeNumber":3}

**Файл test\_graph\_7.json**

{"1":[2],"2":[1,3],"3":[2],"NodeNumber":3}

**Файл test\_graph\_8.json**

{"1":[2],"2":[1],"3":[],"NodeNumber":3}

**Файл test\_graph\_9.json**

{"1":[2],"2":[3],"3":[],"NodeNumber":3}

**Файл test\_graph\_10.json**

{"1":[2],"2":[],"3":[],"NodeNumber":3}

**Файл test\_graph\_11.json**

{"1":[],"NodeNumber":1}

**Файл test\_graph\_12.json**

{"1":[1,2,3],"2":[1,3],"3":[1,2],"NodeNumber":3}

**Файл test\_graph\_13.json**

{"1":[2],"2":[1],"3":[4],"4":[3],"NodeNumber":4}