**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.**

| Студентка гр. 1381 |  | Тулегенова А.О. |
| --- | --- | --- |
| Студент гр. 1381 |  | Сагидуллин Э.Р. |
| Студентка гр. 1381 |  | Васильева О.М. |
| Руководитель |  | Фирсов М.А. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**НА УЧЕБНУЮ ПРАКТИКУ**

| Студентка Тулегенова А.О. группы 1381 | | |
| --- | --- | --- |
| Студент Сагидуллин Э.Р. группы 1381 | | |
| Студентка Васильева О.М. группы 1381  Тема практики: Сильно связные компоненты орграфа | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Kotlin с графическим интерфейсом.  Алгоритм: алгоритм Косарайю. | | |
| Сроки прохождения практики: 30.06.2023 – 13.07.2023 | | |
| Дата сдачи отчета: 00.07.2023 | | |
| Дата защиты отчета: 00.07.2023 | | |
|  | | |
| Студентка |  | Тулегенова А.О. |
| Студент |  | Сагидуллин Э.Р. |
| Студентка |  | Васильева О.М. |
| Руководитель |  | Фирсов М.А. |

**АННОТАЦИЯ**

Данная работа посвящена изучению и реализации алгоритма Косарайю для нахождения сильно связных компонент в орграфе. В работе представлен графический интерфейс, разработанный для визуализации работы алгоритма Косарайю. Интерфейс позволяет пользователю вводить орграфы, состоящие из вершин и направленных ребер, а затем выполнять алгоритм Косарайю для поиска сильно связных компонент. Результаты работы алгоритма визуализируются в виде графической диаграммы, где каждая сильно связная компонента обозначается отдельным цветом.

**SUMMARY**

This work is devoted to the study and implementation of the Kosaraju algorithm for finding strongly connected components in a digraph. The paper presents a graphical interface designed to visualize the operation of the Kosaraju algorithm. The interface allows the user to enter digraphs consisting of vertices and directed edges, and then execute the Kosaraju algorithm to search for strongly connected components. The results of the algorithm are visualized in the form of a graphical diagram, where each strongly connected component is indicated by a separate color.

**СОДЕРЖАНИЕ**

# 

[**ВВЕДЕНИЕ**](#_gyc9a9y4raod) **4**

[**1. ТРЕБОВАНИЯ К ПРОГРАММЕ**](#_iyclvoeodmar) **5**

[1.1. Исходные Требования к программе](#_ib6x8bzhmma9) 6

[1.1.1 Требования к графическому интерфейсу](#_ru1blwhyy7e4) 6

[1.1.2. Требования к вводу исходных данных](#_p3j3mdx71wf5) 9

[1.1.3. Требования к визуализации](#_7vk8db51o0g) 9

[1.2. Уточнение требований после сдачи 1-ой версии](#_7ls0zhy5fbsn) 10

[**2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ**](#_vp40dnuh7f74) **10**

[2.1. План разработки](#_ws0e5gpkcjlr) 11

[2.2. Распределение ролей в бригаде](#_bqa51o3on2li) 13

[**3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ**](#_dkzs372ejbua) **14**

[3.1 Основные классы](#_wwf2f8t7bny5) 15

[3.2. Основные методы](#_eaksj6628l0k) 17

[**4. ТЕСТИРОВАНИЕ**](#_wmaeepso6u3d) **22**

[4.1. План тестирования](#_iw425ld25liq) 23

[4.2. Ошибки взаимодействия с графом](#_ru8kaabk1w5) 24

[4.3. Граничные случаи для визуализации алгоритма Косарайю](#_chorq2cb1f7z) 25

[4.4. Ошибки чтения файла](#_46fppvnh0oih) 30

[4.5. Ошибки интерфейса](#_xa3hoho3okhg) 32

[**ЗАКЛЮЧЕНИЕ**](#_ufqmlvelzls) **36**

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**](#_ouutbftkdyi6) **37**

[**ПРИЛОЖЕНИЕ А**](#_ibcrbntl3luv) **38**

# **ВВЕДЕНИЕ**

Данная работа посвящена изучению и реализации алгоритма Косарайю для определения сильно связных компонент в орграфах. Сильно связные компоненты являются важным понятием в теории графов и находят широкое применение в различных областях, таких как анализ социальных сетей, оптимизация транспортных систем и другие.

Целью данной практической работы является исследование и реализация алгоритма Косарайю, а также разработка графического интерфейса для визуализации его работы. Графический интерфейс предоставляет пользователю возможность вводить орграфы с вершинами и направленными ребрами, а затем запускать алгоритм Косарайю для поиска сильно связных компонент. Результаты работы алгоритма визуализируются в виде графической диаграммы, где каждая сильно связная компонента обозначается уникальным цветом.

Реализация алгоритма Косарайю включает в себя несколько этапов: первоначальная обработка графа, построение транспонированного графа, выполнение обхода в глубину согласно измененному порядку вершин и определение сильно связных компонент. Полученная реализация алгоритма и графический интерфейс предоставляют удобный инструмент для анализа сильно связных компонент в орграфах. Исследователи и практики могут эффективно использовать этот инструмент для изучения структуры орграфов и выявления важных связей и зависимостей.

# **1. ТРЕБОВАНИЯ К ПРОГРАММЕ**

## 1.1. Исходные Требования к программе

### 1.1.1 Требования к графическому интерфейсу

Данное приложение должно иметь графический интерфейс, что означает, что пользователи смогут взаимодействовать с программой с помощью графических элементов, таких как кнопки, поля ввода и т. д. Также следует предусмотреть возможности взаимодействия с графическими элементами с помощью мыши.

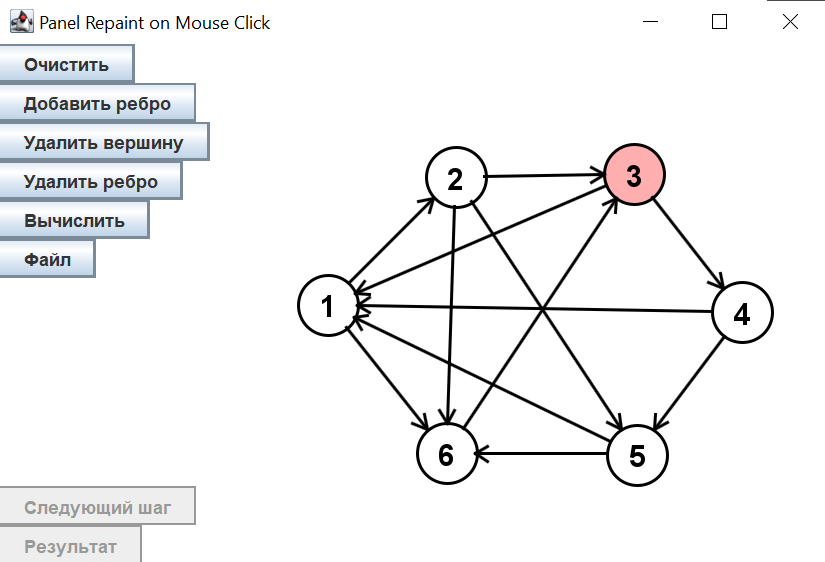
****

Рисунок 1. Скриншот графического интерфейса.

Функциональность программы:

1. Кнопка "Очистить":

При нажатии данной кнопки весь граф на холсте будет удален. Все вершины и ребра, отображаемые на холсте, будут удалены, и холст будет очищен.

2. Кнопка "Добавить ребро":

Для соединения двух вершин необходимо выполнить следующие шаги:

1. Выделите одну вершину, щелкнув на нее.

2. Выделите вторую вершину, щелкнув на нее.

3. После выполнения этих шагов между выделенными вершинами будет установлено ребро.

Ребро будет нарисовано между выбранными вершинами на холсте.

Текстовые пояснения при этом также будут отображаться на графическом интерфейсе. Для удобства, первая вершина выделяется цветом.

3. Кнопка "Удалить вершину":

При нажатии данной кнопки будет удалена выбранная вершина и все ребра, связанные с этой вершиной. Щелчок на вершину, которую нужно удалить, выберет ее и все соединенные с ней ребра будут также удалены. При этом будет отображаться текстовое пояснение с просьбой кликнуть на вершину.

4. Кнопка "Удалить ребро":

Для удаления ребра между двумя вершинами необходимо выполнить следующие шаги:

1. Выделите одну вершину, щелкнув на нее.

2. Выделите вторую вершину, щелкнув на нее.

3. После выполнения этих шагов между выделенными вершинами будет удалено ребро, если оно существует.

Текстовые пояснения при этом также будут отображаться на графическом интерфейсе. Для удобства, первая вершина выделяется цветом.

5. Кнопка "Вычислить":

При нажатии данной кнопки будет запущен алгоритм Косарайю для поиска сильно связных компонент графа. После нажатия на холсте будет представлен первый шаг алгоритма. Выводиться текстовое пояснение, что агоритм был запущен и объяснение того, что произошло на первом шаге. Также при нажатии этой кнопки, кнопки, ответственные за удаление/добавление ребер и вершин, становятся недоступными, пока не будут выполнены все шаги алгоритма.

6. Кнопка "Файл":

При нажатии данной кнопки открывается проводник для выбора файла. Пользователь может выбрать файл, содержащий описание графа в виде списка ребер. Выбранный файл будет загружен, и граф из файла будет отображен на холсте.

7. Кнопка "Следующий шаг":

При нажатии данной кнопки происходит переход к следующему шагу алгоритма. При этом выводятся текстовые пояснения того, какие изменения произошли на данном шаге. Если был выполнен последний шаг алгоритма, то выполнение алгоритма завершаются и кнопки, ответственные за удаление/добавление ребер и вершин, вновь становятся доступными. При этом кнопка “Следующий шаг” заблокирована, пока не запущен алгоритм, без запуска алгоритма в них нет потребности.

8. Кнопка "Результат":

При нажатии данной кнопки сразу визуализируется конечный результат выполнения программы. Кнопки, ответственные за удаление/добавление ребер и вершин, вновь становятся доступными.. Появляется текстовое пояснение, что на холсте представлено конечное разбиение вершин на компоненты связности. Выполнение алгоритма завершено. При этом кнопка и “Результат” заблокирована, пока не запущен алгоритм, без запуска алгоритма в них нет потребности.

Примечания:

Вся визуализация графа и выполнение операций происходят на холсте программы. Для выделения вершины и соединения ребер необходимо использовать щелчки мыши. Графический интерфейс программы обеспечивает удобное взаимодействие с графами и операциями над ними.   
 Добавление вершин происходит при нажатии на холст мышкой. Также возможно перемещение вершин по холсту, если зажать вершину левой кнопкой мыши и перетащить ее по холсту.

### 1.1.2. Требования к вводу исходных данных

Должна быть возможность считывания входных данных из файла. Данные будут представлены в следующем виде: на каждой строке будут записаны через пробел индексы двух вершин (индексы заданы в виде чисел), соответствующие вершинам и направленному ребру графа. Также входные данные могут создаваться при работе в приложении, то есть при добавлении вершин и ребер на панели приложения, а именно при нажатии мышки или соответствующей кнопки на панели приложения, задается некоторый граф, на котором будет запускаться алгоритм. После задания графа в приложении или с помощью файла данные будут преобразованы к спискам смежности.

### 1.1.3. Требования к визуализации

Приложение предусматривает возможность взаимодействия с графическим интерфейсом, при нажатии левой кнопки мышки по холсту добавляется вершина, также есть возможность перетаскивать вершины по холсту зажатием левой кнопки мышки.

Приложение ясное и удобное для пользователя, при нажатии кнопок выводятся текстовые пояснение с просьбой кликнуть на вершину, а в случае добавления/удаления ребра будет ожидаться нажатие двух вершин и после нажатия на первую вершину она будет выделяться цветом.  
 Алгоритм предоставляет возможность пошагового отображения итераций, для этого можно воспользоваться кнопкой “Следующий шаг”, на каждом шаге будут также выведены текстовые пояснения того, какие изменения произошли. Также во время работы алгоритма будет выполняться обход графа в глубину, поэтому на каждом шаге будет подсвечиваться текущая вершина. В конце алгоритма мы должны получить разбиение вершин на компоненты сильной связности, оно будет выполнено с помощью раскрашивания вершин в разные цвета, в зависимости от того, какой компоненте они принадлежат, ребра соединяющие вершины одной компоненты также будут раскрашены в соответствующие цвета. После завершения алгоритма будет выведена соответствующая надпись и кнопки будут разблокированы. Вершины будут сохранять раскраску до тех пор, пока граф не будет изменен, а именно пока не будет удалено/добавлено хоть одно ребро или вершина.

## 1.2. Уточнение требований после сдачи 1-ой версии

1. В приложении должна быть возможность вводить рёбра с использованием правой кнопки мыши, без необходимости нажатия на кнопку "Добавить ребро".
2. В приложении должна быть возможность перетаскивать вершины во время и в конце алгоритма (без изменения окраски).
3. Чтобы был ясен порядок обхода на 3 шаге, вершины следует помечать номерами в обходе и/или вывести порядок перед выполнением 3 шага в виде строки чисел, при этом оставить эту строку на экране на время выполнения 3 шага.

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

## 2.1. План разработки

1. Выбрать тему проекта. Разработать спецификацию и план разработки. Распределение ролей в бригаде.
2. После согласования спецификации и плана разработки:  
   Разработка приложения была разделена на три основные части: реализация алгоритма Косарайю, а именно класса Algorithm; реализация различных способов задания графа с помощью файла и графическим способом: добавление/удаление вершин и ребер в графе с помощью графического интерфейса, а также добавление кнопок для запуска и демонстрации шагов алгоритма (прототип приложения); объединение первых двух частей: визуализация работы алгоритма: пошаговое изображение работы алгоритма.

Итого будет происходить параллельная разработка алгоритма Косарайю и прототипа приложения различными членами бригады, ориентируясь на составленную UML-диаграмму, после чего они будут объединены для пошаговой визуализации работы алгоритма.

При создании каждой из основных частей будет происходить тестирование.

| **Задача** | **Сроки выполнения** |
| --- | --- |
| Спецификация | До 3 июля |
| План разработки | До 3 июля |
| Написание прототипа | До 5 июля |
| Сдача прототипа | 5 июля |
| Реализация алгоритма Косарайю | До 6 июля |
| Объединение двух частей проекта | До 7 июля |
| Сдача 1-ой версии | 7-8 июля |
| Исправление замечаний и ошибок, реализация дополнительного функционала. Сдача 2-ой и 3-ей версии. | 10-13 июля |

Примечание:

Сдача 1-ой версии проекта планируется на 7-8 июля. После сдачи первой версии будут исправлены замечания и ошибки, а также реализован дополнительный функционал при необходимости для сдачи второй и третьей версии проекта. Дополнительные сроки и детали разработки могут быть определены в дальнейшем в зависимости от потребностей проекта и обсуждений с командой разработчиков. Также планируется написать план тестирования, который будет сдан совместно с прототипом к 5-му июля.  
 В первой версии будет реализован сам алгоритм с возможностью пошаговой визуализации его итераций или получения сразу конечного представления результата алгоритма, а также весь необходимый функционал для добавления/удаления ребер и вершин.

Во второй версии планируется добавить кнопку “Предыдущий шаг”, для отката к прошлому шага алгоритма, при этом восстанавливается вид графа на предыдущем шаге и текстовые пояснения. Также будет добавлена возможность сохранять текущее представление графа в файл. Также будет добавлена возможность сгенерировать случайный граф с помощью кнопки “Сгенерировать”, которая будет ожидать нажатие на вершину и ввода нового имени вершины в соответствующее поле. Также планируется добавить возможность сохранения текущего представления графа в виде изображения. Также будет добавлена возможность подсветки ребер, которые соединяют вершины, относящиеся к одной компоненте. Ребра которые соединяют разные компоненты, останутся черного цвета. Также будут исправлены возможные ошибки и замечания, которые были не замечены в первой версии.

## 2.2. Распределение ролей в бригаде

Таблица 1. Распределение ролей.

|  | **Тулегенова А.О** | **Сагидуллин Э.Р** | **Васильева О.М.** |
| --- | --- | --- | --- |
| 1. | Создание uml диаграммы архитектуры проекта. Написание плана разработки | Создание репозитория проекта на github. Разработка спецификации. | Реализация классов Algorithm и Graph (функционал для реализации алгоритма и структуры данных) |
| 2. | Реализация классов  Panel и FileHandler (функционал, ответственный за корректное изображение самого графа и правильное хранение) | Реализация классов Panel, MouseHandler и Vertex (функционал, ответственный за удобный и интуитивно-понятный для пользователя графический интерфейс) |
| 3. | Объединение и уточнение функционала разработанных классов в соответствии с составленной диаграммой. Связь классов, ответственных за визуализацию, с классом алгоритма с помощью класса Mediator и соответствующих кнопок на панели экрана. | |  |
| 4. | Тестирование классов  Panel и FileHandler | Тестирование классов Panel, MouseHandler и Vertex | Тестирование классов Algorithm и Graph |
| 5. | Создание дополнительного функционала для пошаговой визуализации работы алгоритма | Разработка и корректировка графического интерфейса |  |
| 6. | Тестирование графического интерфейса, пошаговой визуализации работы алгоритма. | |  |

# **3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ**

## **3.1 Основные классы**

Архитектура проекта состоит из следующих основных классов:

1. Graph

Данный класс необходим для хранения графа. В качестве способа представления графа был выбран подход на основе списка смежности. Этот подход является наиболее подходящим для реализации алгоритма Косарайю, который включает в себя обход графа в глубину, требующий быстрый доступ к инцидентным вершинам к данной. Список смежности будет реализован с помощью стандартной модифицируемой коллекции Mutable Map, где в качестве ключа будет идентификатор вершины графа, а в качестве значения модифицируемый список Mutable List, содержащий идентификаторы инцидентных вершин.

В этом классе должны быть реализованы методы для совершения следующих базовых операций на графе: добавление вершины, добавление ориентированного ребра, удаление вершины, удаление ребра, получение списка всех вершин, получение инцидентных вершин для данной вершины, получение транспонированного графа.

1. Algorithm

Данный класс реализует алгоритм Косарайю, который решает задачу поиска компонент сильной связности в ориентированном графе. Конструктор данного класса должен принимать объект класса Graph. В этом классе планируется реализовать методы выполняющие следующие операции: первоначальная обработка графа, построение транспонированного графа, выполнение обхода в глубину согласно измененному порядку вершин и определение сильно связных компонент. В качестве результата работы будет создан и возвращен объект стандартной коллекции содержащий информацию о сильно связных компонент графа. Также хранит сведения о порядке обхода вершин на транспонированном и обычном графе для визуализации алгоритма.

1. FileHandler

Данный класс отвечает за считывание графа из файла. Он содержит метод, позволяющий считывать информацию о вершинах и ребрах графа из текстового файла. Передает считанную информацию панели, для отрисовки считанного графа.

1. Mediator

Данный класс является точкой входа в программу и агрегирует работу других классов. Он содержит метод run(), который запускает выполнение программы. В классе Mediator должны создаются экземпляры классов Panel, Graph и Algorithm, а затем использоваться для считывания графа, выполнения алгоритма Косарайю и вывода пошаговой визуализации работы алгоритма.

1. MouseHandler

Данный класс отвечает за считывание графа с помощью пользовательского ввода через графический интерфейс. Он содержит методы, которые позволяют обрабатывать действия пользователя, связанные с созданием вершин, добавлением ребер и взаимодействием с графическим представлением графа, обрабатывая соответствующие события мыши и создают вершины или ребра на основе координат, полученных от пользователя. Может хранить данные о вершине или ребре с которым взаимодействует пользователь в настоящее время.

1. Panel

Класс Panel отвечает за отрисовку графического интерфейса, включая отображение вершин и ребер графа, а также взаимодействие с пользователем. Он содержит методы для отрисовки элементов интерфейса, а именно вершин и ребер графа и текстовой информации, обработки событий и обновления графического представления при изменении структуры графа в соответствии с действием пользователя. Также хранит информацию о графе в виде списков смежности вершин для изображения вершин и ребер.

1. Vertex

Данный класс представляет вершину графа и хранит информацию о ее координатах, идентификатора, а также другую необходимую информацию. Он содержит методы для получения и установки координат вершины.

1. Explanation

Enum-класс, который используется для предоставления текстовых пояснений о функционале при нажатии определенных кнопок или объяснении шагов работы алгоритма. Он содержит текстовые константы, которые представляют возможные пояснения, которые могут быть выведены на экран пользователю.

1. GraphicalInterface

Данный класс, отвечает за отрисовку графического интерфейса программы. Он выполняет следующие функции: добавление кнопок на экран; создание и размещение кнопок на панели или окне приложения; назначение обработчиков событий для кнопок, чтобы они выполняли определенные действия при нажатии; открытие окна с выбором файла.

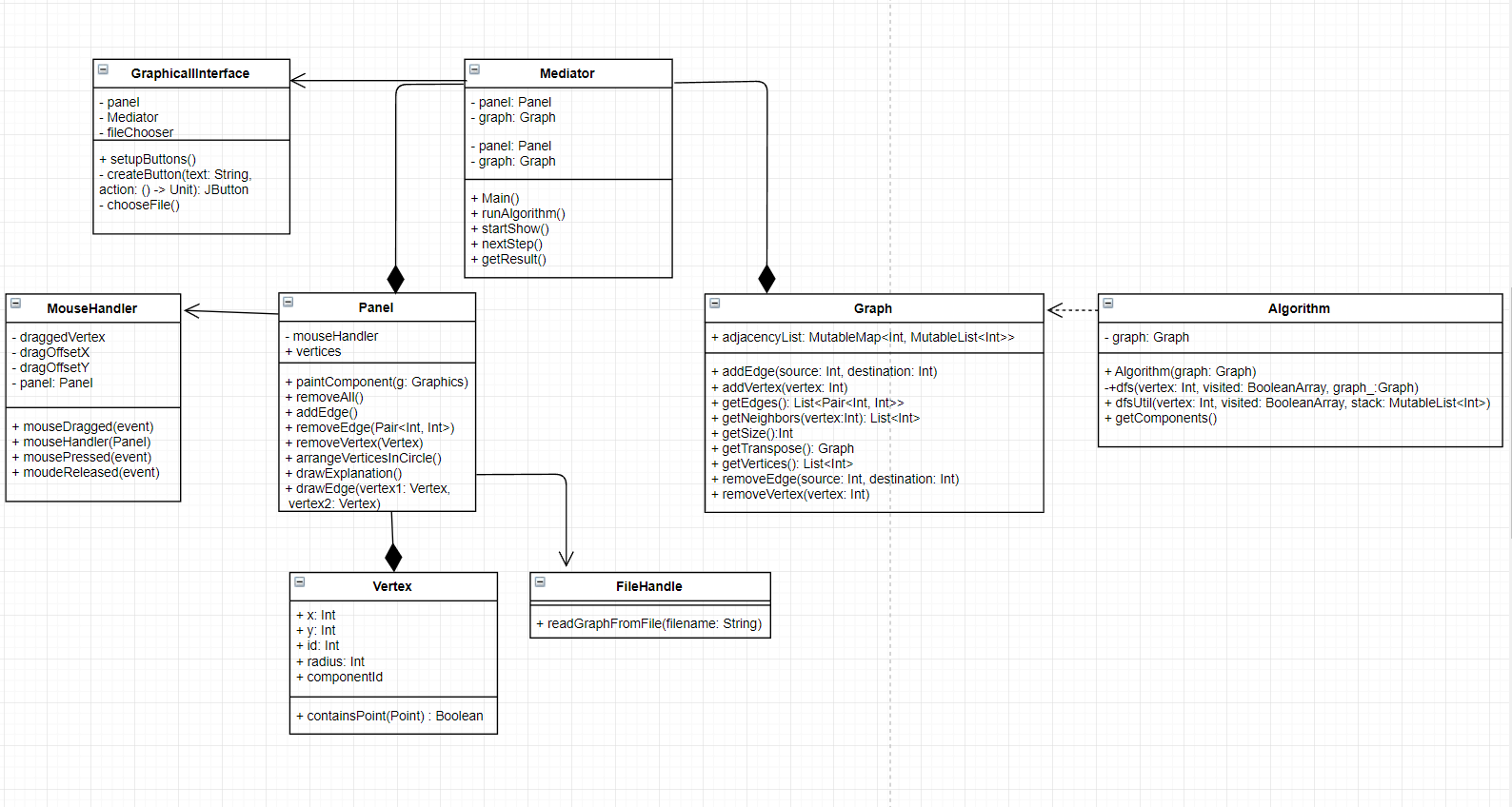


Рисунок 2. UML-диаграмма.

## 3.2. Основные методы

1. Algorithm:

* dfsUtil(vertex: Int) - приватный метод принимает вершину. Рекурсивно осуществляется обход от данной вершины, пока это возможно, и заполнение стека. Также заполняется список первого обхода ( traversalFirst).
* dfs(vertex: Int, result: MutableList<Int>, graph\_: Graph) - приватный метод принимает на вход вершину, список для результата и транспонированный граф. Метод по своей работе схож с методов dfsUtil() за исключением того, что данный метод не работает со стеком. Осуществляется обход в глубину и запись в список результатов вершин, принадлежащих текущей компоненте связности. Также заполняется список второго обхода по уже транспонированному графу (traversalSecond).
* getComponents() - главный метод, который ищет компоненты сильной связности. Заводим стек вершин и список для занесения результатов. Заполняем стек по порядку обхода графа. Затем транспонируем граф. Пока стек не пуст, проходимся по нему, извлекая вершину с конца стека и осуществляя обход в глубину по транспонированному графу. При этом заводим переменную component, которая будет хранить текущую компоненту сильной связности.

1. Graph:

* addEdge(source: Int, destination: Int) - добавление ребер. Метод на вход принимает вершину source, из которой исходит ребро, и вершину destination, в которую входит ребро, и добавляет ребро в список смежности adjacencyList.
* addVertex(vertex: Int) - добавление вершин. Метод на вход принимает индекс вершины и добавляет ее в список смежности adjacencyList.
* getTranspose(): Graph - метод транспонирует исходный граф. Создается объект типа Graph, далее осуществляется проход по всем вершинам графа и его смежным вершинам. В новый граф добавляется обратно-направленное ребро. Метод возвращает новый граф.
* getSize(): Int - возвращает количество вершин.
* removeVertex(vertex: Int) - удаление вершины из списка смежности, принимает на вход индекс вершины, и удаляет ее и все ее вхождения в списки смежности вершин.
* removeEdge(source: Int, destination: Int) - удаление ребра из списка смежности, Метод на вход принимает вершину source, из которой исходит ребро, и вершину destination, в которую входит ребро и удаляет ее из списка смежности первой вершины.

1. Vertex

* createWithId(x: Int, y: Int): Vertex - создать вершину с уникальным идентификатором с заданными координатами вершин, для задания уникального идентификатора используется переменная класса idCounter, которая увеличивается на единицу, при создании нового объекта класса.
* containsPoint(point: Point): Boolean - метод, который проверяет, кликнули ли на данную вершину и возвращает соответствующее значение.

1. MouseHandler

* override fun mousePressed(e: MouseEvent) - метод, обрабатывающий событие нажатия кнопки мыши, проверяет была ли нажата одна из вершин, если да, то делает с ней определенное действие исходя из состояния программы, если нет, то создает в этом месте новую вершину.
* override fun mouseReleased(e: MouseEvent) - метод, обрабатывающий событие прекращения нажатия на мышку. Освобождает переменную draggedVertex - ни одна вершина больше не нажата.
* override fun mouseDragged(e: MouseEvent) - метод, обрабатывающий событие перетаскивания зажатой мышки. Реализует перетаскивание вершин по полю, изменяя координаты draggedVertex в соответствии с координатами мышки, не выходя за пределы окна.

1. GraphicalInterface

* setupButtons() - установка кнопок в начале работы программы, инициализирует объекты кнопок, привязывает их к экрану и задает кнопкам действия при нажатии.
* createButton(text: String, action: () -> Unit): JButton - вспомогательный метод для создания кнопок, принимает текст кнопки и ее действие и возвращает соответствующую им кнопку, объект класса JButton.
* chooseFile() - метод, который открывает окно выбора файла, переводит содержимое файла в граф, с помощью функции readGraphFromFile объекта FileHandler, и передает его панели.
* saveGraphToFile() - метод, обрабатывающий событие нажатия на кнопку “Сохранить в файл”. Преобразует информацию о графе, хранящуюся в поле panel.vertices, и сохраняет в виде текстового файла, при этом открывается окно для задания имени файла.
* makeScreenshot() - метод, обрабатывающий событие нажатия на кнопку “Скриншот”. Сохраняет текущее окно как изображение формата png, при этом открывается окно для задания имени изображения.

1. FileHandler

* readGraphFromFile(filePath: String): MutableMap<Vertex, MutableSet<Vertex>>? - метод, который принимает на вход имя файла, считывает его содержимое и создает соответствующие ему списки смежности вершин, если файл имеет правильную структуру, иначе возвращает null.

1. Mediator

* run() - метод, создающий окно приложения и задает ему объект класса Panel(), также создает объект класса GraphicalInterface и вызывает его функцию setupButtons для установки кнопок.
* startShow() - метод, обрабатывающий событие нажатия на кнопку “Вычислить”. Переводит содержимое панели (panel.vertices) в объект класса Graph и запускает алгоритм Косарайю на полученном графе.
* nextStep() - метод, обрабатывающий событие нажатия на кнопку “Следующий шаг”. Проверяет текущее состояние программы и, исходя из этого, выбирает что нужно отобразить на текущем шаге. Если был выполнен последний шаг, то происходит вызов метода getResult().
* prevStep() - метод, обрабатывающий событие нажатия на кнопку “Предыдущий шаг”. Проверяет текущее состояние программы и, исходя из этого, выбирает что нужно отобразить на текущем шаге. При вызове этой функции при старте алгоритма, то ничего не изменится.
* getResult() - метод, обрабатывающий событие нажатия на кнопку “Результат”. Показывает конечный результат работы алгоритма Косарайю на графе.
* getVertexList(idList: List<Int>): MutableList<Vertex> - вспомогательный метод для перевода списка индексов вершин в список соответствующих им объектов класса Vertex.

1. Panel

* disableMouseListener(enable: Boolean) - метод, включающий или блокирующий обрабатывание событий нажатия на мышку, для добавления новых вершин.
* disableButtons() - метод, который для всех кнопок панели меняет возможность нажатия на данную кнопку (значение поля isEnabled) на противоположное значение.
* override fun paintComponent(g: Graphics) - метод, отвечающий за отрисовку содержимого экрана - вершин, подсветку вершин, ребер и текстовых пояснений, в зависимости от текущего состояния программы.
* drawExplanation() - вспомогательный метод, для отрисовки текстовых пояснений, с помощью функции drawString().
* drawEdge(vertex1: Vertex, vertex2: Vertex) - вспомогательный метод, для отрисовки ребер на графе, принимает объекты класса Vertex, и рисует между ними ребро с помощью функции drawLine() и drawArrow().
* drawStartVertex() - вспомогательный метод для подсвечивания стартовой вершины, при добавлении и удалении ребра.
* drawVisitedVertices() - вспомогательный метод, для подсвечивания посещенных вершин при обходе графа в глубину. Посещенные вершины хранятся в поле visited.
* drawStronglyConnectedComponents() - вспомогательный метод, для подсвечивания вершин в цвет ее компоненты сильной связности.
* componentNeighbors(vertex1: Vertex, vertex2: Vertex): Int - вспомогательный метод, который принимает на вход две вершины графа и возвращает индекс компоненты, которой они принадлежат. Если они принадлежат разным компонентам, то будет возвращено -1.
* drawVertices() - вспомогательный метод, для рисования вершин на холсте.
* drawArrow(x: Int, y: Int, angle: Double) - вспомогательный метод для рисования ориентированного ребра, принимает начальные координаты точки, через которую проходит стрелка и под каким углом нужно нарисовать стрелку, высчитывает конечные координаты точки стрелки и рисует соответствующую линию с помощью функции drawLine().
* removeAllPoints() - метод, обрабатывающий событие нажатия на кнопку “Очистить”. Удаляет все вершины на панели и соответствующие им ребра.
* addEdge() - метод, обрабатывающий событие нажатия на кнопку “Добавить ребро”. Задает соответствующее состояние объекту класса MouseHandler, для того чтобы обработать события нажатия на две вершины и добавляет вторую вершину в список смежности первой вершины.
* removeVertex() - метод, обрабатывающий событие нажатия на кнопку “Удалить вершину”. Задает соответствующее состояние объекту класса MouseHandler, для того чтобы обработать события нажатия вершину и удаляет вершину из поля vertices.
* removeEdge() - метод, обрабатывающий событие нажатия на кнопку “Удалить ребро”. Задает соответствующее состояние объекту класса MouseHandler, для того чтобы обработать события нажатия на две вершины и удаляет вторую вершину из списка смежности первой вершины.
* arrangeVerticesInCircle() - метод, который вызывается объектом класса FileHаndler, для отрисовки считанного из файла графа. Расставляет вершины графа по кругу.
* randomColor(): Color - метод, который возвращает случайный цвет.
* getVertexColor(id: Int): Color - метод. который возвращает цвет вершины по индексу ее компоненты.

# **4. ТЕСТИРОВАНИЕ**

## 4.1. План тестирования

В ходе тестирования программы необходимо рассмотреть следующие граничные случаи для проверки корректной работы программы:

Таблица 2. Граничные случаи.

| **Ошибки взаимодействия с графом** | 1. Попытка переместить вершину за пределы холста. 2. Соединение вершин, которые уже связаны ребром. 3. Удаление несуществующего ребра. 4. Попытка добавить вершину за пределами холста. 5. Ожидалось нажатие на вершину графа, но была нажата область, где нет вершины. |
| --- | --- |
| **Ошибки чтения файла** | 1. Попытка прочитать некорректный файл с неверным описанием графа. 2. Попытка прочитать пустой файл. 3. Неверный формат файла |
| **Граничные случаи для визуализации алгоритма Косарайю** | 1. Компоненты связности состоящие из одной вершины, из двух и более вершин 2. Пустой граф 3. Граф без ребер 4. Граф с изолированными вершинами 5. Граф с одной сильно связной компонентой 6. Граф с циклами и без циклов |
| **Ошибки интерфейса** | 1. Нажатие подряд на несколько кнопок, которые ожидают нажатия на вершину, нажатие на кнопку вычисления алгоритма. 2. Проблемы с отображением текстовых пояснений, которые занимают более одной строки, накладывание текста на граф. 3. Уменьшение размеров окна при добавлении вершины. |

## 4.2. Ошибки взаимодействия с графом

1. Попытка переместить вершину за пределы холста.

Попытка переместить вершину за пределы холста невозможна, в коде присутствует проверка того, что в каждый момент координаты центра вершин должны быть меньше чем границы поля, но больше нуля, с учетом радиуса вершины.

1. Соединение вершин, которые уже связаны ребром.

При соединении вершин связанных ребром ничего не произойдет. Так как список смежности каждой вершины задан с помощью стандартной коллекцией Set (множество), то при добавлении в него инцидентной вершины, которая уже входит во множество, между данными вершинами по-прежнему останется только одно исходное ребро ( не мультиграф).

1. Удаление несуществующего ребра.

При попытке удаления несуществующего ребра граф не изменится. Так как вершина, с которой удаляется ребро не входит во множество инцидентных вершин для данной, то граф сохраняет исходное состояние.

1. Попытка создать вершину за пределами холста.

Невозможно создать вершину за пределами холста, так как вершина создается с помощью нажатия левой кнопки мыши по холсту.

1. Нажатие в точки холста не принадлежащие вершинам.

В случае, если ожидалось нажатие на вершину графа, но была нажата область, не принадлежащая окрестности ни одной из вершин, в этой точке будет создана вершина, программа выйдет из состояния, ожидающего нажатия на вершину.

## 4.3. Граничные случаи для визуализации алгоритма Косарайю

1. Компоненты связности состоящие из одной, из двух и более вершин:

Алгоритм корректно обрабатывает случаи с любым количеством вершин принадлежащих различным компонентам сильной связности, а также различную связь между компонентами сильной связности (связные и несвязные ребрами компоненты):



Рисунок 3. Компонента связности из множества вершин.

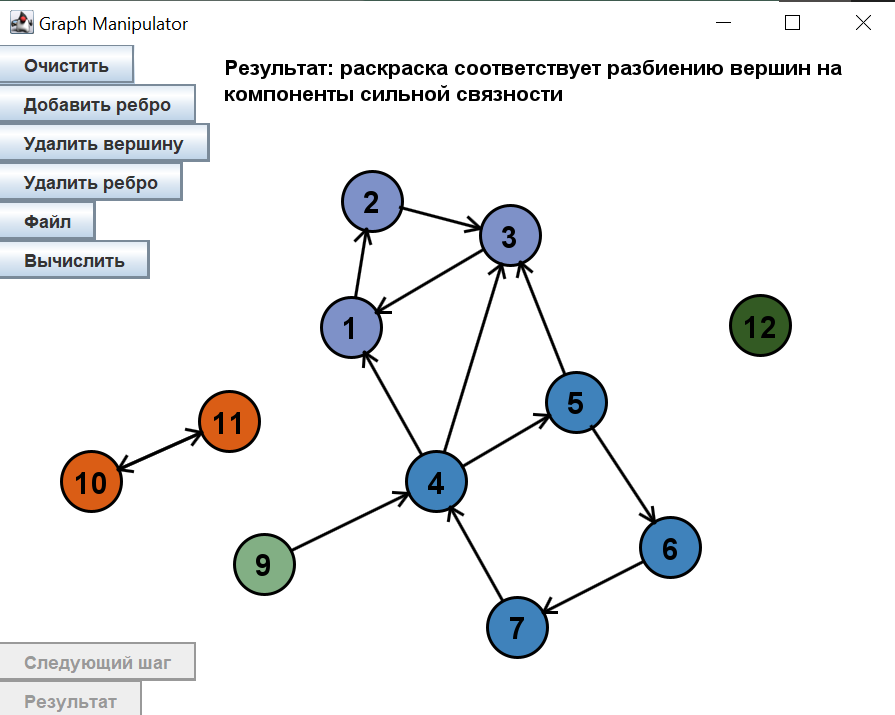


Рисунок 4. Различная связь между компонентами связности.

1. Пустой граф:

Приложение не допускает запуска алгоритма на пустом графе, кнопка “Вычислить” будет заблокирована, пока не будет добавлена хоть одна вершина.

1. Граф без ребер:

Алгоритм корректно обрабатывает ситуацию, когда в графе отсутствуют ребра, в результате каждой вершине соответствует своя компонента сильной связности:



Рисунок 5. Тестирование алгоритма на графе без ребер.

1. Граф с изолированными вершинами:

Алгоритм также корректно обрабатывает ситуацию, когда только некоторые вершины графа изолированы:

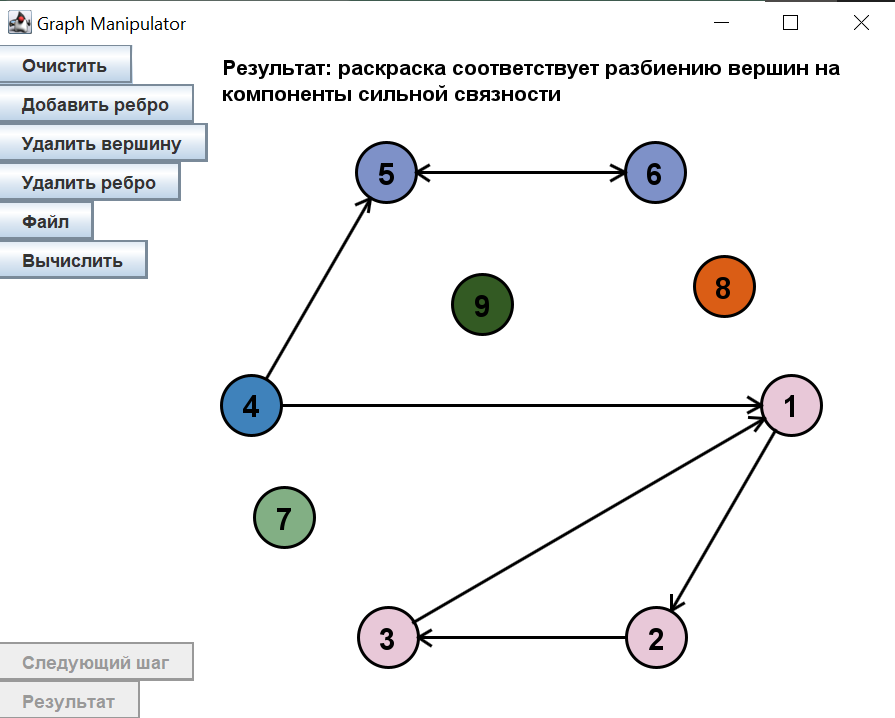


Рисунок 6. Тестирование алгоритма на графе с изолированными вершинами.

1. Граф с одной сильно связной компонентой:

Алгоритм корректно обрабатывает ситуацию, когда граф имеет одну компоненту сильной связности:



Рисунок 7. Тестирование алгоритма на графе с одной сильно связной компонентой.

1. Граф с циклами и без циклов:

Как и ожидалось, в ситуации когда граф не имеет циклов, каждой вершине соответствует своя компонента сильной связности:

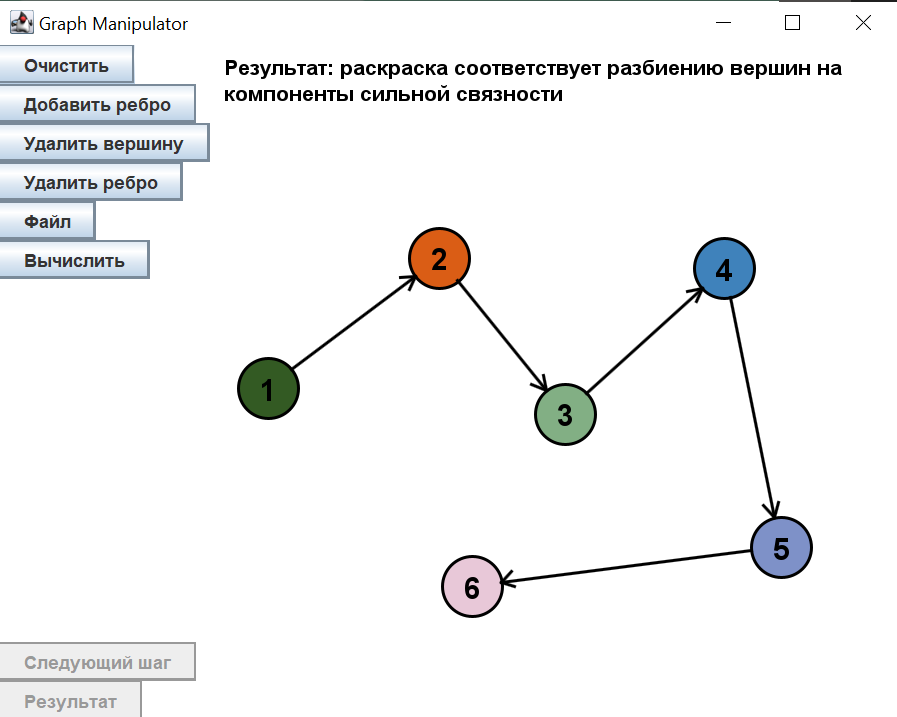


Рисунок 8. Тестирование алгоритма на графе без циклов.

В ситуации с циклами, каждому циклу соответствует своя компонента сильной связности:

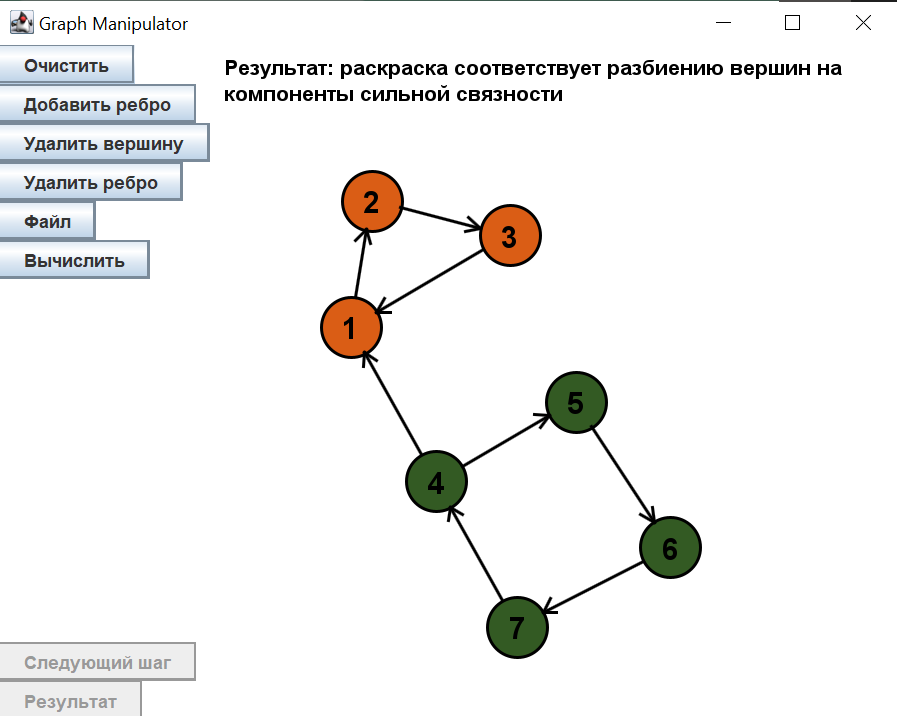


Рисунок 9. Тестирование алгоритма на графе с циклами.

## 4.4. Ошибки чтения файла

1. Попытка прочитать некорректный файл с неверным описанием графа.

Если описание графа задано неверно, то выводится соответствующая надпись:

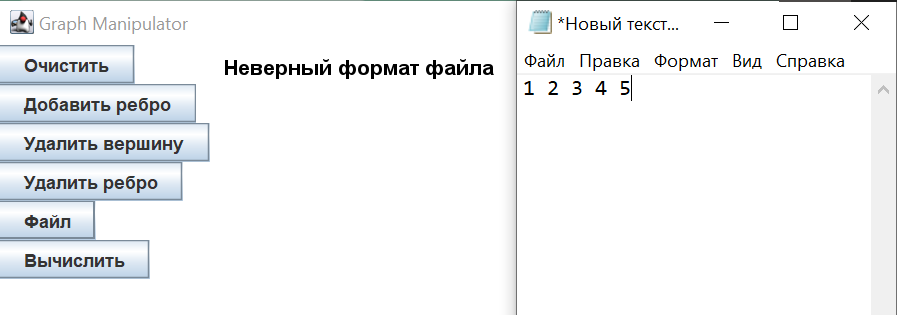


Рисунок 10. Неверное описание графа (более двух индексов в строке).

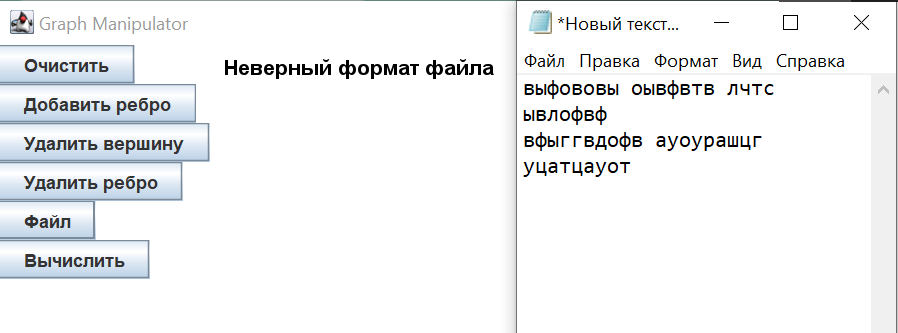


Рисунок 11. Неверное описание графа (больше двух индексов в строке, индексы не являются числами).

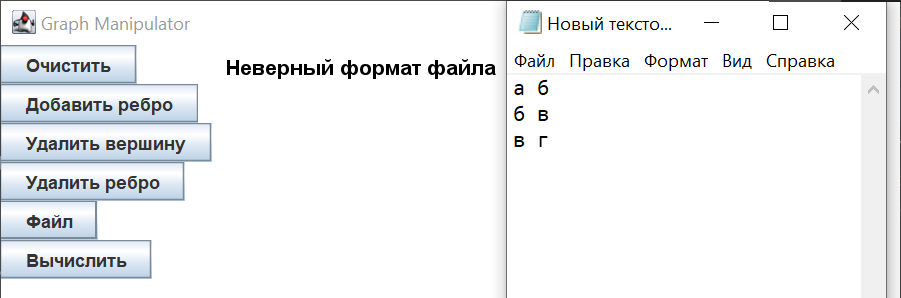


Рисунок 12. Неверное описание графа (индексы не являются числами).

1. Попытка прочитать пустой файл.

При попытке прочитать пустой файл, выводится пустой граф:

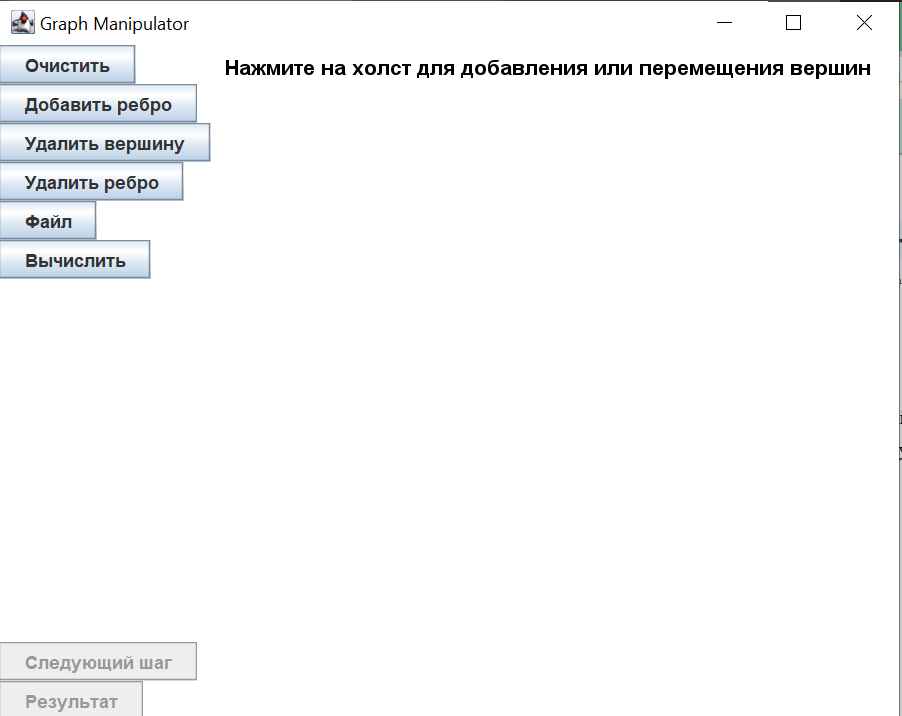
****

Рисунок 13. Граф задает пустой файл.

1. Неверный формат файла

Невозможно выбрать файл не txt-формата, так как окно выбора файла отображает только txt-файла.

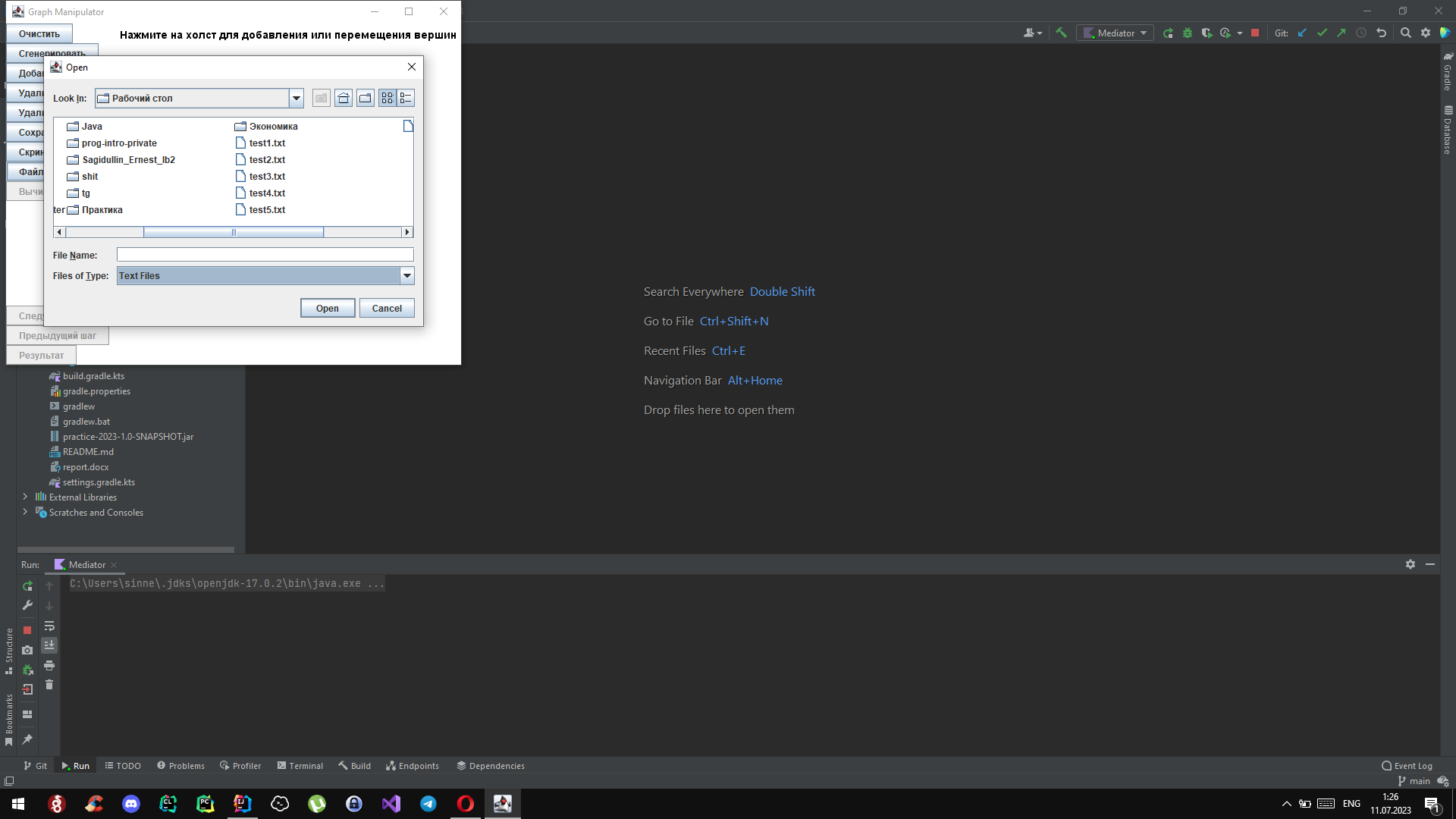


Рисунок 14. Окно выбора файла.

## 4.5. Ошибки интерфейса

1. Нажатие подряд на несколько кнопок, которые ожидают какое-либо действие, нажатие на кнопку вычисления алгоритма.

Каждое нажатие на кнопку, которая ожидает действия, или нажатие на кнопку вычисления алгоритма и открытия файла, переводит программу в новое состояние и за ним следует выход из предыдущего состояния. Поэтому ситуация с нажатием подряд на несколько кнопок, которые ожидают нажатия на вершину корректно обрабатывается программой.

1. Проблемы с отображением текстовых пояснений, которые занимают более одной строки, накладывание текста на граф.

В случае если длина текстовых пояснений превышает размеры окна, то текст будет разделен на несколько строк. Текст может накладываться на вершины, однако это никак не скажется на возможности кликать на вершины, перетаскивать вершины, добавлять/удалять ребра между вершинами,

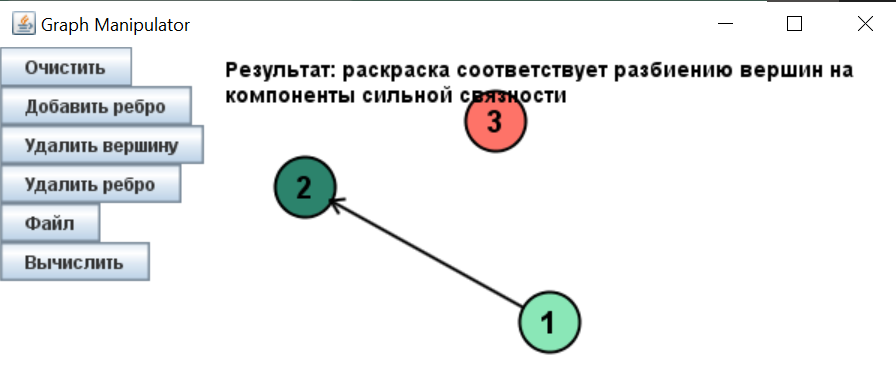


Рисунок 15. Перенос текстовых пояснений, накладывание текста на граф.

1. Уменьшение размеров окна при добавлении вершины.

В данном случае вершина не изменит своих координат и не удалится, при увеличении размеров окна вершина вновь будет изображена.

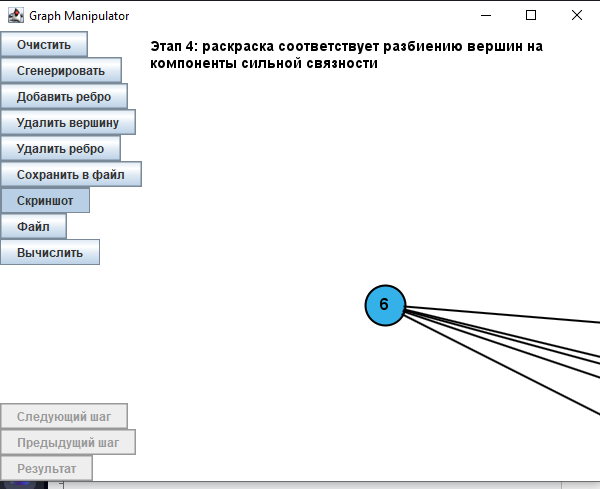


Рисунок 16. Свернутое окно меньшего размера.

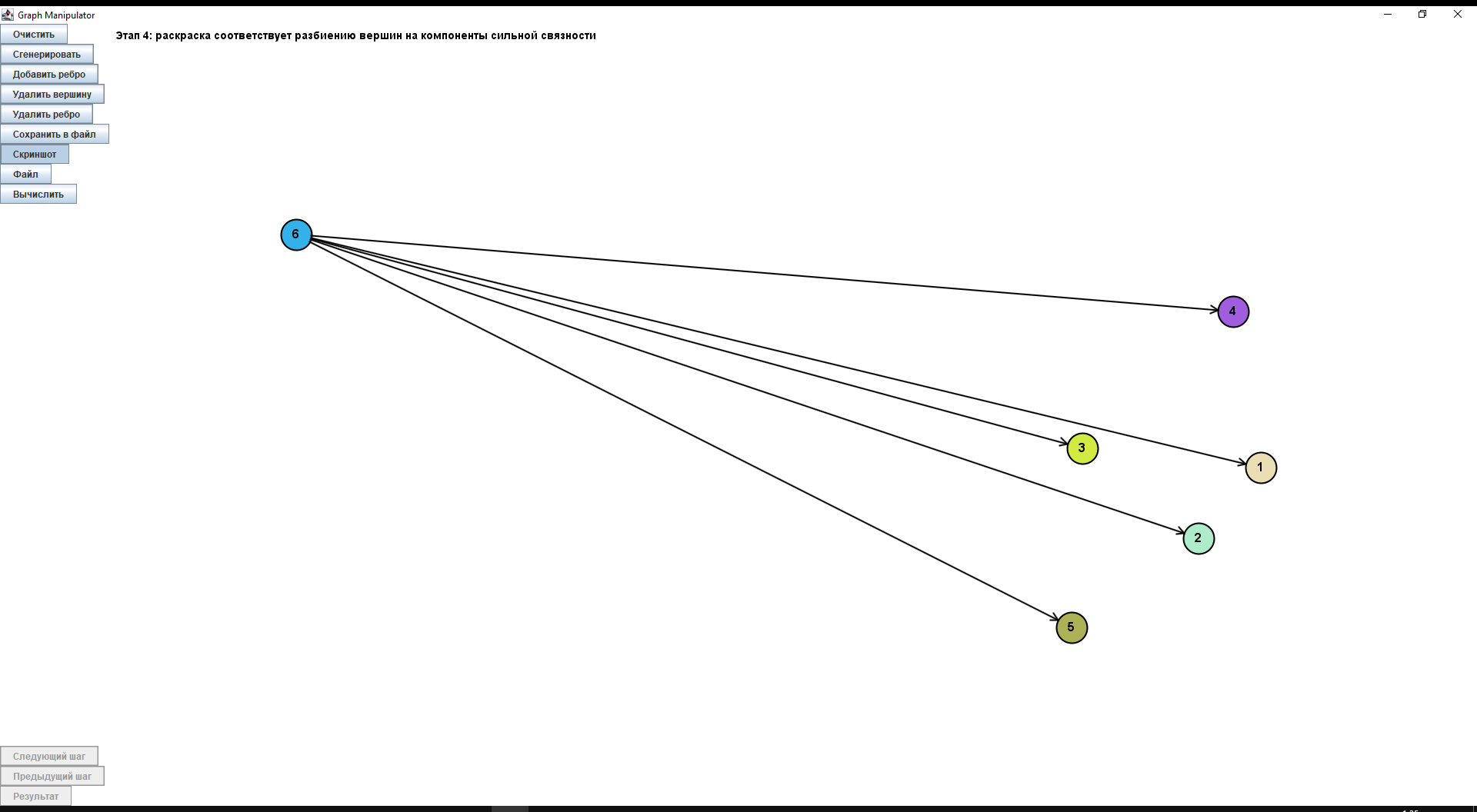


Рисунок 17. Развернутое окно сохраняющее элементы.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы, все поставленные цели для разработки оконного приложения на Kotlin с использованием библиотеки Swing для визуализации алгоритма Косарайю были успешно выполнены.

Приложение предоставляет удобный пользовательский интерфейс с различными функциями, которые позволяют добавлять и удалять вершины и ребра, перемещать вершины по полю, считывать графы из файлов и сохранять их в файлы, делать скриншоты окна приложения, генерировать случайный граф, а также визуализировать работу алгоритма Косарайю пошагово или выводить конечный результат. Приложение также выводит текстовые пояснения, что позволяет пользователям делает интерфейс понятным и удобным для работы с графом.

Также было проведено тестирование крайних случаев. Разработка приложения происходила согласно плану разработки. Итоговое приложение предоставляет все необходимые функции для работы с графами, а также позволяет пошагово визуализировать, понять и изучить работу алгоритма Косарайю.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ZetCode - Go, C#, Python, Java, JavaScript programming. <https://zetcode.com/kotlin/swing/>

2. Официальный сайт языка Kotlin. <https://kotlinlang.org/>

3. Хабр — русскоязычный веб-сайт в формате системы тематических коллективных блогов. <https://habr.com/ru/articles/305974/>

1. Хабр — русскоязычный веб-сайт в формате системы тематических коллективных блогов. https://habr.com/ru/articles/537290/

# **ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

class Algorithm(private val graph: Graph) {

var traversalFirst: MutableList<Int> = mutableListOf()

val traversalSecond: MutableList<Int> = mutableListOf()

var exitOrder: MutableList<Int> = mutableListOf()

private lateinit var stack: MutableList<Int>

private fun dfsUtil(vertex: Int) {

traversalFirst.add(vertex)

for (neighbor in graph.getNeighbors(vertex)) {

if (neighbor !in traversalFirst) {

dfsUtil(neighbor)

}

}

stack.add(vertex)

}

private fun dfs(vertex: Int, result: MutableList<Int>, graph\_: Graph) {

traversalSecond.add(vertex)

result.add(vertex)

for (neighbor in graph\_.getNeighbors(vertex)) {

if (neighbor !in traversalSecond) {

dfs(neighbor, result, graph\_)

}

}

}

fun getComponents(): List<List<Int>> {

stack = mutableListOf()

val result: MutableList<List<Int>> = mutableListOf()

for (vertex in graph.getVertices()) {

if (vertex !in traversalFirst) {

dfsUtil(vertex)

}

}

val transposedGraph = graph.getTranspose()

exitOrder = stack.toMutableList()

while (stack.isNotEmpty()) {

val vertex = stack.removeAt(stack.size - 1)

if (vertex !in traversalSecond) {

val component = mutableListOf<Int>()

dfs(vertex, component, transposedGraph)

result.add(component)

}

}

return result

}

}

enum class Explanations(val text: String) {

NOTEXT("Нажмите на холст для добавления или перемещения вершин"),

CHOOSEVERTEX1("Нажмите на вершину, из которой ведет ребро"),

CHOOSEVERTEX2("Нажмите на вершину, в которую ведет ребро"),

DELVERTEX("Нажмите на вершину, которую следует удалить"),

FILEERROR("Файл задан неверно"),

TRANSPOSEGRAPH("Этап 2: Транспонировали граф для повторного обхода графа, указали на вершинах номер выхода"),

DFSONTRANSPOSE("Этап 3: Поиск в глубину на транспонированном графе, выбирая не посещённую вершину с максимальным номером, каждое дерево обхода - компонента сильной связности"),

DFS("Этап 1: Поиск в глубин, запоминая, в каком порядке выходили из вершин."),

RESULT("Этап 4: раскраска соответствует разбиению вершин на компоненты сильной связности"),

START("Старт: Визуализация алгоритма Косарайю"),

INVALIDFILEFORMAT("Неверный формат файла")

}

import java.io.File

object FileHandler {

fun readGraphFromFile(filePath: String): MutableMap<Vertex, MutableSet<Vertex>>? {

val vertices = mutableMapOf<Vertex, MutableSet<Vertex>>()

val verticesMap = mutableMapOf<Int, Vertex>()

val file = File(filePath)

Vertex.idCounter = 1

file.bufferedReader().useLines { lines ->

lines.forEach { line ->

val vertexId = line.split(" ").map {

try {

it.toInt()

} catch (e: NumberFormatException) {

return null

}

}

if (vertexId.size == 2 && vertexId[0] != vertexId[1]) {

verticesMap.getOrPut(vertexId[0]) { Vertex.createWithId(0, 0) }

verticesMap.getOrPut(vertexId[1]) { Vertex.createWithId(0, 0) }

val vertex1 = verticesMap[vertexId[0]]!!

val vertex2 = verticesMap[vertexId[1]]!!

vertices.getOrPut(vertex1) { mutableSetOf() }.add(vertex2)

vertices.getOrPut(vertex2) { mutableSetOf() }

} else if (vertexId.size == 1) {

verticesMap.getOrPut(vertexId[0]) { Vertex.createWithId(0, 0) }

val vertex1 = verticesMap[vertexId[0]]!!

vertices.getOrPut(vertex1) { mutableSetOf() }

} else

return null

}

}

return vertices

}

}

import java.io.File

import kotlin.random.Random

class Graph {

private val adjacencyList: MutableMap<Int, MutableList<Int>> = mutableMapOf()

fun addVertex(vertex: Int) {

adjacencyList.getOrPut(vertex) { mutableListOf() }

}

fun addEdge(source: Int, destination: Int) {

adjacencyList.getOrPut(source) { mutableListOf() }.add(destination)

adjacencyList.getOrPut(destination) { mutableListOf() }

}

fun removeVertex(vertex: Int) {

adjacencyList.remove(vertex)

adjacencyList.values.forEach { it.remove(vertex) }

}

fun removeEdge(source: Int, destination: Int) {

adjacencyList[source]?.remove(destination)

}

fun getVertices(): Set<Int> {

return adjacencyList.keys

}

fun getNeighbors(vertex: Int): List<Int> {

return adjacencyList[vertex] ?: emptyList()

}

fun getTranspose(): Graph {

val transposedGraph = Graph()

for (vertex in adjacencyList.keys) {

for (neighbor in adjacencyList[vertex]!!) {

transposedGraph.addEdge(neighbor, vertex)

}

}

return transposedGraph

}

fun getSize(): Int {

return adjacencyList.size

}

}

object GraphGenerator {

private const val leftBorder = 20

private const val rightBorder = 31

private fun generateRandomList(): List<Pair<Int, Int>> {

val n = Random.nextInt(leftBorder, rightBorder)

val g = Random.nextInt(0, 3\*n)

val list = mutableListOf<Pair<Int, Int>>()

for (i in 1..g) {

val m = Random.nextInt(0, n)

var k = Random.nextInt(0, n)

while (k == m) {

k = Random.nextInt(0, n)

}

val pair = Pair(m, k)

list.add(pair)

}

return list

}

fun generateGraph(): MutableMap<Vertex, MutableSet<Vertex>> {

val edgesList = generateRandomList()

val verticesMap = mutableMapOf<Int, Vertex>()

val vertices = mutableMapOf<Vertex, MutableSet<Vertex>>()

edgesList.forEach { (source, destination)->

verticesMap.getOrPut(source) { Vertex.createWithId(0, 0) }

verticesMap.getOrPut(destination) { Vertex.createWithId(0, 0) }

val vertex1 = verticesMap[source]!!

val vertex2 = verticesMap[destination]!!

vertices.getOrPut(vertex1) { mutableSetOf() }.add(vertex2)

vertices.getOrPut(vertex2) { mutableSetOf() }

}

return vertices

}

}

import java.awt.GraphicsEnvironment

import java.awt.Rectangle

import java.awt.Robot

import java.io.BufferedWriter

import java.io.File

import java.io.FileWriter

import javax.imageio.ImageIO

import javax.swing.Box

import javax.swing.JButton

import javax.swing.JFileChooser

import javax.swing.SwingUtilities

import javax.swing.filechooser.FileNameExtensionFilter

class GraphicalInterface(private val panel: Panel,

private val mediator: Mediator) {

private val fileChooser = JFileChooser()

fun setupButtons() {

val buttonsData = listOf(

ButtonData("Очистить") { panel.removeAllPoints() },

ButtonData("Сгенерировать") { panel.generateGraph() },

ButtonData("Добавить ребро") { panel.addEdge() },

ButtonData("Удалить вершину") { panel.removeVertex() },

ButtonData("Удалить ребро") { panel.removeEdge() },

ButtonData("Сохранить в файл") { saveGraphToFile() },

ButtonData("Скриншот") { makeScreenshot() },

ButtonData("Файл") { chooseFile() },

)

buttonsData.forEach { buttonData ->

val button = JButton(buttonData.text).apply {

isFocusPainted = false

addActionListener { buttonData.action() }

}

panel.add(button)

}

val nextStepButton = createButton("Следующий шаг") { mediator.nextStep() }

val prevStepButton = createButton("Предыдущий шаг") { mediator.prevStep() }

val startAlgorithmButton = createButton("Вычислить") {

panel.repaint()

panel.disableButtons()

panel.disableMouseListener(false)

mediator.startShow()

}

val resultButton = createButton("Результат") {

mediator.getResult()

panel.disableButtons()

panel.disableMouseListener(true)

}

panel.startButton = startAlgorithmButton

panel.add(startAlgorithmButton)

panel.add(Box.createVerticalGlue())

panel.add(nextStepButton)

panel.add(prevStepButton)

panel.add(resultButton)

fileChooser.currentDirectory = java.io.File(".")

fileChooser.fileSelectionMode = JFileChooser.FILES\_ONLY

}

private fun createButton(text: String, action: () -> Unit): JButton {

return JButton(text).apply {

isFocusPainted = false

isEnabled = false

addActionListener { action() }

}

}

private fun chooseFile() {

val fileFilter = FileNameExtensionFilter("Text Files", "txt")

fileChooser.fileFilter = fileFilter

val returnValue = fileChooser.showOpenDialog(panel)

if (returnValue == JFileChooser.APPROVE\_OPTION) {

val selectedFile = fileChooser.selectedFile

val fileName = selectedFile.name

val fileExtension = fileName.substringAfterLast(".", "")

if (fileExtension == "txt") {

val pathFile = selectedFile.absolutePath

val result = FileHandler.readGraphFromFile(pathFile)

if (result != null) {

panel.vertices = result

panel.arrangeVerticesInCircle()

} else {

panel.explanation = Explanations.FILEERROR

panel.removeAllPoints()

panel.repaint()

}

} else {

panel.explanation = Explanations.INVALIDFILEFORMAT

panel.removeAllPoints()

panel.repaint()

}

}

}

private fun saveGraphToFile() {

val fileChooser = JFileChooser()

fileChooser.fileSelectionMode = JFileChooser.FILES\_ONLY

fileChooser.fileFilter = FileNameExtensionFilter("Text Files (\*.txt)", "txt")

val result = fileChooser.showSaveDialog(null)

if (result == JFileChooser.APPROVE\_OPTION) {

val selectedFile = fileChooser.selectedFile

var filePath = selectedFile.absolutePath

if (!filePath.endsWith(".txt"))

filePath += ".txt"

try {

val writer = BufferedWriter(FileWriter(filePath))

for ((source, targets) in panel.vertices) {

for (target in targets)

writer.write("${source.id} ${target.id}\n")

if (targets.isEmpty()) writer.write("${source.id}\n")

}

writer.close()

} catch (e: Exception) {

e.printStackTrace()

}

}

}

private fun makeScreenshot() {

try {

val window = SwingUtilities.getWindowAncestor(panel)

val graphicsDevice = GraphicsEnvironment.getLocalGraphicsEnvironment().defaultScreenDevice

val screenshot = Robot(graphicsDevice).createScreenCapture(

Rectangle(window.locationOnScreen.x + 8, window.locationOnScreen.y,

window.width - 16, window.height))

val fileChooser = JFileChooser()

fileChooser.fileSelectionMode = JFileChooser.FILES\_ONLY

fileChooser.dialogTitle = "Save Screenshot"

fileChooser.fileFilter = FileNameExtensionFilter("PNG Images", "png")

val returnValue = fileChooser.showSaveDialog(null)

if (returnValue == JFileChooser.APPROVE\_OPTION) {

val selectedFile = fileChooser.selectedFile

var filePath = selectedFile.absolutePath

if (!filePath.endsWith(".png"))

filePath += ".png"

ImageIO.write(screenshot, "png", File(filePath))

}

} catch (ex: Exception) {

ex.printStackTrace()

}

}

}

data class ButtonData(val text: String, val action: () -> Unit)

import java.awt.Dimension

import javax.swing.BoxLayout

import javax.swing.JFrame

class Mediator {

lateinit var panel: Panel

private lateinit var componentsList: List<List<Int>>

private lateinit var algorithm: Algorithm

private var graph = Graph()

private val vertexMap: MutableMap<Int, Vertex> = mutableMapOf()

private var currentState: State = State.NONE

enum class State {

NONE,

DFS1,

DFS2

}

fun run() {

val frame = JFrame("Graph Manipulator")

frame.defaultCloseOperation = JFrame.EXIT\_ON\_CLOSE

frame.contentPane.preferredSize = Dimension(600, 450)

frame.minimumSize = Dimension(600, 450)

panel = Panel()

panel.layout = BoxLayout(panel, BoxLayout.Y\_AXIS)

frame.contentPane.add(panel)

val graphicalInterface = GraphicalInterface(panel, this)

graphicalInterface.setupButtons()

frame.pack()

frame.isVisible = true

}

fun startShow(){

panel.sccList = mutableListOf()

panel.vertices.keys.forEach { key -> key.bypassNumber = null }

panel.explanation = Explanations.START

panel.vertices.forEach{(k, v) -> vertexMap[k.id!!] = k}

panel.vertices.forEach { (k, v) ->

v.forEach { n ->

graph.addEdge(k.id!!, n.id!!)

}

if (v.isEmpty()) graph.addVertex(k.id!!)

}

algorithm = Algorithm(graph)

componentsList = algorithm.getComponents()

}

fun nextStep(){

when (currentState) {

State.NONE -> {

currentState = State.DFS1

nextStep()

}

State.DFS1 -> {

panel.explanation = Explanations.DFS

val visitedCount = panel.visited.size

if (visitedCount < algorithm.traversalFirst.size) {

panel.visited = getVertexList(algorithm.traversalFirst.subList(0, visitedCount + 1))

//panel.visited.last().bypassNumber = visitedCount + 1

} else {

panel.vertices = panel.transposeGraph(panel.vertices)

currentState = State.DFS2

panel.visited = mutableListOf()

panel.explanation = Explanations.TRANSPOSEGRAPH

val exitOrder = getVertexList(algorithm.exitOrder)

for (i in 0 until exitOrder.size)

exitOrder[i].bypassNumber = i + 1

}

panel.repaint()

}

State.DFS2 -> {

panel.explanation = Explanations.DFSONTRANSPOSE

val visitedCount = panel.visited.size

if (visitedCount < algorithm.traversalSecond.size) {

val visited = algorithm.traversalSecond.subList(0, visitedCount + 1)

checkComponents(visited)

} else {

panel.vertices = panel.transposeGraph(panel.vertices)

currentState = State.NONE

panel.visited = mutableListOf()

panel.disableButtons()

panel.disableMouseListener(true)

getResult()

}

panel.repaint()

}

}

}

fun prevStep() {

when (currentState) {

State.NONE -> {}

State.DFS1 -> {

panel.explanation = Explanations.DFS

val visitedCount = panel.visited.size

if (visitedCount != 0) {

panel.visited = getVertexList(algorithm.traversalFirst.subList(0, visitedCount - 1))

if (visitedCount == 1) panel.explanation = Explanations.START

} else {

currentState = State.NONE

panel.explanation = Explanations.START

panel.visited = mutableListOf()

}

panel.repaint()

}

State.DFS2 -> {

panel.explanation = Explanations.DFSONTRANSPOSE

val visitedCount = panel.visited.size

if (visitedCount != 0) {

val visited = algorithm.traversalSecond.subList(0, visitedCount - 1)

checkComponents(visited)

if (visitedCount == 1) panel.explanation = Explanations.TRANSPOSEGRAPH

} else {

currentState = State.DFS1

panel.vertices = panel.transposeGraph(panel.vertices)

panel.explanation = Explanations.DFS

panel.visited = getVertexList(algorithm.traversalFirst)

}

panel.repaint()

}

}

}

private fun checkComponents(visited: MutableList<Int>) {

panel.visited = getVertexList(visited)

val currentComponents = mutableListOf<MutableList<Vertex>>()

for (component in componentsList) {

if (visited.containsAll(component))

currentComponents.add(getVertexList(component))

}

panel.sccList = currentComponents

}

fun getResult() {

panel.explanation = Explanations.RESULT

panel.vertices.keys.forEach { key -> key.bypassNumber = null }

if (currentState == State.DFS2) {

panel.vertices = panel.transposeGraph(panel.vertices)

currentState = State.NONE

}

val components = mutableListOf<MutableList<Vertex>>()

for (component in componentsList) {

components.add(getVertexList(component))

}

panel.sccList = components

panel.repaint()

panel.visited = mutableListOf()

graph = Graph()

}

private fun getVertexList(idList: List<Int>): MutableList<Vertex> {

val vertices = mutableListOf<Vertex>()

for (id in idList)

vertexMap[id]?.let { vertices.add(it) }

return vertices

}

companion object {

@JvmStatic

fun main(args: Array<String>) {

Mediator().run()

}

}

}

import java.awt.event.MouseAdapter

import java.awt.event.MouseEvent

class MouseHandler(private val panel: Panel) : MouseAdapter() {

private var draggedVertex: Vertex? = null

private var dragOffsetX = 0

private var dragOffsetY = 0

var startVertex: Vertex? = null

var isDrawingEdge: Boolean = false

var isRemovingVertex: Boolean = false

var isRemovingEdge: Boolean = false

var addingVertex: Boolean = true

override fun mousePressed(e: MouseEvent) {

val clickedPoint = e.point

var vertexClicked = false

if (e.button == MouseEvent.BUTTON3 && addingVertex)

isDrawingEdge = true

for (vertex in panel.vertices.keys) {

if (vertex.containsPoint(clickedPoint)) {

draggedVertex = vertex

dragOffsetX = clickedPoint.x - vertex.x

dragOffsetY = clickedPoint.y - vertex.y

vertexClicked = true

break

}

}

if (!vertexClicked && e.x >= 0 && e.y >= 0 && addingVertex) {

panel.vertices[Vertex.createWithId(e.x, e.y)] = mutableSetOf()

panel.explanation = Explanations.NOTEXT

resetFlags()

panel.repaint()

}

if ((isDrawingEdge || isRemovingEdge) && startVertex == null && vertexClicked) {

startVertex = draggedVertex

} else if (isDrawingEdge && startVertex != null && vertexClicked && startVertex != draggedVertex) {

panel.vertices[startVertex]?.add(draggedVertex!!)

resetFlags()

} else if (isRemovingEdge && startVertex != null && vertexClicked && startVertex != draggedVertex) {

panel.vertices[startVertex]?.remove(draggedVertex!!)

resetFlags()

} else if (isRemovingVertex && vertexClicked) {

panel.vertices.remove(draggedVertex)

panel.vertices.forEach { (vert, adjacencyList) ->

if (adjacencyList.contains(draggedVertex))

panel.vertices[vert]?.remove(draggedVertex)

}

resetFlags()

}

panel.repaint()

}

override fun mouseReleased(e: MouseEvent) {

draggedVertex = null

}

override fun mouseDragged(e: MouseEvent) {

if (e.x - dragOffsetX >= 0 && e.y - dragOffsetY >= 0 &&

e.x - dragOffsetX < panel.width && e.y - dragOffsetY < panel.height)

draggedVertex?.let {

it.x = e.x - dragOffsetX

it.y = e.y - dragOffsetY

panel.repaint()

}

}

private fun resetFlags() {

startVertex = null

isDrawingEdge = false

isRemovingVertex = false

isRemovingEdge = false

}

}

import java.awt.\*

import java.awt.event.KeyAdapter

import java.awt.event.KeyEvent

import java.util.\*

import javax.swing.JButton

import javax.swing.JPanel

import kotlin.math.\*

class Panel : JPanel() {

private lateinit var g2d: Graphics2D

private val mouseHandler = MouseHandler(this)

private val fontStyle = Font("Arial ", Font.BOLD, 16)

var explanation: Explanations = Explanations.NOTEXT

var startButton: JButton? = null

var vertices: MutableMap<Vertex, MutableSet<Vertex>> = mutableMapOf()

var sccList: MutableList<MutableList<Vertex>> = mutableListOf()

var sccColorList: MutableMap<Int, Color> = mutableMapOf()

var visited: MutableList<Vertex> = mutableListOf()

init {

addMouseListener(mouseHandler)

addMouseMotionListener(mouseHandler)

disableMouseListener(true)

background = Color.WHITE

layout = FlowLayout()

}

companion object {

private const val ARROW\_SIZE = 10

private const val LINE\_WIDTH = 2f

private val VERTEX\_COLOR = Color.BLACK

private val START\_VERTEX\_COLOR = Color.PINK

private val TEXT\_COLOR = Color.BLACK

}

fun disableMouseListener(enable: Boolean) {

mouseHandler.addingVertex = enable

}

fun disableButtons() {

for (component in components)

if (component is JButton)

component.isEnabled = !component.isEnabled

}

private fun drawStartVertex(){

mouseHandler.startVertex?.let { startVertex ->

explanation = Explanations.CHOOSEVERTEX2

val halfRad = startVertex.radius / 2

g2d.color = START\_VERTEX\_COLOR

g2d.fillOval(startVertex.x - halfRad, startVertex.y - halfRad, startVertex.radius, startVertex.radius)

}

}

private fun drawVisitedVertices(){

with(g2d){

val sumOfLengths = sccList.sumOf { it.size }

if (sumOfLengths <= vertices.size && explanation in listOf(Explanations.DELVERTEX,

Explanations.CHOOSEVERTEX2, Explanations.CHOOSEVERTEX1, Explanations.NOTEXT)) {

sccList = mutableListOf()

vertices.keys.forEach { key -> key.bypassNumber = null }

}

if (sccList.size >= sccColorList.size) {

for (i in sccColorList.size..sccList.size)

color = getVertexColor(i)

}

color = sccColorList[sccList.size]

for(vertex in visited){

val halfRad = vertex.radius / 2

fillOval(vertex.x - halfRad, vertex.y - halfRad, vertex.radius, vertex.radius)

}

}

}

private fun drawStronglyConnectedComponents(){

with(g2d){

for(id in 0 until sccList.size) {

color = sccColorList[id]

sccList[id].forEach{ vertex ->

val halfRad = vertex.radius / 2

fillOval(vertex.x - halfRad, vertex.y - halfRad, vertex.radius, vertex.radius)

}

}

}

}

private fun componentNeighbors(vertex1: Vertex, vertex2: Vertex): Int {

for (id in 0 until sccList.size) {

if (sccList[id].contains(vertex1) && sccList[id].contains(vertex2)){

return id

}

}

return -1

}

private fun drawVertices() {

vertices.forEach { (vertex, adjacencyList) ->

if (vertex.x >= 0 && vertex.y >= 0) {

with(g2d) {

stroke = BasicStroke(LINE\_WIDTH)

color = VERTEX\_COLOR

font = fontStyle

val halfRad = vertex.radius / 2

drawOval(vertex.x - halfRad, vertex.y - halfRad, vertex.radius, vertex.radius)

var letter = vertex.id.toString()

if (vertex.bypassNumber != null)

letter = letter + "|" + vertex.bypassNumber.toString()

val fm = fontMetrics

val letterWidth = fm.stringWidth(letter)

val letterHeight = fm.height

val letterX = vertex.x - halfRad + (vertex.radius - letterWidth) / 2

val letterY = vertex.y - halfRad / 4 \* 5 + (vertex.radius + letterHeight) / 2

color = TEXT\_COLOR

drawString(letter, letterX, letterY)

adjacencyList.forEach { adjacencyVertex ->

if (explanation == Explanations.RESULT){

val componentId = componentNeighbors(vertex, adjacencyVertex)

if (componentId > -1 ){

color = sccColorList[componentId]

}

}

drawEdge(vertex, adjacencyVertex)

color = VERTEX\_COLOR

}

}

}

}

if (explanation in listOf(Explanations.DFS, Explanations.DFSONTRANSPOSE)) {

if (visited.size >= 2) {

val currVertex = visited.last()

for (i in visited.indices.reversed()) {

if (vertices[visited[i]]?.contains(currVertex) == true) {

g2d.color = Color.RED

drawEdge(visited[i], currVertex)

g2d.color = VERTEX\_COLOR

break

}

}

}

}

}

override fun paintComponent(g: Graphics) {

super.paintComponent(g)

g2d = g as? Graphics2D ?: return

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON)

drawStartVertex()

drawVisitedVertices()

drawStronglyConnectedComponents()

drawVertices()

drawExplanation()

startButton?.isEnabled = vertices.isNotEmpty() && mouseHandler.addingVertex

}

private fun drawExplanation() {

with(g2d) {

val x = 150

var y = 20

color = TEXT\_COLOR

font = Font("Arial", Font.BOLD, 14)

val words: List<String>

if (visited.isNotEmpty())

words = (explanation.text + ". Переход к вершине " +

visited.last().id.toString() + ".").split(" ")

else words = explanation.text.split(" ")

var line = StringBuilder()

val maxWidth = width - 150

for (word in words) {

if (fontMetrics.stringWidth("$line $word") <= maxWidth) {

line.append(word).append(" ")

} else {

drawString(line.toString(), x, y)

y += fontMetrics.height

line = StringBuilder("$word ")

}

}

drawString(line.toString(), x, y)

}

if (explanation !in listOf(Explanations.DFSONTRANSPOSE, Explanations.START,

Explanations.DFS, Explanations.TRANSPOSEGRAPH, Explanations.RESULT))

explanation = Explanations.NOTEXT

}

private fun drawEdge(vertex1: Vertex, vertex2: Vertex) {

val dx = vertex2.x - vertex1.x

val dy = vertex2.y - vertex1.y

val angle = atan2(dy.toDouble(), dx.toDouble())

val startEndpoint = calculateEndpoint(vertex1, angle, vertex1.radius / 2)

val endEndpoint = calculateEndpoint(vertex2, angle, -vertex2.radius / 2)

with(g2d) {

drawLine(startEndpoint.x, startEndpoint.y, endEndpoint.x, endEndpoint.y)

drawArrow(endEndpoint.x, endEndpoint.y, angle - Math.PI / 6)

drawArrow(endEndpoint.x, endEndpoint.y, angle + Math.PI / 6)

}

}

private fun calculateEndpoint(vertex: Vertex, angle: Double, radiusOffset: Int): Point {

val x = vertex.x + radiusOffset \* cos(angle)

val y = vertex.y + radiusOffset \* sin(angle)

return Point(x.toInt(), y.toInt())

}

private fun Graphics2D.drawArrow(x: Int, y: Int, angle: Double) {

val arrowSize = ARROW\_SIZE

val x1 = (x - arrowSize \* cos(angle)).toInt()

val y1 = (y - arrowSize \* sin(angle)).toInt()

drawLine(x, y, x1, y1)

}

fun removeAllPoints() {

vertices = mutableMapOf()

sccList = mutableListOf()

vertices.keys.forEach { key -> key.bypassNumber = null }

Vertex.idCounter = 1

repaint()

}

private fun updateMouseHandler(isDrawingEdge: Boolean, isRemovingVertex: Boolean, isRemovingEdge: Boolean) {

with(mouseHandler) {

startVertex = null

this.isDrawingEdge = isDrawingEdge

this.isRemovingVertex = isRemovingVertex

this.isRemovingEdge = isRemovingEdge

}

}

fun addEdge() {

if (vertices.size >= 2) {

updateMouseHandler(

isDrawingEdge = true,

isRemovingVertex = false, isRemovingEdge = false

)

explanation = Explanations.CHOOSEVERTEX1

repaint()

}

}

fun removeVertex() {

if (vertices.isNotEmpty()) {

updateMouseHandler(

isDrawingEdge = false,

isRemovingVertex = true, isRemovingEdge = false

)

explanation = Explanations.DELVERTEX

repaint()

}

}

fun removeEdge() {

if (vertices.any { (\_, value) -> value.size >= 1 }) {

updateMouseHandler(

isDrawingEdge = false,

isRemovingEdge = true, isRemovingVertex = true

)

explanation = Explanations.CHOOSEVERTEX1

repaint()

}

}

fun arrangeVerticesInCircle() {

sccList = mutableListOf()

explanation = Explanations.NOTEXT

val radius = min(width, height) \* 0.4

val centerX = width / 2 + 50

val centerY = height / 2 + 15

val angleStep = 2 \* PI / vertices.size

var angle = 0.0

for (vertex in vertices.keys) {

val x = centerX + (radius \* cos(angle)).toInt()

val y = centerY + (radius \* sin(angle)).toInt()

vertex.x = x

vertex.y = y

angle += angleStep

}

repaint()

}

private fun randomColor(): Color {

val random = Random()

val red = random.nextInt(256)

val green = random.nextInt(256)

val blue = random.nextInt(256)

return Color(red, green, blue)

}

private fun getVertexColor(id: Int): Color {

return sccColorList.getOrPut(id) { randomColor() }

}

fun transposeGraph(graph: MutableMap<Vertex, MutableSet<Vertex>>): MutableMap<Vertex, MutableSet<Vertex>> {

val transposedGraph: MutableMap<Vertex, MutableSet<Vertex>> = mutableMapOf()

for ((vertex, \_) in graph)

transposedGraph[vertex] = mutableSetOf()

for ((vertex, adjacencyList) in graph) {

for (adjacencyVertex in adjacencyList) {

transposedGraph[adjacencyVertex]?.add(vertex)

}

}

return transposedGraph

}

fun generateGraph(){

removeAllPoints()

vertices = GraphGenerator.generateGraph()

arrangeVerticesInCircle()

}

}

import java.awt.Point

import java.awt.Rectangle

class Vertex(

var x: Int = 0,

var y: Int = 0,

) {

val radius: Int = 40

var bypassNumber: Int? = null

var id: Int? = null

private set

companion object {

var idCounter: Int = 1

fun createWithId(x: Int, y: Int): Vertex {

val vertex = Vertex(x, y)

vertex.id = idCounter++

return vertex

}

}

private val bounds: Rectangle

get() = Rectangle(x - radius / 2, y - radius / 2, radius, radius)

fun containsPoint(point: Point): Boolean {

return bounds.contains(point)

}

}