

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

ОТЧЕТ
по учебной практике
Тема: Сильно связанные компоненты орграфа

Студентка гр. 3383

Кривошеина Д.А.

Студент гр. 3383

Матвеев Н.С.

Руководитель

Фирсов М.А.

Санкт-Петербург

2025

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студентка Кривошеина Д.А. группы 3383

Студент Матвеев Н.С. группы 3383

Тема практики: Сильно связанные компоненты орграфа

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: Алгоритм Косарайю (поиск компонент сильной связности).

Сроки прохождения практики: 25.06.2024 – 08.07.2024

Дата сдачи отчета: 07.07.2024

Дата защиты отчета: 07.07.2024

Студентка		Кривошеина Д.А.
Студент		Матвеев Н.С.
Руководитель		Фирсов М.А.

АННОТАЦИЯ

Необходимо разработать и реализовать графическое приложение, визуализирующее работу алгоритма Косарайю по поиску компонент сильной связности (КСС), на Java. Программа должна наглядно демонстрировать работу алгоритма и предоставлять удобный интерфейс для создания графа (в том числе сохранения/загрузки графа), выполнения алгоритма по шагам, просмотра анимации процесса выполнения алгоритма на графе и незамедлительного получения результата.

В ходе выполнения работы необходимо:

- Определить требования к приложению и архитектуру, составить план разработки
- Реализовать структуры для представления графа, алгоритм, находящий КСС в графе, и понятный графический интерфейс
- Защитить проект

SUMMARY

It is necessary to develop and implement a graphical application that visualizes the operation of Kosaraju's algorithm for finding strongly connected components (SCC) in Java. The program should clearly demonstrate the algorithm's execution and provide a user-friendly interface for creating graphs (including saving/loading graphs), executing the algorithm step-by-step, viewing an animation of the algorithm's process on the graph, and obtaining immediate results.

During the project implementation, it is required to:

- Define application requirements and architecture, and create a development plan
- Implement structures for graph representation, the SCC-finding algorithm, and an intuitive graphical interface
- Defend the project

СОДЕРЖАНИЕ

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ	1
АННОТАЦИЯ	2
СОДЕРЖАНИЕ	3
ВВЕДЕНИЕ	4
Задача	4
Описание алгоритма	4
1. ТРЕБОВАНИЯ К ПРОГРАММЕ	5
1.1. Исходные требования к программе	5
1.1.1 Требования к визуализации	5
1.1.2 План тестирования	7
1.2 Уточнение требований после сдачи прототипа	8
1.3 Уточнения требований после сдачи версии 1	8
1.4 Уточнения требований после сдачи версии 2	8
2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ	9
2.1 План разработки	9
2.2 Распределение ролей	10
3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ	11
3.1. Структуры данных	11
3.1.1 Хранение графа	11
3.1.2 Реализация алгоритма	13
3.1.3 Реализация интерфейса	14
3.2. Основные методы	17
3.2.1 Хранение графа	17
3.2.2 Реализация алгоритма	18
3.2.3 Реализация интерфейса	21
4. ТЕСТИРОВАНИЕ	28
4.1. Обработка ошибок	28
4.2. Проверка работы алгоритма на различных графах:	32
4.3. Проверка корректности работы интерфейса:	37
ЗАКЛЮЧЕНИЕ	39
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	40
ПРИЛОЖЕНИЕ А	41

ВВЕДЕНИЕ

Задача

Реализовать на языке программирования Java программу, представляющую собой визуализатор алгоритма нахождения компонент сильной связности орграфа.

Описание алгоритма

Используется алгоритм Kosaraju (Косарайю).

Алгоритм:

- 1) обходом в глубину рассчитывается порядок выхода вершин исходного графа.
- 2) исходный граф транспонируется.
- 3) выполняется обход в глубину в транспонированном графе в обратном порядке выхода вершин.
- 4) в течение обхода в глубину формируются компоненты сильной связности.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

1.1.1 Требования к визуализации

Библиотека: JavaFX

Элементы интерфейса: Холст для графа, панель управления, лог с текстовым выводом шагов алгоритма.

Граф: Граф строго ориентирован, веса не используются, кратные ребра не создаются, максимальное количество вершин 15.

Хранение графа в файле JSON: в файле формата JSON граф будет храниться как список вершин и список рёбер; каждая вершина будет характеризоваться такими полями, как координата на холсте (по x и по y) и номер (id); каждое ребро будет характеризоваться номером вершины, из которой оно исходит (source), номером вершины, в которую оно входит (target), и цветом (color).

Элементы управления холстом:

Кнопки:

- «Загрузить граф» — загружает граф из файла формата JSON.
- «Сохранить граф» — скачивает граф в файл формата JSON.
- «Добавить вершину» — добавляет вершину на холст нажатием левой кнопки мыши; вершины автоматически нумеруются при создании.
- «Добавить ребро» — добавляет ориентированное ребро на холст нажатием левой кнопки мыши — сначала по начальной вершине, потом по конечной.
- «Очистить граф» — удаляет полностью граф с холста.
- «Удалить элемент» — удаляет с холста элемент, выбранный нажатием левой кнопки мыши.

Элементы управления алгоритмом:

Кнопки:

- «СТАРТ / ПАУЗА» — если алгоритм еще не запущен, запускает визуализацию работы алгоритма и вывода поясняющего текста в отдельной области, в ином случае приостанавливает визуализацию алгоритма, повторное нажатие возобновляет работу.
- «Получить результат» — сразу показывает результат полной работы алгоритма (без пошаговой визуализации алгоритма).
- «Выполнить по шагам» — алгоритм выполняется шаг за шагом и для выполнения следующего шага ожидается нажатие пользователем этой же кнопки, которая после начала выполнения алгоритма изменит свое название на «Следующий шаг».

Визуализация алгоритма:

Древесные ребра отмечаются зеленым цветом, обратные — голубым, направленные вперёд — синим и поперечные — фиолетовым. Выход из вершины помечается номером в стеке, во время работы алгоритма на вершинах выводятся номера в порядке обхода.

В конце работы вершины компоненты сильной связности будут выделены соответствующим номером. Примерный вид интерфейса представлен на рис.1:

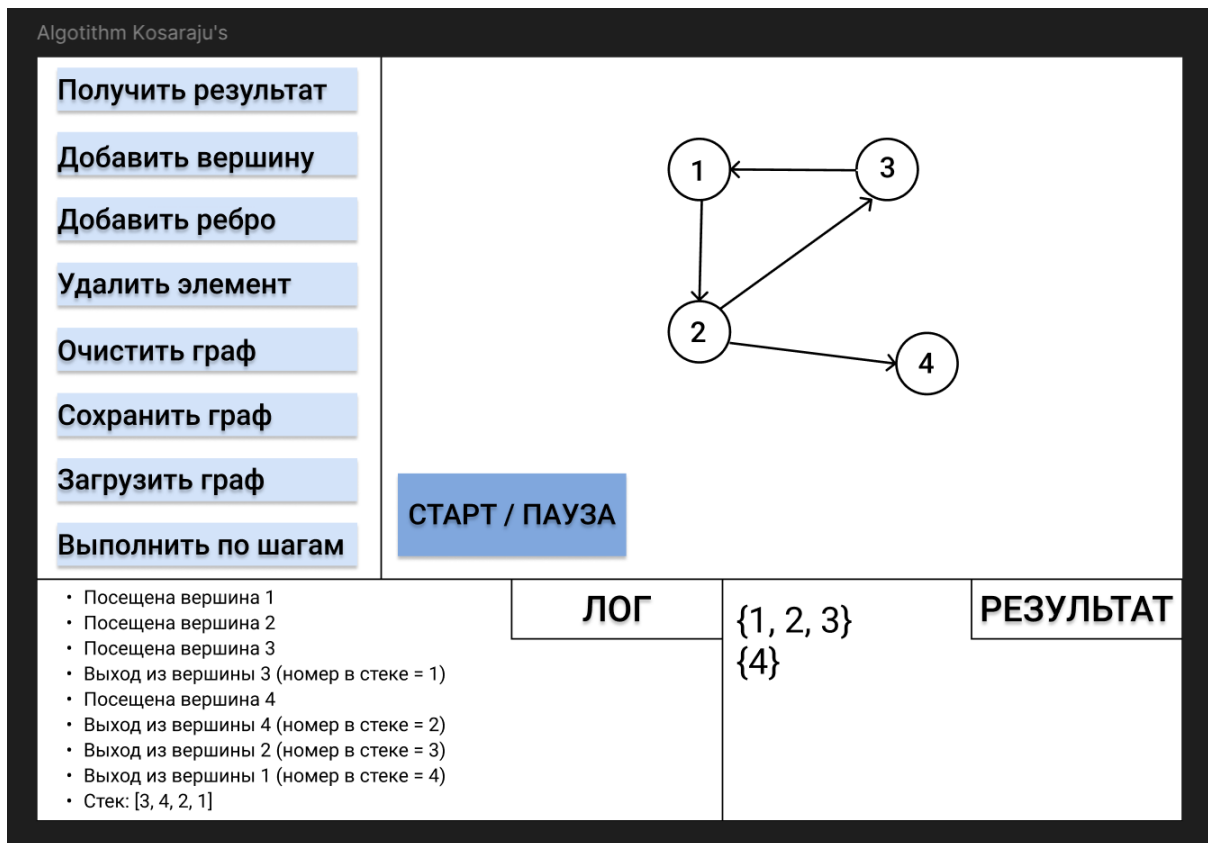


Рисунок 1: Примерный дизайн интерфейса

1.1.2 План тестирования

Обработка ошибок:

- Работа с файлом: некорректный файл, файл не формата JSON.
- Работа алгоритма: пустой граф.
- Работа интерфейса: добавление вершин больше допустимого значения, создание нескольких одинаковых ребер из одной вершины в другую

Проверка корректности работы интерфейса: Нажатие кнопок, расширение экрана, работа холста, работа текстового поля.

Проверка работы алгоритма на различных графах: граф из одной вершины, граф с несколькими компонентами сильной связности, несвязный граф, полный граф, граф с петлями.

1.2 Уточнение требований после сдачи прототипа

Окрасить кнопки в разные цвета в зависимости от их принадлежности к тому или иному смысловому блоку.

Первый смысловой блок (выполнение алгоритма):

- «Получить результат»
- «Выполнить по шагам»

Второй смысловой блок (создание графа на холсте):

- «Добавить вершину»
- «Добавить ребро»
- «Удалить элемент»
- «Очистить граф»

Третий смысловой блок (работа с файлом):

- «Загрузить граф»
- «Сохранить граф»

1.3 Уточнения требований после сдачи версии 1

1. В логе выделить крупные этапы: 1-ый и 2-ой обходы начинать с красной строки.
2. Размер вершин чуть-чуть уменьшить.
3. Выводить диалог сохранения файла (с подтверждением в случае перезаписи существующего) при сохранении графа.
4. Если во время обхода уже некуда идти, но при этом остались непройденные вершины, то "прыжок" отмечать в логе.

1.4 Уточнения требований после сдачи версии 2

1. Лог должен прокручиваться на выполненный шаг автоматически.
2. Неактивные в данный момент кнопки делать серых оттенков.
3. Добавлять на вершины номера в порядке обхода и на первом этапе тоже.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1 План разработки

План разработки представлен в табл.1.

Таблица 1: План разработки

Дата	Этап проекта	Реализованные возможности	Выполнено
30.06.25	Согласование плана и спецификации	Написание требований и плана	+
01.07.25	Сдача прототипа	Разработка прототипа интерфейса и написание алгоритма	+
02.07.25	Сдача версии 1	Добавить функционал, обеспечивающий работоспособность кнопок, встроить алгоритм без поэтапной визуализации на исходном графе, то есть кнопки «Выполнить по шагам» и «СТАРТ / ПАУЗА» не будут работать, но будет работать кнопка «Перейти к результату», после нажатия которой на исходном графе будет визуализирован результат и в ячейке «ЛОГ» будут прописаны текстовые пояснения к тому, какие шаги выполнялись для получения результата	+
04.07.25	Сдача версии 2	Добавить поэтапную	+

		<p>визуализацию алгоритма на исходном графе, обеспечивающую работоспособность кнопок «Выполнить по шагам» и «СТАРТ / ПАУЗА»</p>	<p>Отклонения:</p> <ul style="list-style-type: none"> - Древесные ребра не выделяются зеленым - Получение результата не работает при пошаговом и анимационном выполнении <p>07.07.25 Сдана финальная версия с доработками</p>
07.07.25	Сдача отчёта		

2.2 Распределение ролей

Разработка алгоритма и его внедрение в интерфейс: Кривошеина Дарья

Разработка интерфейса: Матвеев Никита

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

3.1.1 Хранение графа

Абстрактный класс Element

Описание: Абстрактный базовый класс для элементов графа.

Класс Vertex (наследник Element)

Описание: Вершина графа.

Поля:

- id (int) — Уникальный идентификатор
- label (String) — Текстовая метка
- x, y (double) — Координаты для визуализации
- color (CustomColor) — Цвет вершины
- entry_time, exit_time (int) — Время входа/выхода при DFS-обходе
- stack_number (int) — Позиция в стеке алгоритма
- SCC_number (int) — Номер компоненты сильной связности
- bypass_number (int) — Порядковый номер обхода

Основные методы:

- Конструкторы: Инициализация вершины с параметрами
- getCoordinate() — Возвращает массив координат [x, y]
- Геттеры/сеттеры для всех полей.

Класс Edge (наследник Element)

Описание: Ребро графа.

Поля:

- source (int) — ID вершины-источника
- target (int) — ID вершины-назначения
- color (CustomColor) — цвет ребра
- edge_type (EdgeType) — тип ребра

Основные методы:

- Конструкторы: Инициализация ребра с параметрами
- Геттеры/сеттеры для всех полей.

Класс-enum EdgeType

Описание: Типы рёбер в графе.

Поля (значения перечисления):

- NONE — Неопределённый тип
- TREE — Древесное ребро
- BACK — Обратное ребро
- FORWARD — Направленное вперед ребро
- CROSS — Поперечное ребро

Класс-enum CustomColor

Описание: Перечисление цветов для визуализации элементов графа.

Поля (значения перечисления):

- BLACK, GRAY, LIGHTGRAY, WHITE, GREEN, CYAN, BLUE, PURPLE, LIGHTGREEN

Представляют цветовые константы для раскраски вершин/рёбер.

Класс Graph (наследник Element)

Описание: Представление графа.

Поля:

- edges (ArrayList<Edge>) - Список рёбер
- vertexes (ArrayList<Vertex>) - Список вершин

- steps (StepwiseExecution) - История шагов алгоритма

Основные методы:

- sortLists() - Сортирует вершины по ID и рёбра по ID вершины-источника/вершины-цели
- transpose() - Транспонирует граф (инвертирует направление рёбер)
- Геттеры/сеттеры для всех полей.

3.1.2 Реализация алгоритма

Класс SCC (Strongly Connected Components)

Описание: Реализация алгоритма поиска сильно связанных компонент (KCC) в ориентированном графе с пошаговой фиксацией операций.

Класс статический и не имеет полей.

Класс-enum Operations

Описание: Операции во время выполнения алгоритма

Поля (значения-перечисления):

- ENTER — Вход в вершину
- LEAVE — Выход из вершины
- NUMBER — Нумерация вершины (по номеру обхода либо по номеру KCC)
- TRANSPOSE — Транспонирование графа
- GO — Проход по ребру
- COLOR — Окрашивание вершины/ребра в цвет

Класс StepwiseExecution

Описание: Фиксация последовательности операций алгоритма для пошаговой визуализации.

Поля:

- operation_sequence (ArrayList<Pair<Operations, Element>>) — Список пар (операция, элемент), хранит последовательность действий и элементов, над которыми выполняются эти действия
- colors (ArrayList<CustomColor>) — Список цветов для операций окрашивания (синхронизирован с operation_sequence)
- logs (ArrayList<String>) — Текстовые описания шагов алгоритма (синхронизирован с operation_sequence)
- logs_modified — Флаг модификации логов (если логи уже модифицированы = true, иначе = false)

3.1.3 Реализация интерфейса

Класс InterfaceController

Описание: Управление взаимодействием с UI (кнопки, обработка событий, отрисовка графа).

Поля:

- primaryStage (Stage) — Основное окно приложения
- graphCanvas (Pane) — Область для рисования графа (холст)
- Управляющие кнопки (Button) — deleteAll, createVertex, createEdge, deleteSmth, fastResult, stepsResult, saveGraph, loadGraph, playPause
- animationController (AnimationController) — Контроллер анимации алгоритма
- styleActiveButton (String) — Стил активнoй в данный момент кнопки
- activeButton (Button) — Активнaя в данный момент кнопка
- isEditMode (boolean) — Флаг проверки пребывания в режиме редактирования графа
- r (double) — радиус вершины графа

- `isAlgorithm` (boolean) — Флаг проверки того, в процессе ли выполнения сейчас алгоритм (для `playPause` и `stepsResult`)
- `isFinishAlgoWithTranspose` (boolean) — Флаг проверки того, завершено ли окончательно выполнение алгоритма (для `playPause` и `stepsResult`)
- `baseGraph` (Graph) — Текущее состояние графа
- `max_vertexes` (int) — Ограничение на количество вершин в графе
- `playImage`, `pauseImage` (Image) — Изображения для кнопки «Старт/Пауза»
- `stepByStepMode` (boolean) — Флаг нахождения в режиме выполнения по шагам
- `isFast` (boolean) — Флаг проверки выполнения кнопки «Получить результат»
- `graphForStart` (Graph) — Изначальное состояние графа (до отрисовки на нем визуализации)

Класс `ClassicEdge` (наследник `Group`)

Описание: Визуальное представление ребра графа.

Поля:

- `line` (Line) — Основная линия ребра
- `leftWing`, `rightWing` (Line) — левая и правая части стрелки
- `startX/Y`, `endX/Y` (double) — координаты начала/конца ребра

Класс `AnimationController`

Описание: Управляет анимацией алгоритмов на графе, включая визуализацию шагов, изменение цветов элементов, пошаговое выполнение.

Поля:

- `graphCanvas` (Pane) — Холст, на котором отрисовываются элементы графа (вершины, ребра)
- `timeline` (Timeline) — Хранит последовательность кадров (KeyFrame), каждый из которых соответствует шагу алгоритма

- `isPaused (boolean)` — Флаг состояния анимации (`true` — анимация приостановлена, `false` — анимация активна)
- `interfaceApp (CustomInterface)` — Ссылка на главный интерфейс приложения
- `currentStep (int)` — Текущий шаг в режиме пошагового выполнения
- `stepActions (List<Runnable>)` — Список действий для пошагового выполнения, каждый `Runnable` соответствует одной операции (например, раскраске вершины)

Класс CustomInterface

Описание: Класс отвечает за построение графического интерфейса приложения для визуализации алгоритмов на графах. Содержит все UI-компоненты: область отрисовки графа, панели управления, текстовые области вывода.

Поля:

- `rootPane (GridPane)` — Организует расположение всех компонентов в сетке.
- `leftButtons (VBox)` — Вертикальная панель с кнопками управления в левой части интерфейса
- `areaForGraph (StackPane)` — Контейнер для области отрисовки графа и кнопки воспроизведения, включает `ScrollPane` для прокрутки большого графа и кнопку `Play` в левом нижнем углу
- `answer (BorderPane)` — Область вывода результатов работы алгоритма
- `log (BorderPane)` — Область вывода логов выполнения алгоритма, автоматически прокручивается вниз при добавлении нового текста
- `graphCanvas (Pane)` — Холст для отрисовки элементов графа (вершин, рёбер)
- `editStyle, fileStyle, playStyle, resultStyle (String)` — CSS-стили для различных элементов (`editStyle` — стиль кнопок редактирования графа,

fileStyle — стиль кнопок работы с файлами, playStyle — стиль кнопки воспроизведения, resultStyle — стиль кнопок запуска алгоритмов)

Класс GraphJsonUtils

Описание: Обеспечивает сериализацию и десериализацию объектов графа в формат JSON, включает операции сохранения графа в файл, загрузки из файла и удаления файлов с графами.

Поля:

- gson (Gson) — Экземпляр библиотеки Gson для работы с JSON, настроенный через GsonBuilder (игнорирует поля без аннотации @Expose, включает красивое форматирование вывода, используется во всех методах класса)

3.2. Основные методы

3.2.1 Хранение графа

Класс Vertex (наследник Element)

Основные методы:

- Конструктор public Vertex(int id, String label, double x, double y) — Инициализация вершины с параметрами.
- Геттеры/сеттеры для всех полей класса.

Класс Edge (наследник Element)

Основные методы:

- Конструкторы public Edge(int source, int target, CustomColor color) — Инициализация ребра с параметрами.
- Геттеры/сеттеры для всех полей класса.

Класс Graph (наследник Element)

Основные методы:

- Конструктор `public Graph(ArrayList<Edge> edges, ArrayList<Vertex> vertexes)` — Метод принимает на вход: `edges` - список ребер, `vertexes` - список вершин. Инициализирует поля класса графа.
- `public void sortLists()` — Метод не принимает ничего на вход. Сортирует вершины по ID и рёбра по ID вершины-источника/вершины-цели. Метод ничего не возвращает.
- `private void sortEdgeList()` — Метод не принимает ничего на вход. Сортирует ребра по ID вершины-источника и вершины-цели. Метод ничего не возвращает.
- `private void sortVertexList()` — Метод не принимает ничего на вход. Сортирует вершины по ID. Метод ничего не возвращает.
- `public void transpose()` — Метод не принимает ничего на вход. Транспонирует граф (инвертирует направление рёбер). Метод ничего не возвращает.
- Геттеры/сеттеры для всех полей класса.

3.2.2 Реализация алгоритма

Класс SCC (Strongly Connected Components)

Основные методы:

- `private static ArrayList<Integer> count_exit_order(Graph graph, boolean[] visited, StringBuilder log)` — Метод принимает на вход: `graph` - граф, `log` - строку, в которую будет записан лог. Рассчитывает порядок выхода вершин при обходе в глубину (DFS), для каждой непосещенной вершины запускает DFS, сохраняя вершины в порядке завершения их обработки. Возвращает список вершин в обратном порядке времени выхода.
- `private static int dfs(Graph graph, int vertex, boolean[] visited, ArrayList<Integer> stack, int time, StringBuilder log, int flag)` — Метод принимает на вход: `graph` - граф, `vertex` - вершина для начала обхода,

visited - список всех вершин с флагами их посещенности, stack - стек для сохранения порядка выхода из вершин, int time - текущее “время”, log - строка для записи лога, flag - флаг, сигнализирующий о том, первый или второй обход происходит. Выполняет рекурсивный обход в глубину. Возвращает время выхода после обработки вершины.

- private static int search_edge(Graph graph, int vertex, boolean[] visited, int last_stop, StringBuilder log, int flag) — Метод принимает на вход: graph - граф, vertex - вершина, ребро от которой надо найти, visited - список всех вершин с флагами их посещенности, last_stop - место в списке ребер, все ребра до которого уже были проверены ранее, log - строка для записи лога, flag - флаг, сигнализирующий о том, первый или второй обход происходит. Ищет подходящее ребро для продолжения обхода в глубину. Возвращает индекс ребра.
- private static Graph transpose_graph(Graph graph) — Метод принимает на вход: graph - граф для транспонирования. Создает транспонированную копию графа. Возвращает транспонированный граф.
- public static ArrayList<ArrayList<Integer>> find_SCC(Graph graph, StringBuilder log) — Метод принимает на вход: graph - граф, log - строку, в которую будет записан лог. Выполняет поиск компонент сильной связности (алгоритм Косарайю). Возвращает список компонент сильной связности.

Класс StepwiseExecution

Основные методы:

- Конструктор public StepwiseExecution() — Метод не принимает ничего на вход. Инициализируются поля класса (создаются объекты полей через new).

- `public void enterVertex(Vertex vertex)` — Метод принимает на вход: `vertex` - вершину, в которую осуществляется вход. Регистрирует операцию входа в вершину. Ничего не возвращает.
- `public void leaveVertex(Vertex vertex)` — Метод принимает на вход: `vertex` - вершина, из которой осуществляется выход. Регистрирует операцию выхода из вершины. Ничего не возвращает.
- `public void colorVertex(Vertex vertex, CustomColor color)` — Метод принимает на вход: `vertex` - вершина, которую нужно окрасить, `color` - цвет, в который нужно окрасить. Регистрирует операцию окраски вершины. Ничего не возвращает.
- `public void numberVertex(Vertex vertex)` — Метод принимает на вход: `vertex` - вершина, которую нужно “пронумеровать” (написать на ней номер). Регистрирует операцию нумерации вершин. Ничего не возвращает.
- `public void goEdge(Edge edge)` — Метод принимает на вход: `edge` - ребро, по которому нужно пройти. Регистрирует переход по ребру. Ничего не возвращает.
- `public void colorEdge(Edge edge, CustomColor color)` — Метод принимает на вход: `edge` - ребро, которое нужно окрасить, `color` - цвет, в который нужно окрасить. Регистрирует операцию окраски ребра. Ничего не возвращает.
- `public void transposeGraph(Graph graph)` — Метод принимает на вход: `graph` - граф для транспонирования. Регистрирует операцию транспонирования графа. Ничего не возвращает.
- `public void addLog(String log)` — Метод принимает на вход: `log` - текстовый лог на данный момент. Добавляет полную текстовую запись текущего состояния лога. Ничего не возвращает.
- `public void modifyLogs()` — Метод не принимает ничего на вход. Изменяет список логов так, что каждый элемент изменяется от полного лога на

текущий момент на только новые строки относительно прошлого элемента. Ничего не возвращает.

- Геттеры для полей класса.

3.2.3 Реализация интерфейса

Класс `InterfaceController`

Основные методы:

- Конструктор `public InterfaceController(CustomInterface interfaceApp, Stage stage)` — На вход принимает: `interfaceApp` - объект класса интерфейса, `stage` - главная сцена интерфейса. Инициализирует контроллер интерфейса, создаётся объект контроллера анимации и вызывается настройка обработчика всех кнопок.
- `private void controlAllButtons()` — Метод не принимает ничего на вход. Настраивает обработчики событий для всех кнопок интерфейса. Метод ничего не возвращает.
- `private void unpackButtons()` — Метод не принимает ничего на вход. Извлекаем ссылки на кнопки интерфейса из контейнера левой панели интерфейса. Метод ничего не возвращает.
- `private void checkEditMode()` — Метод не принимает ничего на вход. Выход из режима редактирования. Метод ничего не возвращает.
- `private void controlPlayButton()` — Метод не принимает ничего на вход. Обработывает кнопку воспроизведения - паузы анимации. Метод ничего не возвращает.
- `private void controlStepsResultBtn()` — Метод не принимает ничего на вход. Настраивает обработчик кнопки "Выполнить по шагам". Метод ничего не возвращает.
- `private void controlFastResultBtn()` — Метод не принимает ничего на вход. Настраивает обработчик кнопки "Получить результат". Метод ничего не возвращает.

- `private void controlDeleteAllBtn()` — Метод не принимает ничего на вход. Настраивает обработчик кнопки "Очистить граф". Метод ничего не возвращает.
- `private void controlDeleteSmthBtn()` — Метод не принимает ничего на вход. Настраивает обработчик кнопки "Удалить элемент", включает режим редактирования. Метод ничего не возвращает.
- `private void controlCreateVertexBtn()` — Метод не принимает ничего на вход. Настраивает обработчик кнопки "Добавить вершину", включает режим редактирования. Метод ничего не возвращает.
- `private void controlCreateEdgeBtn()` — Метод не принимает ничего на вход. Настраивает обработчик кнопки "Добавить ребро", включает режим редактирования. Метод ничего не возвращает.
- `private void controlLoadGraphBtn()` — Метод не принимает ничего на вход. Настраивает обработчик кнопки "Загрузить граф". Метод ничего не возвращает.
- `private void controlSaveGraphBtn()` — Метод не принимает ничего на вход. Настраивает обработчик кнопки "Сохранить граф". Метод ничего не возвращает.
- `private StackPane createContainerVertex(double x, double y, String label)` — Метод принимает: координаты для вершины, `label` - название вершины. Создает вершину на холсте. Возвращает объект класса `StackPane` (вершина на основе контейнера).
- `private Alert createAlert(String message)` — Метод принимает на вход: `message` - сообщение, которое будет показываться в диалоговом окне. Создает кастомизированное информационное диалоговое окно. Возвращает объект класса `Alert` (диалоговое окно).
- `private static Graph createGraph(Pane graphCanvas)` — Метод принимает на вход: `graphCanvas` - холст, где рисуется граф. Преобразует графические элементы на холсте в объект `Graph`. Метод возвращает объект класса граф.

- `private void drawGraphFromLoad(Graph graph)` — Метод принимает на вход: `graph` - граф. Визуализирует загруженный граф на холсте. Метод ничего не возвращает.
- `private CubicCurve drawLoopEdge(double centerX, double centerY, double r)` — Метод принимает на вход: координаты и `r` - радиус круга (отображение вершины на холсте). Создает ребро-петлю для холста. Метод возвращает кубическую кривую (объект класса `CubicCurve`).
- `private void changeColorDefault()` — Метод не принимает ничего на вход. Окрашивает кнопки по умолчанию. Метод ничего не возвращает.
- `private void changeColorNon()` — Метод не принимает ничего на вход. Окрашивает кнопки в неактивный формат. Метод ничего не возвращает.
- `private double[] findCoordForEdge(StackPane source, StackPane target)` — Рассчитывает координаты центров вершин для ребра между ними. Метод возвращает массив типа `double`, который содержит координаты.
- `private double[] findCoordForDrawEdge(Vertex v1, Vertex v2)` — Метод принимает на вход две вершины. Рассчитывает координаты начала и конца ребра между двумя вершинами с учетом их радиуса. Метод возвращает массив типа `double`, который содержит координаты.
- `private StackPane findStackPaneAt(double x, double y)` — Метод принимает на вход координаты. Находит вершину (`StackPane`) по координатам курсора. Возвращает объект класса `StackPane`.

Класс `ClassicEdge` (наследник `Group`)

Основные методы:

- Конструктор `public ClassicEdge(double startX, double startY, double endX, double endY)` — Метод принимает на вход: координаты начала и конца. Создание ребра для холста, основываясь на координатах.
- `public void setAnotherColorLines(Color color)` — Метод принимает на вход `color` - цвет для раскраски ребра. Смена цвета ребра. Метод ничего не возвращает.

- Геттеры для координат.

Класс AnimationController

Основные методы:

- Конструктор `public AnimationController(Pane pane, CustomInterface interfaceApp)` — Метод принимает на вход: `pane` - холст, `interfaceApp` - интерфейс. Инициализирует поле холста, поле интерфейса.
- `private String giveColor(CustomColor color)` — Метод принимает на вход: `color` - запрашиваемый цвет. Конвертирует цвет, соответствующий элементу из перечисления, в цвет для работы с библиотекой интерфейса. Возвращает строку с цветом для работы с библиотекой интерфейса.
- `public void transposeEdges(Graph graph)` — Метод принимает на вход: `graph` - граф, в котором необходимо транспонировать ребра. Транспонирует все ребра графа на холсте и вызывает транспонирование ребер в классе графа. Ничего не возвращает.
- `private void uncolorVertexContour(Vertex vertex)` — Метод принимает на вход: `vertex` - вершину. Ставит стандартную раскраску контура для вершины. Ничего не возвращает.
- `private void uncolorVertexBackground(Vertex vertex)` — Метод принимает на вход: `vertex` - вершину. Ставит стандартную раскраску внутренней части вершины. Ничего не возвращает.
- `private void colorVertexBackground(Vertex vertex, CustomColor customColor)` — Метод принимает на вход: `vertex` - вершину, `customColor` - цвет, в который её нужно окрасить. Выполняет раскраску внутренней части вершины. Ничего не возвращает.
- `private void colorVertexContour(Vertex vertex, CustomColor customColor)` — Метод принимает на вход: `vertex` - вершину, `customColor` - цвет, в который её нужно окрасить. Выполняет раскраску контура вершины. Ничего не возвращает.

- `private void colorEdge(Edge edge, CustomColor customColor)` — Метод принимает на вход: `edge` - ребро, `customColor` - цвет, в который его нужно окрасить. Выполняет раскраску ребра. Ничего не возвращает.
- `private void uncolorEdge(Edge edge)` — Метод принимает на вход: `edge` - ребро. Ставит стандартную раскраску ребра. Ничего не возвращает.
- `private void addNumber(Vertex vertex, int flag)` — Метод принимает на вход: `vertex` - вершину, `flag` - флаг, сигнализирующий первый или второй обход происходит. Добавляет номер на вершину в процессе обхода. Метод ничего не возвращает.
- `private void removeNumber(Vertex vertex)` — Метод принимает на вход: `vertex` - вершину. Убирает номер с вершины. Метод ничего не возвращает.
- `private void defaultColorAll(Graph graph)` — Метод принимает на вход: граф. Ставятся стандартные раскраски для вершин и ребер. Метод ничего не возвращает. Метод ничего не возвращает.
- `private void colorElement(Element element, CustomColor color)` — Метод принимает на вход: `element` - элемент, который нужно раскрасить, `color` - в какой цвет. Выполняет универсальное окрашивание элементов графа. Метод ничего не возвращает.
- `public void prepareStepExecution(Graph graph, int millis)` — Метод принимает на вход: `graph` - граф и `millis` - количество миллисекунд, за которое выполняется операция. Подготавливает последовательность шагов для пошагового выполнения операций над графом. Метод ничего не возвращает.
- `public void resetSteps()` — Метод ничего не принимает. Очищает последовательность шагов. Метод ничего не возвращает.
- `public boolean executeNextStep()` — Метод ничего не принимает. Выполняет следующий шаг из подготовленной последовательности. Метод возвращает `true` - если шаг выполнен, `false` - иначе.

- `public void visualizeAlgorithm(Graph graph, int millis)` — Метод принимает на вход: `graph` - граф, `milis` - ????. Визуализирует выполнение алгоритма на графе с анимацией. Метод ничего не возвращает.
- `private void processOperation(Operations operation, Element element, ArrayList<CustomColor> colors, int colorIndex, Graph graph, int i, ArrayList<String> logs, int millis, int flag)` — Метод принимает на вход: `operation` - операция для выполнения, `element` - ребро/вершина/граф, к которым применяется операция, `colors` - список цветов для операций окраски, `colorIndex` - индекс последнего использованного цвета из списка цветов, `graph` - граф, `i` - индекс текущей операции в списке всех операций, `logs` - список текстовых логов, сопровождающий выполнение операций, `millis` - количество миллисекунд, за которое выполняется операция, `flag` - флаг, сигнализирующий о том, первый или второй обход происходит. Обработывает конкретную операцию визуализации. Метод ничего не возвращает.
- `public void togglePause()` — Метод ничего не принимает. Приостанавливает или возобновляет анимацию. Метод ничего не возвращает.
- `public Timeline getTimeline()` — Метод ничего не принимает. Возвращает текущее состояние последовательности кадров. Метод возвращает поле с последовательностью кадров анимации.
- `public boolean isPaused()` — Метод ничего не принимает. Выполняет проверку: пауза или идет анимация. Метод возвращает `true` - если сейчас пауза, `false` - иначе.

Класс CustomInterface

Основные методы:

- Конструктор `public CustomInterface()` — Метод ничего не принимает на вход. Вызывает `createGridPane()` для создания главного контейнера интерфейса.

- `private void createGridPane()` — Метод ничего не принимает на вход. Создает корневой макет приложения в виде сетки. Ничего не возвращает.
- `private StackPane graphArea()` — Метод ничего не принимает на вход. Создает область для визуализации графа с интерактивными элементами. Возвращает созданную область для визуализации графа.
- `private VBox buttonsForLeftSide()` — Метод ничего не принимает на вход. Создает панель с кнопками управления слева. Возвращает созданную панель.
- `private BorderPane textProgramArea(String name)` — Метод принимает на вход: `name` - название создаваемой текстовой области. Создает текстовую область с заголовком. Возвращает созданную текстовую область.
- Геттеры для полей класса.

Класс `GraphJsonUtils`

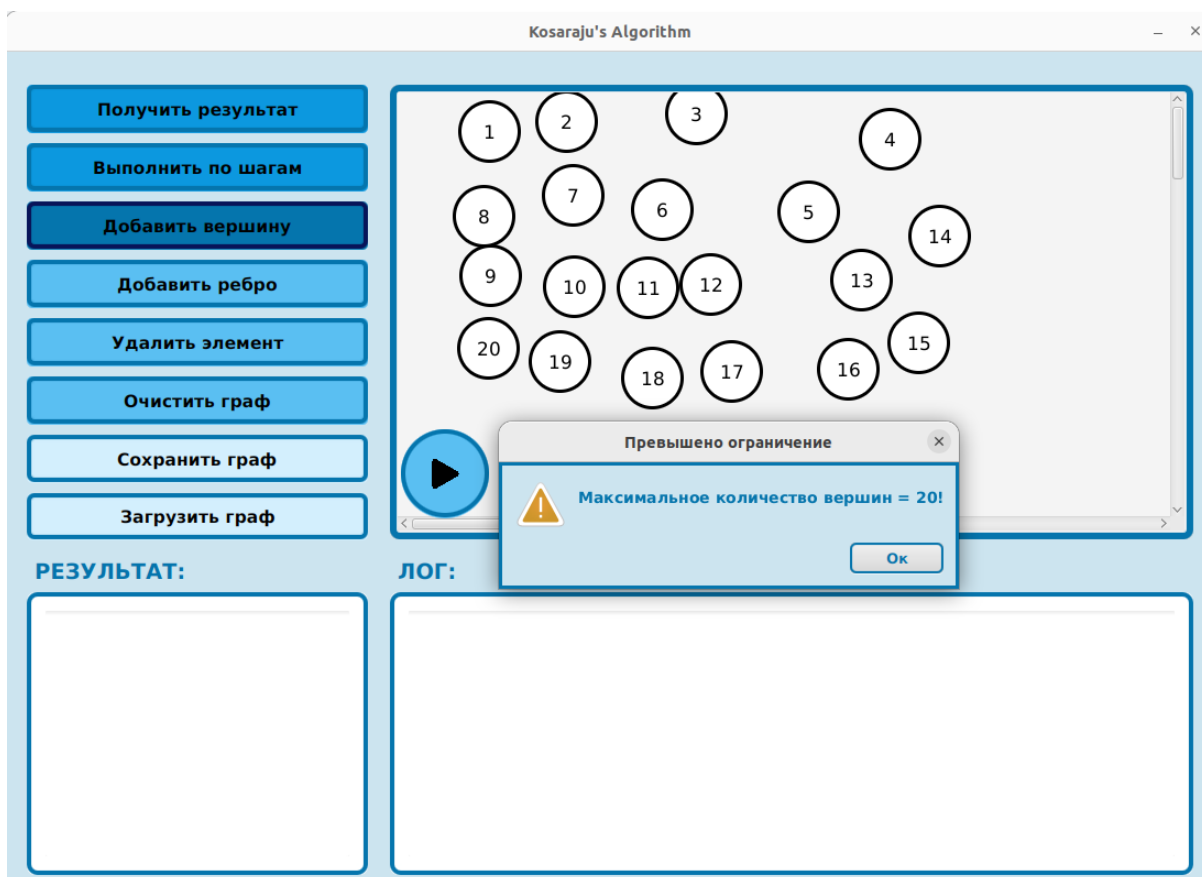
Основные методы:

- `public static void saveGraph(Graph graph, String filePath)` — Метод принимает на вход: `graph` - граф, который нужно сохранить, `filePath` - путь к файлу, в который нужно сохранить граф. Сохраняет объект графа в JSON-файл. Ничего не возвращает.
- `public static Graph loadGraph(String filePath)` — Метод принимает на вход: `filePath` - путь к файлу, из которого нужно загрузить граф. Загружает граф из JSON-файла. Ничего не возвращает.

4. ТЕСТИРОВАНИЕ

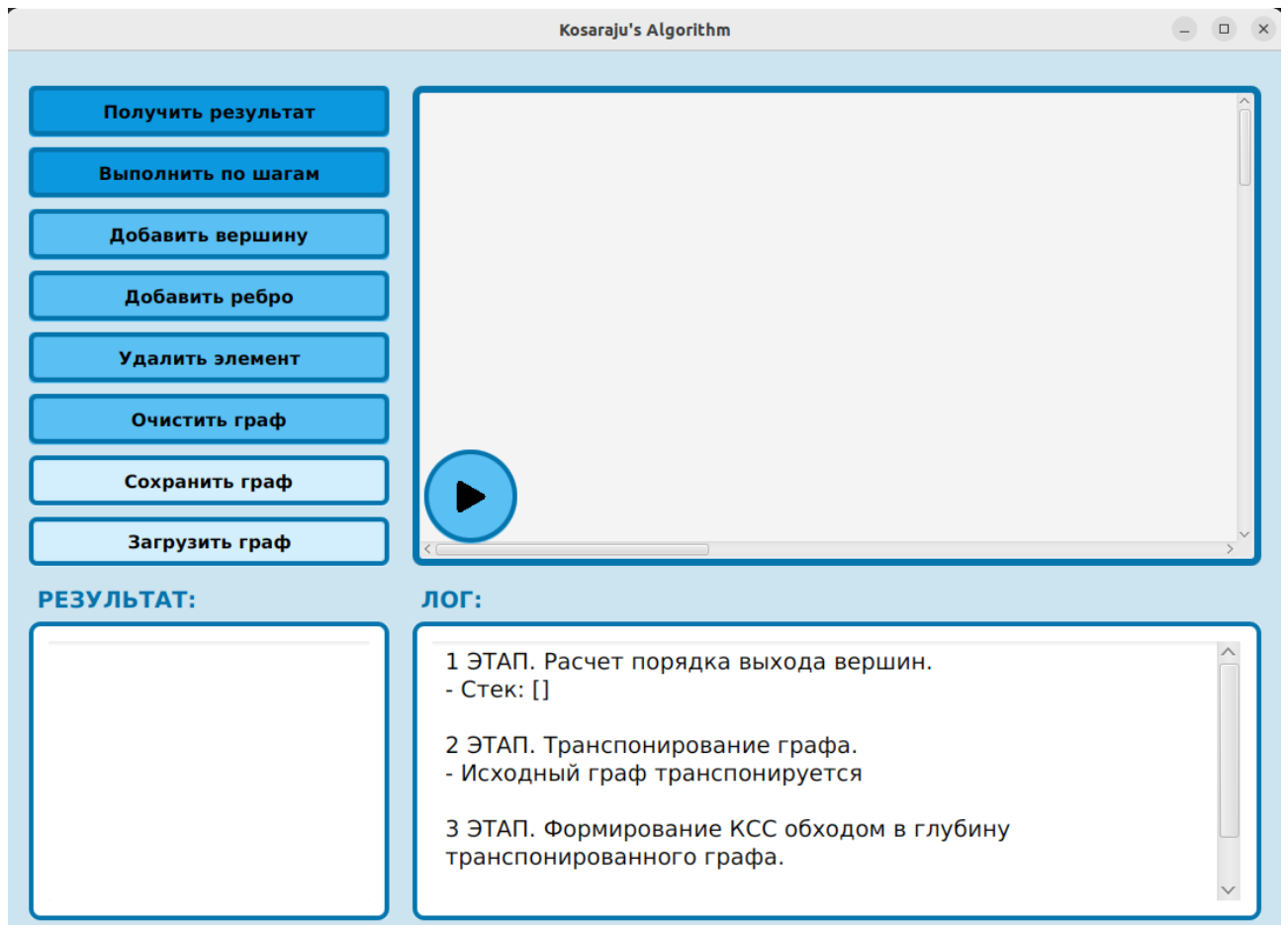
4.1. Обработка ошибок

Добавление вершин больше допустимого количества:

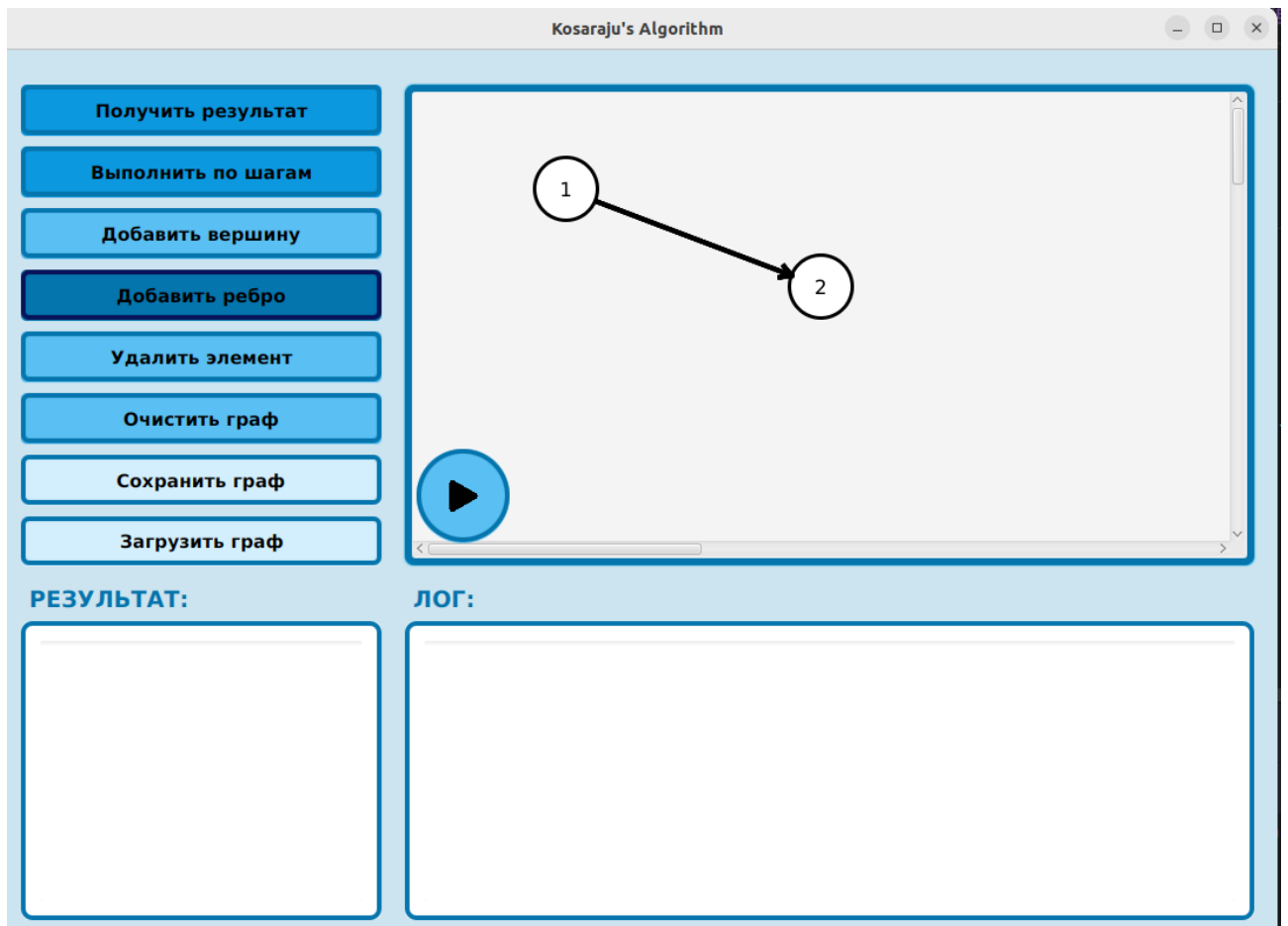


Запуск алгоритма на пустом графе:

Кнопки “Выполнить по шагам” и “Старт/Пауза” заблокированы, для демонстрации представлен результат после нажатия “Получить результат”.

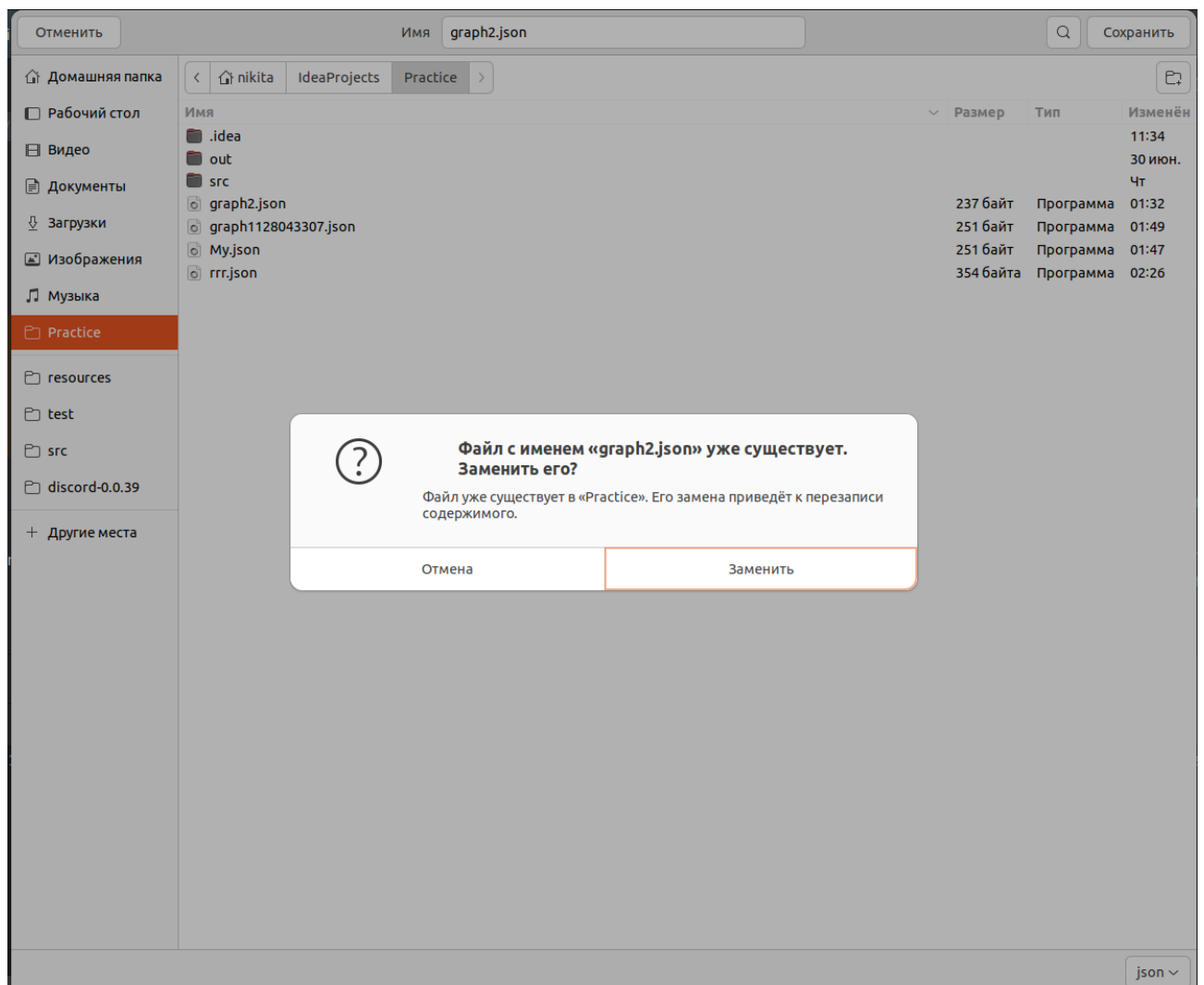


Создание нескольких одинаковых ребер из одной вершины в другую:
Рисуется только одно ребро.

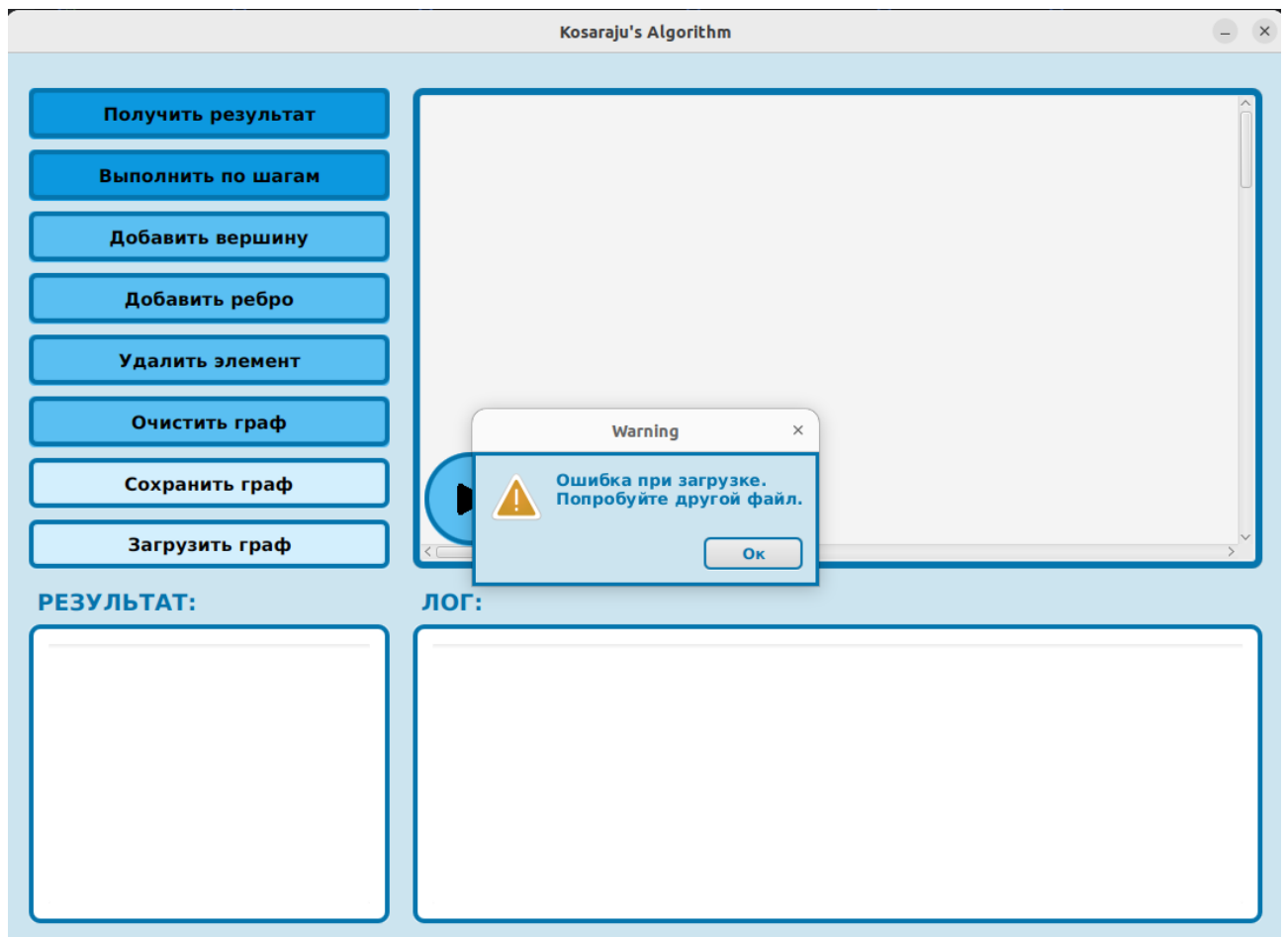


Работа с файлом:

Проверка “Сохранить граф”, если файл с таким именем уже существует:



“Загрузить граф” если файл некорректный:



4.2. Проверка работы алгоритма на различных графах:

1. Граф из одной вершины:

Kosaraju's Algorithm

Получить результат
Выполнить по шагам
Добавить вершину
Добавить ребро
Удалить элемент
Очистить граф
Сохранить граф
Загрузить граф

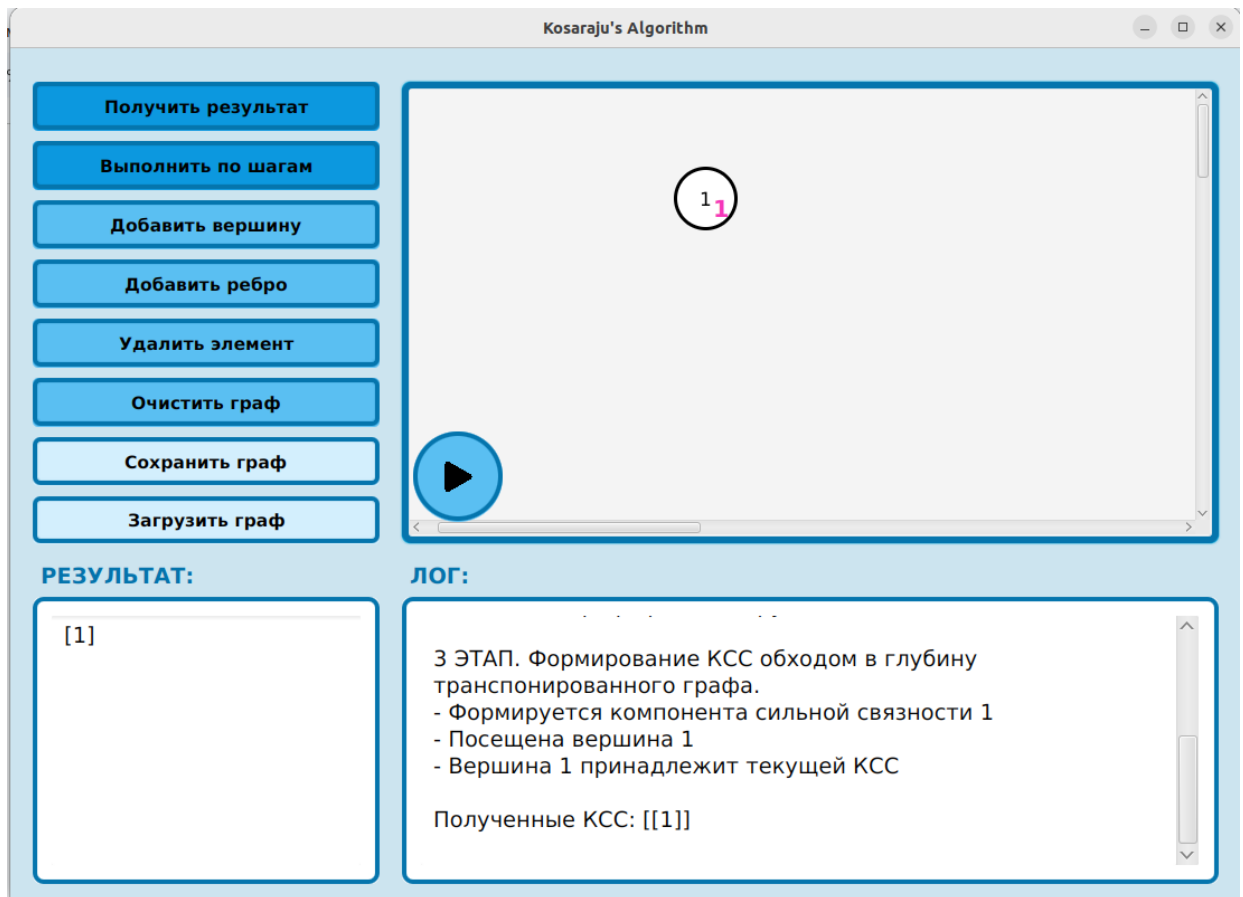
РЕЗУЛЬТАТ:

[1]

ЛОГ:

3 ЭТАП. Формирование КСС обходом в глубину транспонированного графа.
- Формируется компонента сильной связности 1
- Посещена вершина 1
- Вершина 1 принадлежит текущей КСС

Полученные КСС: [[1]]



2. Несвязный граф:

Kosaraju's Algorithm

Получить результат

Выполнить по шагам

Добавить вершину

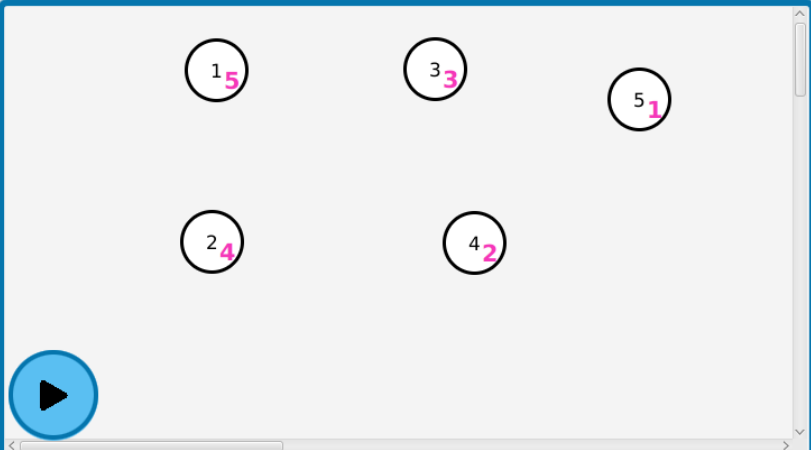
Добавить ребро

Удалить элемент

Очистить граф

Сохранить граф

Загрузить граф



РЕЗУЛЬТАТ:

```
[5]
[4]
[3]
[2]
[1]
```

ЛОГ:

1 ЭТАП. Расчет порядка выхода вершин.

- Прыжок в вершину 1
- Посещена вершина 1
- Вершина 1 добавлена в стек
- Прыжок в вершину 2
- Посещена вершина 2
- Вершина 2 добавлена в стек
- Прыжок в вершину 3
- Посещена вершина 3

3. Граф с несколькими компонентами сильной связности

Kosaraju's Algorithm
⌵ ⌶ ⌵

Получить результат

Выполнить по шагам

Добавить вершину

Добавить ребро

Удалить элемент

Очистить граф

Сохранить граф

Загрузить граф

▶

РЕЗУЛЬТАТ:

```
[8, 7, 9, 6]
[5, 4, 3]
[2, 1]
```

ЛОГ:

- Формируется компонента сильной связности 3
- Посещена вершина 1
- Переход по ребру (1, 2)
- Посещена вершина 2
- Вершина 2 принадлежит текущей КСС
- Вершина 1 принадлежит текущей КСС

Полученные КСС: [[8, 7, 9, 6], [5, 4, 3], [2, 1]]

4. Полный граф

Kosaraju's Algorithm
⌵ □ ✕

Получить результат

Выполнить по шагам

Добавить вершину

Добавить ребро

Удалить элемент

Очистить граф

Сохранить граф

Загрузить граф

▶

РЕЗУЛЬТАТ:

[3, 5, 4, 2, 1]

ЛОГ:

- Посещена вершина 5
- Вершина 5 принадлежит текущей КСС
- Вершина 4 принадлежит текущей КСС
- Вершина 2 принадлежит текущей КСС
- Вершина 1 принадлежит текущей КСС

Полученные КСС: [[3, 5, 4, 2, 1]]

5. Граф с петлями

Kosaraju's Algorithm

Получить результат

Выполнить по шагам

Добавить вершину

Добавить ребро

Удалить элемент

Очистить граф

Сохранить граф

Загрузить граф

1 3

2 4

5 2

3 2

4 2

6 1

▶

РЕЗУЛЬТАТ:

[6]
[4, 5, 3]
[1]
[2]

ЛОГ:

1 ЭТАП. Расчет порядка выхода вершин.
- Прыжок в вершину 1
- Посещена вершина 1
- Ребро (1, 1) обратное
- Переход по ребру (1, 2)
- Посещена вершина 2
- Ребро (2, 2) обратное
- Вершина 2 добавлена в стек
- Вершина 1 добавлена в стек

4.3. Проверка корректности работы интерфейса:

Расширение экрана: Функция переключения в полноэкранный режим и возврата к оконному режиму работает корректно. Интерфейс адекватно адаптируется под новый размер области просмотра, не вызывая потери или наложения элементов.

Нажатие кнопок: Все представленные кнопки интерфейса успешно реагируют на нажатие. Действия, назначенные кнопкам, выполняются корректно. Визуальные изменения состояния при нажатии присутствуют и работают ожидаемо.

Работа холста: Все заявленные взаимодействия с холстом работают корректно. Элементы на холсте отображаются правильно.

Работа текстового поля: Текстовые поля для вывода данных функционируют правильно. Курсор устанавливается корректно по клику. Поле прокрутки длинного текста активируется правильно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было успешно реализовано графическое приложение, визуализирующее работу алгоритма Косарайю, на языке Java.

В процессе разработки были достигнуты следующие результаты:

- Разработан функциональный интерфейс, предоставляющий возможность создания и редактирования графа, сохранения графа в файл формата JSON и загрузки графа из файла формата JSON, получения моментального результата работы алгоритма, просмотра анимации процесса выполнения алгоритма с возможностью поставить паузу, выполнения алгоритма по шагам.
- Выполнение алгоритма сопровождается текстовым описанием каждого действия для большей наглядности.
- Во время анимации ребра и вершины разделяются цветами для удобного отслеживания проделанных алгоритмом шагов и текущего состояния всех компонент графа.
- Все возможные ошибки были учтены, обработаны и проверены во время тестирования, для некоторых ситуаций, когда пользователь совершает ошибку (пытается загрузить файл, в котором нет графа, пытается добавить больше вершин, чем это предусмотрено в программе) появляется диалоговое окно, сообщающее ему о проблеме.

Таким образом, поставленная задача выполнена и реализованное приложение может быть использовано как в учебных целях, так и для демонстрации принципа работы алгоритма Косарайю.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы: Построение и анализ / Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн. М.: “Вильямс”, 2005. 635 с.
2. Алгоритм Kosaraju по полкам. URL: <https://habr.com/ru/articles/537290/> (дата обращения: 30.06.2025)
3. Поиск компонент сильной связности: алгоритм Косарайю. URL: <https://habr.com/ru/articles/331904/> (дата обращения: 30.06.2025)
4. JavaFX. URL: <https://openjfx.io/javadoc/24/> (дата обращения: 01.07.2025)
5. Учебник по JavaFX: начало работы. URL: <https://habr.com/ru/articles/474292/> (дата обращения: 01.07.2025)

ПРИЛОЖЕНИЕ А

CustomColor.java:

```
package algorithm.graph;
```

```
public enum CustomColor {  
    BLACK,  
    GRAY,  
    LIGHTGRAY,  
    WHITE,  
    GREEN,  
    CYAN,  
    BLUE,  
    PURPLE,  
    LIGHTGREEN  
}
```

Edge.java:

```
package algorithm.graph;
```

```
import com.google.gson.annotations.Expose;
```

```
public class Edge extends Element {  
    @Expose  
    private int source;  
    @Expose  
    private int target;  
    @Expose  
    private CustomColor color;  
    @Expose  
    private EdgeType edge_type;  
  
    public Edge(int source, int target, CustomColor color){  
        this.source = source;  
        this.target = target;  
        this.color = color;  
        this.edge_type = EdgeType.NONE;  
    }  
  
    public Edge(int source, int target){  
        this.source = source;  
        this.target = target;  
        this.color = CustomColor.BLACK;  
        this.edge_type = EdgeType.NONE;  
    }  
  
    public int getSource(){  
        return source;  
    }  
  
    public int getTarget(){  
        return target;  
    }  
  
    public CustomColor getColor(){  
        return color;  
    }  
  
    public EdgeType getEdgeType() {  
        return edge_type;  
    }  
}
```

```

    public void setEdgeType(EdgeType edge_type) {
        if (this.edge_type == EdgeType.NONE) {
            this.edge_type = edge_type;
        }
    }

    public void setSource(int source) {
        this.source = source;
    }

    public void setTarget(int target) {
        this.target = target;
    }
}

```

EdgeType.java:

```

package algorithm.graph;

public enum EdgeType {
    NONE,
    TREE,           // древесное
    BACK,           // обратное
    FORWARD,        // направленное вперед
    CROSS           // поперечное
}

```

Element.java:

```

package algorithm.graph;

public abstract class Element {}

```

Graph.java:

```

package algorithm.graph;

import java.util.ArrayList;

import algorithm.StepwiseExecution;
import com.google.gson.annotations.Expose;

public class Graph extends Element {
    @Expose
    private ArrayList <Edge> edges;
    @Expose
    private ArrayList <Vertex> vertexes;
    private StepwiseExecution steps;

    public Graph(ArrayList<Edge> edges, ArrayList<Vertex> vertexes) {
        this.edges = edges;
        this.vertexes = vertexes;
        sortLists();
        this.steps = new StepwiseExecution();
    }

    public void sortLists() {
        sortEdgeList();
        sortVertexList();
    }
}

```

```

public int getVertexCount(){
    return vertexes.size();
}

public ArrayList<Vertex> getVertexList(){
    return vertexes;
}

public int getEdgeCount(){
    return edges.size();
}

public ArrayList<Edge> getEdgeList(){
    return edges;
}

private void sortVertexList(){
    vertexes.sort((v1, v2) -> {
        return Integer.compare(v1.getId(), v2.getId());
    });
}

private void sortEdgeList(){
    edges.sort((e1, e2) -> {
        int cmp = Integer.compare(e1.getSource(), e2.getSource());
        if (cmp != 0) return cmp;
        return Integer.compare(e1.getTarget(), e2.getTarget());
    });
}

public StepwiseExecution getSteps(){
    return steps;
}

public void transpose(){
    for (Edge e : edges){
        int source = e.getSource();
        int target = e.getTarget();
        e.setSource(target);
        e.setTarget(source);
    }
}
}

```

Vertex.java:

```

package algorithm.graph;
import com.google.gson.annotations.Expose;

public class Vertex extends Element {
    @Expose
    private int id;
    @Expose
    private String label;
    @Expose
    private double x;
    @Expose
    private double y;
    @Expose
    private CustomColor color;
    private int entry_time;
    private int exit_time;
}

```

```

private int stack_number;
private int SCC_number;
private int bypass_number;

public Vertex(int id, String label, double x, double y){
    this.id = id;
    this.label = label;
    this.x = x;
    this.y = y;
    this.entry_time = 0;
    this.exit_time = 0;
    this.stack_number = 0;
    this.color = CustomColor.WHITE;
    this.SCC_number = 0;
}

public Vertex(int id){
    this.id = id;
    this.label = "" + id;
    this.x = 0.0;
    this.y = 0.0;
    this.entry_time = 0;
    this.exit_time = 0;
    this.stack_number = 0;
    this.color = CustomColor.WHITE;
    this.SCC_number = 0;
}

public int getId(){
    return id;
}

public double getX(){
    return x;
}

public double getY(){
    return y;
}

public void setId(int id){
    this.id = id;
}

public void setX(double x){
    this.x = x;
}

public void setY(double y){
    this.y = y;
}

public int getEntryTime(){
    return entry_time;
}

public CustomColor getColor(){
    return color;
}

public void setEntryTime(int entry_time) {
    this.entry_time = entry_time;
}

```

```

    public void setExitTime(int exit_time) {
        this.exit_time = exit_time;
    }

    public void setColor(CustomColor color) {
        this.color = color;
    }

    public void setStackNumber(int stack_number){
        this.stack_number = stack_number;
    }

    public int getSCCNumber() {
        return SCC_number;
    }

    public void setSCCNumber(int SCC_number){
        this.SCC_number = SCC_number;
    }

    public String getLabel(){
        return label;
    }

    public void setLabel(String label) {
        this.label = label;
    }

    public int getBypassNumber(){
        return bypass_number;
    }

    public void setBypassNumber(int bypass_number) {
        this.bypass_number = bypass_number;
    }
}

```

Operations.java:

```

package algorithm;

public enum Operations {
    ENTER,
    LEAVE,
    NUMBER,
    TRANSPOSE,
    GO,
    COLOR
}

```

SCC.java:

```

package algorithm;
import algorithm.graph.*;

import java.util.ArrayList;

public class SCC {

```

```

        public static ArrayList<ArrayList<Integer>> find_SCC(Graph graph,
StringBuilder log){
            log.append("1 ЭТАП. Расчет порядка выхода вершин.\n");
            boolean[] visited = new boolean[graph.getVertexCount()];
            ArrayList<Integer> exit_order = count_exit_order(graph, visited, log);

            ArrayList<String> exit_order_for_log = new ArrayList<>();
            for (int i = 0; i < exit_order.size(); i++){
                exit_order_for_log.add(graph.getVertexList().get(exit_order.get(i) -
1).getLabel());
            }

            log.append("- Стек: " + exit_order_for_log + "\n");
            log.append("\n2 ЭТАП. Транспонирование графа.\n");
            graph.getSteps().transposeGraph(graph);
            graph.getSteps().addLog(log.toString());
            log.append("- Исходный граф транспонируется\n");
            log.append("\n3 ЭТАП. Формирование КСС обходом в глубину
транспонированного графа.\n");

            Graph transposed_graph = transpose_graph(graph);
            ArrayList<ArrayList<Integer>> result = new ArrayList<>();
            visited = new boolean[graph.getVertexCount()];

            int counter = 0;
            for (int i = exit_order.size() - 1; i >= 0; i--){
                if (!visited[exit_order.get(i) - 1]){
                    counter++;
                    log.append("- Формируется компонента сильной связности " +
counter + "\n");
                    ArrayList<Integer> component = new ArrayList<>();
                    dfs(transposed_graph, exit_order.get(i), visited, component, 0,
log, 1);

                    result.add(component);
                }
            }

            for (int i = 0; i <
transposed_graph.getSteps().getOperationSequence().size(); i++){

graph.getSteps().getOperationSequence().add(transposed_graph.getSteps().getOpera
tionSequence().get(i));

graph.getSteps().addLog(transposed_graph.getSteps().getLogs().get(i));
            }

            for (int i = 0; i < result.size(); i++){
                for (Integer v : result.get(i)){
                    graph.getVertexList().get(v - 1).setSCCNumber(i + 1);
                    transposed_graph.getVertexList().get(v - 1).setSCCNumber(i + 1);
                }
            }

            ArrayList<ArrayList<Integer>> result_for_log = new ArrayList<>();
            for (int i = 0; i < result.size(); i++){
                ArrayList<Integer> comp = new ArrayList<>();
                for (Integer elem : result.get(i)){
                    comp.add(Integer.parseInt(graph.getVertexList().get(elem -
1).getLabel()));
                }
                result_for_log.add(comp);
            }

```

```

        log.append("\nПолученные КС: " + result_for_log + "\n");
        graph.getSteps().getLogs().remove(graph.getSteps().getLogs().size() -
1);
        graph.getSteps().getLogs().add(log.toString());
        graph.getSteps().modifyLogs();

        ArrayList<Vertex> vertexes_for_bypass = (ArrayList<Vertex>)
graph.getVertexList().clone();
        vertexes_for_bypass.sort((v1, v2) -> {
            int cmp = Integer.compare(v1.getEntryTime(), v2.getEntryTime());
            return cmp;
        });
        for (int i = 0; i < vertexes_for_bypass.size(); i++){
            graph.getVertexList().get(vertexes_for_bypass.get(i).getId() -
1).setBypassNumber(i + 1);
        }
        return result_for_log;
    }

    private static ArrayList<Integer> count_exit_order(Graph graph, boolean[]
visited, StringBuilder log){
        ArrayList<Integer> exit_order = new ArrayList<>();
        int time = 0;
        for (int i = 0; i < visited.length; i++){
            if (!visited[i]){
                log.append("- Прыжок в вершину " +
graph.getVertexList().get(i).getLabel() + "\n");
                time++;
                time = dfs(graph, i + 1, visited, exit_order, time, log, 0);
            }
        }
        return exit_order;
    }

    private static int dfs(Graph graph, int vertex, boolean[] visited,
ArrayList<Integer> stack, int time, StringBuilder log, int flag){
        visited[vertex - 1] = true;
        Vertex current_vertex = graph.getVertexList().get(vertex - 1);
        current_vertex.setEntryTime(time);
        current_vertex.setColor(CustomColor.LIGHTGRAY);

        log.append("- Посещена вершина " + current_vertex.getLabel() + "\n");
        graph.getSteps().enterVertex(current_vertex);
        graph.getSteps().addLog(log.toString());
        if (flag == 0){
            graph.getSteps().colorVertex(current_vertex, CustomColor.LIGHTGRAY);
            graph.getSteps().addLog(log.toString());
        } else {
            graph.getSteps().numberVertex(current_vertex);
            graph.getSteps().addLog(log.toString());
        }

        int edge_index = search_edge(graph, vertex, visited, 0, log, flag);
        while (edge_index != -1){
            time++;
            graph.getSteps().leaveVertex(current_vertex);
            graph.getSteps().addLog(log.toString());
            time = dfs(graph, graph.getEdgeList().get(edge_index).getTarget(),
visited, stack, time, log, flag);
            graph.getSteps().enterVertex(current_vertex);
            graph.getSteps().addLog(log.toString());
            edge_index = search_edge(graph, vertex, visited, edge_index, log,
flag);

```



```

    }
    time++;
    stack.add(vertex);
    current_vertex.setExitTime(time);
    current_vertex.setColor(CustomColor.GRAY);
    current_vertex.setStackNumber(graph.getVertexCount() - stack.size());

    graph.getSteps().leaveVertex(current_vertex);
    if (flag == 0) {
        log.append("- Вершина " + current_vertex.getLabel() + " добавлена в
стек\n");
        graph.getSteps().addLog(log.toString());
        graph.getSteps().colorVertex(current_vertex, CustomColor.GRAY);
        graph.getSteps().addLog(log.toString());
    } else {
        log.append("- Вершина " + current_vertex.getLabel() + " принадлежит
текущей КСС\n");
        graph.getSteps().addLog(log.toString());
    }
    return time;
}

private static int search_edge(Graph graph, int vertex, boolean[] visited,
int last_stop, StringBuilder log, int flag){
    for (int i = last_stop; i < graph.getEdgeCount(); i++){
        Edge current_edge = graph.getEdgeList().get(i);
        if (current_edge.getSource() == vertex &&
!visited[current_edge.getTarget() - 1]){
            current_edge.setEdgeType(EdgeType.TREE);
            log.append("- Переход по ребру (" +
graph.getVertexList().get(current_edge.getSource() - 1).getLabel() + ", "
+ graph.getVertexList().get(current_edge.getTarget() -
1).getLabel() + ")\n");
            graph.getSteps().goEdge(current_edge);
            graph.getSteps().addLog(log.toString());
            return i;
        }
        Vertex source_vertex =
graph.getVertexList().get(current_edge.getSource() - 1);
        Vertex target_vertex =
graph.getVertexList().get(current_edge.getTarget() - 1);
        if (source_vertex.getId() == vertex && target_vertex.getColor() ==
CustomColor.LIGHTGRAY
            && current_edge.getEdgeType() == EdgeType.NONE){
            current_edge.setEdgeType(EdgeType.BACK);
            if (flag == 0) {
                log.append("- Ребро (" + source_vertex.getLabel() + ", " +
target_vertex.getLabel() + ") обратное\n");
                graph.getSteps().colorEdge(current_edge, CustomColor.CYAN);
                graph.getSteps().addLog(log.toString());
            }

            } else if (source_vertex.getId() == vertex &&
target_vertex.getColor() == CustomColor.GRAY
            && current_edge.getEdgeType() == EdgeType.NONE){
                if (source_vertex.getEntryTime() <
target_vertex.getEntryTime()){
                    current_edge.setEdgeType(EdgeType.FORWARD);
                    if (flag == 0) {
                        log.append("- Ребро (" + source_vertex.getLabel() + ", "
+ target_vertex.getLabel() + ") направленное вперёд\n");
                        graph.getSteps().colorEdge(current_edge,
CustomColor.BLUE);

```

```

        graph.getSteps().addLog(log.toString());
    }
    } else {
        current_edge.setEdgeType(EdgeType.CROSS);
        if (flag == 0) {
            log.append("- Ребро (" + source_vertex.getLabel() + ", "
+ target_vertex.getLabel() + ") попережное\n");
            graph.getSteps().colorEdge(current_edge,
CustomColor.PURPLE);
            graph.getSteps().addLog(log.toString());
        }
    }
}
return -1;
}

private static Graph transpose_graph(Graph graph) {
    ArrayList<Vertex> new_vertexes = new ArrayList<>();
    for (int i = 0; i < graph.getVertexCount(); i++) {
        new_vertexes.add(new Vertex(
            graph.getVertexList().get(i).getId(),
            graph.getVertexList().get(i).getLabel(),
            graph.getVertexList().get(i).getX(),
            graph.getVertexList().get(i).getY()
        ));
    }

    ArrayList<Edge> new_edges = new ArrayList<>();
    for (int i = 0; i < graph.getEdgeCount(); i++) {
        new_edges.add(new Edge(
            graph.getEdgeList().get(i).getTarget(),
            graph.getEdgeList().get(i).getSource(),
            graph.getEdgeList().get(i).getColor()
        ));
    }
    return new Graph(
        new_edges,
        new_vertexes
    );
}
}

```

StepwiseExecution.java:

```

package algorithm;

import algorithm.graph.*;
import javafx.util.Pair;

import java.util.ArrayList;

public class StepwiseExecution {
    private ArrayList<Pair<Operations, Element>> operation_sequence;
    private ArrayList<CustomColor> colors;
    private ArrayList<String> logs;
    private boolean logs_modified = false;

    public StepwiseExecution() {
        operation_sequence = new ArrayList<>();
        colors = new ArrayList<>();
        logs = new ArrayList<>();
    }
}

```

```

    }

    public ArrayList<Pair<Operations, Element>> getOperationSequence(){
        return operation_sequence;
    }

    public ArrayList<CustomColor> getColors(){
        return colors;
    }

    public ArrayList<String> getLogs() {
        return logs;
    }

    public void enterVertex(Vertex vertex){
        operation_sequence.add(new Pair(Operations.ENTER, vertex));
    }

    public void leaveVertex(Vertex vertex){
        operation_sequence.add(new Pair<>(Operations.LEAVE, vertex));
    }

    public void colorVertex(Vertex vertex, CustomColor color){
        operation_sequence.add(new Pair<>(Operations.COLOR, vertex));
        colors.add(color);
    }

    public void numberVertex(Vertex vertex){
        operation_sequence.add(new Pair<>(Operations.NUMBER, vertex));
    }

    public void goEdge(Edge edge){
        operation_sequence.add(new Pair<>(Operations.GO, edge));
    }

    public void colorEdge(Edge edge, CustomColor color){
        operation_sequence.add(new Pair<>(Operations.COLOR, edge));
        colors.add(color);
    }

    public void transposeGraph(Graph graph){
        operation_sequence.add(new Pair<>(Operations.TRANSPOSE, graph));
    }

    public void addLog(String log){
        logs.add(log);
    }

    public void modifyLogs(){
        if (!logs_modified) {
            String previous_before = logs.get(0);
            for (int i = 1; i < logs.size(); i++) {
                String current_before = logs.get(i);
                logs.set(i, logs.get(i).substring(previous_before.length()));
                previous_before = current_before;
            }
            logs_modified = true;
        }
    }
}

```

AnimationController.java:

```
package application;

import algorithm.graph.Element;
import algorithm.Operations;
import algorithm.graph.*;
import javafx.animation.KeyFrame;
import javafx.animation.PauseTransition;
import javafx.animation.Timeline;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Node;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.shape.Circle;
import javafx.scene.shape.CubicCurve;
import javafx.util.Duration;
import javafx.util.Pair;
import javafx.scene.paint.Color;
import java.util.ArrayList;
import java.util.List;

public class AnimationController {

    private Pane graphCanvas;
    private Timeline timeline;
    private boolean isPaused = false;
    private CustomInterface interfaceApp;
    private int currentStep = 0;
    private List<Runnable> stepActions = new ArrayList<>();

    public AnimationController(Pane pane, CustomInterface interfaceApp){
        graphCanvas = pane;
        this.interfaceApp = interfaceApp;
    }

    private String giveColor(CustomColor color){
        switch (color) {
            case BLACK:
                return "black";
            case GRAY:
                return "gray";
            case LIGHTGRAY:
                return "lightgray";
            case WHITE:
                return "white";
            case GREEN:
                return "green";
            case CYAN:
                return "skyblue";
            case BLUE:
                return "steelblue";
            case PURPLE:
                return "mediumpurple";
            case LIGHTGREEN:
                return "darkseagreen";
            default:
                return "black";
        }
    }
}
```

```

    }

    private void defaultColorAll(Graph graph){
        List<Vertex> vertexes = graph.getVertexList();
        List<Edge> edges = graph.getEdgeList();
        for (Vertex v : vertexes){
            uncolorVertexBackground(v);
            uncolorVertexContour(v);
            removeNumber(v);
        }
        for (Edge e : edges){
            uncolorEdge(e);
        }
    }

    public void prepareStepExecution(Graph graph, int millis){
        stepActions.clear();
        ArrayList<Pair<Operations, Element>> steps =
graph.getSteps().getOperationSequence();
        ArrayList<CustomColor> colors = graph.getSteps().getColors();
        ArrayList<String> logs = graph.getSteps().getLogs();
        int color_counter = 0;
        int flag = 0;
        for (int i = 0; i < steps.size(); i++) {
            final int step_index = i;
            Operations current_operation = steps.get(i).getKey();
            Element element = steps.get(i).getValue();
            final int final_color_counter = color_counter;

            if (current_operation == Operations.COLOR) {
                color_counter++;
            }
            if (current_operation == Operations.TRANSPOSE){
                flag = 1;
            }
            int final_flag = flag;
            stepActions.add(() -> processOperation(current_operation, element,
colors, final_color_counter,
                graph, step_index, logs, millis, final_flag));
        }
    }

    public boolean executeNextStep() {
        if (currentStep < stepActions.size()) {
            stepActions.get(currentStep).run();
            currentStep++;
            return true;
        }
        return false;
    }

    public void resetSteps() {
        currentStep = 0;
        stepActions.clear();
    }

    public void transposeEdges(Graph graph){
        defaultColorAll(graph);
        List<Edge> edges = graph.getEdgeList();
        for (Edge e : edges){
            Node found =
graphCanvas.lookup("#edge_id"+e.getSource()+"_"+e.getTarget());
            if (found instanceof ClassicEdge){

```

```

        ClassicEdge edge = (ClassicEdge) found;
        ClassicEdge transposeEdge = new ClassicEdge(edge.getEndX(),
edge.getEndY(), edge.getStartX(), edge.getStartY());
        StackPane found1 =
(StackPane) edge.getProperties().get("source");
        StackPane found2 =
(StackPane) edge.getProperties().get("target");
        transposeEdge.getProperties().put("source", found2);
        transposeEdge.getProperties().put("target", found1);
        int id1 = (int) found2.getProperties().get("vertex_id");
        int id2 = (int) found1.getProperties().get("vertex_id");
        transposeEdge.setId("edge_id"+id1+"_"+id2);
        graphCanvas.getChildren().removeAll(found);
        graphCanvas.getChildren().add(transposeEdge);
    }
}
graph.transpose();
}

private void uncolorVertexContour(Vertex vertex){
    Node found = graphCanvas.lookup("#vertex_id"+vertex.getId());
    if (found instanceof StackPane){
        StackPane v =(StackPane) found;
        Circle circle = (Circle) v.getChildren().get(0);
        circle.setStyle(circle.getStyle()+"-fx-stroke-width: 3.0;" + "-fx-
stroke: black;");
    }
}

private void uncolorVertexBackground(Vertex vertex){
    Node found = graphCanvas.lookup("#vertex_id"+vertex.getId());
    if (found instanceof StackPane){
        StackPane v =(StackPane) found;
        Circle circle = (Circle) v.getChildren().get(0);
        circle.setStyle(circle.getStyle()+"-fx-fill: white;");
    }
}

private void colorVertexBackground(Vertex vertex, CustomColor customColor){
    String color = giveColor(customColor);
    Node found = graphCanvas.lookup("#vertex_id"+vertex.getId());
    if (found instanceof StackPane){
        StackPane v =(StackPane) found;
        Circle circle = (Circle) v.getChildren().get(0);
        circle.setStyle( circle.getStyle()+"-fx-fill: " + color +""");
    }
}

private void colorVertexContour(Vertex vertex, CustomColor customColor){
    String color = giveColor(customColor);
    Node found = graphCanvas.lookup("#vertex_id"+vertex.getId());
    if (found instanceof StackPane){
        StackPane v =(StackPane) found;
        Circle circle = (Circle) v.getChildren().get(0);
        circle.setStyle( circle.getStyle()+
            "-fx-stroke: " + color +"";
            "-fx-stroke-width: 4.0;");
    }
}

private void colorEdge(Edge edge, CustomColor customColor){
    String color = giveColor(customColor);

```

```

        Node found =
graphCanvas.lookup("#edge_id"+edge.getSource()+"_"+edge.getTarget());
        if (found instanceof ClassicEdge){
            ClassicEdge e = (ClassicEdge) found;
            e.setAnotherColorLines(Color.valueOf(color));
        }
        else if (found instanceof CubicCurve){
            CubicCurve loop = (CubicCurve) found;
            loop.setStroke(Color.valueOf(color));
        }
    }

    private void uncolorEdge(Edge edge){
        Node found =
graphCanvas.lookup("#edge_id"+edge.getSource()+"_"+edge.getTarget());
        if (found instanceof ClassicEdge){
            ClassicEdge e = (ClassicEdge) found;
            e.setAnotherColorLines(Color.BLACK);
        }
        else if (found instanceof CubicCurve){
            CubicCurve loop = (CubicCurve) found;
            loop.setStroke(Color.BLACK);
        }
    }

    private void addNumber(Vertex vertex, int flag){
        Node found = graphCanvas.lookup("#vertex_id"+vertex.getId());
        if (found instanceof StackPane){
            StackPane stackpane = (StackPane) found;
            Label number;
            if (flag == 0){
                number = new Label(String.valueOf(vertex.getBypassNumber()));
            } else {
                number = new Label(String.valueOf(vertex.getSCCNumber()));
            }
            number.setTextFill(Color.BLACK);
            number.setStyle(
                "-fx-text-fill: #f538b9;" +
                "-fx-font-size: 1.6em;" +
                "-fx-font-weight: 900;" );
            StackPane.setAlignment(number, Pos.BOTTOM_RIGHT);
            StackPane.setMargin(number, new Insets(0, 7, 7, 0));
            stackpane.getChildren().add(number);
        }
    }

    private void removeNumber(Vertex vertex) {
        Node found = graphCanvas.lookup("#vertex_id" + vertex.getId());
        if (found instanceof StackPane) {
            StackPane stackPane = (StackPane) found;

            List<Node> nodesToRemove = new ArrayList<>();

            for (Node child : stackPane.getChildren()) {
                if (child instanceof Label) {
                    if (child.getStyle().contains("-fx-text-fill: #f538b9;")) {
                        nodesToRemove.add(child);
                    }
                }
            }
            stackPane.getChildren().removeAll(nodesToRemove);
        }
    }

```

```

private void colorElement(Element element, CustomColor color){
    if (element instanceof Vertex){
        colorVertexBackground((Vertex) element, color);
    } else {
        colorEdge((Edge) element, color);
    }
}

public void visualizeAlgorithm(Graph graph, int millis) throws
InterruptedException {
    if (timeline != null) {
        timeline.stop();
    }
    isPaused = false;
    timeline = new Timeline();
    ArrayList<Pair<Operations, Element>> steps =
graph.getSteps().getOperationSequence();
    ArrayList<CustomColor> colors = graph.getSteps().getColors();
    ArrayList<String> logs = graph.getSteps().getLogs();
    long delay_millis = 0;
    int color_counter = 0;
    int flag = 0;
    for (int i = 0; i < steps.size(); i++){

        Operations current_operation = steps.get(i).getKey();
        if (current_operation == Operations.TRANSPOSE){
            flag = 1;
        }

        int final_color_counter = color_counter;
        int final_i = i;
        int final_flag = flag;
        KeyFrame keyFrame = new KeyFrame(
            Duration.millis(delay_millis),
            e -> processOperation(current_operation,
steps.get(final_i).getValue(),
                                colors, final_color_counter, graph, final_i, logs,
millis, final_flag)

        );
        if (current_operation == Operations.COLOR){
            color_counter++;
        }

        timeline.getKeyFrames().add(keyFrame);
        delay_millis += millis;
    }

    timeline.play();
}

private void processOperation(Operations operation, Element element,
ArrayList<CustomColor> colors, int colorIndex,
                                Graph graph, int i, ArrayList<String> logs,
int millis, int flag) {
    Node center = interfaceApp.getLog().getCenter();
    if (center != null && center instanceof TextArea) {
        TextArea text_area = (TextArea) center;
        if (i == 0) {
            text_area.setText(logs.get(i));
        } else {
            text_area.appendText(logs.get(i));
        }
    }
}

```



```

    }
    text_area.setStyle(text_area.getStyle() +
        "-fx-transition: all 0.3s ease;" +
        "-fx-animation: none;"
    );

    Platform.runLater(() -> {
        text_area.positionCaret(text_area.getText().length());
        text_area.setScrollTop(Double.MAX_VALUE);
    });
}
switch (operation) {
    case ENTER:
        colorVertexContour((Vertex) element, CustomColor.GREEN);
        if (flag == 0){
            addNumber((Vertex) element, flag);
        }
        break;
    case LEAVE:
        uncolorVertexContour((Vertex) element);
        break;
    case NUMBER:
        addNumber((Vertex) element, flag);
        break;
    case TRANSPOSE:
        transposeEdges(graph);
        break;
    case GO:
        Edge edge = (Edge) element;
        Vertex sourceVertex = graph.getVertexList().get(edge.getSource()
- 1);
        Vertex targetVertex = graph.getVertexList().get(edge.getTarget()
- 1);

        colorEdge(edge, CustomColor.GREEN);

        PauseTransition firstHalf = new
PauseTransition(Duration.millis(millis / 2));
        firstHalf.setOnFinished(event1 -> {
            if (sourceVertex != null) {
                uncolorVertexContour(sourceVertex);
            }
            if (targetVertex != null) {
                colorVertexContour(targetVertex, CustomColor.GREEN);
            }
        });

        PauseTransition secondHalf = new
PauseTransition(Duration.millis(millis / 2));
        secondHalf.setOnFinished(event2 -> {
            uncolorEdge(edge);
            if (flag == 0){
                colorEdge(edge, CustomColor.LIGHTGREEN);
            }
        });

        firstHalf.play();
        secondHalf.play();
        break;
    case COLOR:
        colorElement(element, colors.get(colorIndex));

```

```

        break;
    }
}

public void togglePause() {
    if (timeline != null) {
        if (isPaused) {
            timeline.play();
        } else {
            timeline.pause();
        }
        isPaused = !isPaused;
    }
}

public boolean isPaused() {
    return isPaused;
}

public Timeline getTimeline() {
    return timeline;
}
}

```

CustomInterface.java:

```

package application;

import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;

import java.util.Arrays;
import java.util.Objects;

public class CustomInterface {
    private GridPane rootPane;
    private VBox leftButtons;
    private StackPane areaForGraph;
    private BorderPane answer;
    private BorderPane log;
    private Pane graphCanvas;
    private String editStyle;
    private String fileStyle;
    private String playStyle;
    private String resultStyle;

    public CustomInterface() {
        createGridPane();
    }

    private void createGridPane() {
        rootPane = new GridPane();
        rootPane.setStyle("-fx-background-color: #cce4ef;");
    }
}

```

```

ColumnConstraints column1 = new ColumnConstraints();
column1.setPercentWidth(30);
rootPane.getColumnConstraints().add(column1);
RowConstraints row1 = new RowConstraints();
row1.setPercentHeight(58);
rootPane.getRowConstraints().add(row1);

RowConstraints row2 = new RowConstraints();
row2.setPercentHeight(40);
rootPane.getRowConstraints().add(row2);

leftButtons = buttonsForLeftSide();
rootPane.add(leftButtons, 0, 0);
GridPane.setMargin(leftButtons, new Insets(30, 0, 0, 20));

areaForGraph = graphArea();
rootPane.add(areaForGraph, 1, 0);
GridPane.setHgrow(areaForGraph, Priority.ALWAYS);
GridPane.setVgrow(areaForGraph, Priority.ALWAYS);
GridPane.setMargin(areaForGraph, new Insets(30, 20, 0, 20));

answer = textProgramArea("PE3YJlBTAT:");
rootPane.add(answer, 0, 1);
GridPane.setMargin(answer, new Insets(10, 0, 0, 20));

log = textProgramArea("JlOF:");
rootPane.add(log, 1, 1);
GridPane.setMargin(log, new Insets(10, 20, 0, 20));
}

private StackPane graphArea(){
    StackPane stackPane = new StackPane();

    Pane contentPane = new Pane();
    contentPane.setMinSize(2000, 2000);
    graphCanvas = contentPane;

    ScrollPane scrollPane = new ScrollPane();
    scrollPane.setMinHeight(300);
    scrollPane.setMinWidth(730);
    scrollPane.setPannable(true);
    scrollPane.setStyle(
        "-fx-border-color: #0575ad; " +
        "-fx-border-width: 6px; " +
        "-fx-border-radius: 6px; " +
        "-fx-background-radius: 8;" +
        "-fx-border-style: solid;"
    );
    scrollPane.setContent(contentPane);

    Image playImage = new
Image(Objects.requireNonNull(CustomInterface.class.getResourceAsStream("play.png
"))));
    ImageView imageView = new ImageView(playImage);
    imageView.setFitWidth(27);
    imageView.setFitHeight(27);
    imageView.setPreserveRatio(true);
    imageView.setSmooth(true);

    Button playButton = new Button("", imageView);
    playButton.setStyle(
        "-fx-background-radius: 50%; " +

```

```

        "-fx-min-width: 80px; " +
        "-fx-min-height: 80px; " +
        "-fx-max-width: 80px; " +
        "-fx-max-height: 80px; " +
        "-fx-background-color: #5abff2; " +
        "-fx-text-fill: black;" +
        "-fx-border-color: #0575ad; " +
        "-fx-border-width: 4px; " +
        "-fx-border-radius: 50%;"
    );
    playStyle = playButton.getStyle();

    stackPane.getChildren().addAll(scrollPane, playButton);

    StackPane.setAlignment(playButton, Pos.BOTTOM_LEFT);
    StackPane.setMargin(playButton, new Insets(0, 0, 20, 10));

    return stackPane;
}

private VBox buttonsForLeftSide() {
    VBox vbox = new VBox(10);
    vbox.setAlignment(Pos.CENTER);
    Button fastResult = new Button("Получить результат");
    Button stepsResult = new Button("Выполнить по шагам");
    Button addVertex = new Button("Добавить вершину");
    Button addEdge = new Button("Добавить ребро");
    Button deleteSmth = new Button("Удалить элемент");
    Button deleteAll = new Button("Очистить граф");
    Button saveGraph = new Button("Сохранить граф");
    Button loadGraph = new Button("Загрузить граф");

    final String buttonStyle =
        "-fx-background-color: #5abff2;" +
        "-fx-text-fill: black;" +
        "-fx-font-size: 1.2em;" +
        "-fx-font-weight: bold;" +
        "-fx-pref-width: 300px;" +
        "-fx-pref-height: 40px;" +
        "-fx-border-color: #0575ad;" +
        "-fx-border-width: 4px;" +
        "-fx-border-radius: 5px;" +
        "-fx-background-radius: 5px;";

    editStyle = buttonStyle;
    for (Button button : Arrays.asList(fastResult, stepsResult, addVertex,
    addEdge, deleteSmth, deleteAll, saveGraph, loadGraph)) {
        button.setStyle(buttonStyle);
        String text = button.getText();
        if (text.equals("Получить результат") || text.equals("Выполнить по
шарам")) {
            button.setStyle(button.getStyle() + "-fx-background-color:
#0c98df;");
            resultStyle = button.getStyle();
        }
        if (text.equals("Сохранить граф") || text.equals("Загрузить граф")) {
            button.setStyle(button.getStyle() + "-fx-background-color:
#d3effd;");
            fileStyle = button.getStyle();
        }
        button.setMaxWidth(Double.MAX_VALUE);
        button.setMaxHeight(Double.MAX_VALUE);
        VBox.setVgrow(button, Priority.ALWAYS);
    }
}

```

```

        vbox.getChildren().add(button);
    }

    return vbox;
}

private BorderPane textProgramArea(String name) {
    BorderPane borderPane = new BorderPane();

    Label titleLabel = new Label(name);
    titleLabel.setStyle(
        "-fx-alignment: center;" +
        "-fx-font-size: 1.5em;" +
        "-fx-font-weight: 900;" +
        "-fx-text-fill: #0575ad;" +
        "-fx-padding: 7px;" +
        "-fx-background-color: #cce4ef;" +
        "-fx-border-width: 0 0 0 0;"
    );

    borderPane.setTop(titleLabel);

    TextArea text_area = new TextArea();
    text_area.setEditable(false);
    text_area.setWrapText(true);

    text_area.setStyle(
        "-fx-background-color: white;" +
        "-fx-background-radius: 8;" +
        "-fx-border-color: #0575ad;" +
        "-fx-border-radius: 8;" +
        "-fx-border-width: 4.0;" +
        "-fx-padding: 12px;" +
        "-fx-font-size: 1.5em;" +
        "-fx-text-fill: black;" +
        "-fx-cursor: default;"
    );

    if (name.equals("JIOF:")) {
        text_area.textProperty().addListener((observable, oldValue,
newValue) -> {
            Platform.runLater(() -> {
                boolean atBottom = text_area.getScrollTop() +
text_area.getHeight() >=
                text_area.getMaxHeight() - 10;

                if (atBottom) {
                    text_area.positionCaret(text_area.getText().length());
                    text_area.setScrollTop(Double.MAX_VALUE);
                }
            });
        });
    }

    borderPane.setCenter(text_area);
    return borderPane;
}

public GridPane getRootPane() {
    return rootPane;
}

public VBox getLeftButtons() {

```

```

        return leftButtons;
    }

    public BorderPane getAnswer() {
        return answer;
    }

    public BorderPane getLog() {
        return log;
    }

    public StackPane getAreaForGraph() {
        return areaForGraph;
    }

    public Pane getGraphCanvas() {
        return graphCanvas;
    }

    public String giveEditStyle() {
        return editStyle;
    }

    public String giveFileStyle() {
        return fileStyle;
    }

    public String givePlayStyle() {
        return playStyle;
    }

    public String giveResultStyle() {
        return resultStyle;
    }
}

```

GraphJsonUtils.java:

```

package application;

import algorithm.graph.*;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import algorithm.graph.Graph;
import com.google.gson.JsonIOException;
import com.google.gson.JsonSyntaxException;

import java.io.*;
import java.util.HashSet;
import java.util.Set;

public class GraphJsonUtils {

    private static final Gson gson = new GsonBuilder()
        .excludeFieldsWithoutExposeAnnotation()
        .setPrettyPrinting()
        .create();

    public static void saveGraph(Graph graph, String filePath) throws
    IOException {
        try (FileWriter writer = new FileWriter(filePath)) {
            gson.toJson(graph, writer);
        }
    }
}

```

```

    }
}

public static Graph loadGraph(String filePath) throws IOException {
    try (FileReader reader = new FileReader(filePath)) {
        Graph graph = gson.fromJson(reader, Graph.class);
        if (graph == null) throw new IOException("Graph is null");
        if (graph.getEdgeList() == null) throw new IOException("Edges list
is null");
        if (graph.getVertexList() == null) throw new IOException("Vertexes
list is null");
        Set<Integer> vertexIds = new HashSet<>();
        for (Vertex v : graph.getVertexList()) {
            if (v.getId() <= 0) throw new IOException("Vertex ID is null");
            if (v.getId() > graph.getVertexCount()) throw new
IOException("Vertex ID is null");
            if (!vertexIds.add(v.getId())) throw new IOException("Duplicate
vertex ID: ");
            if (v.getLabel() == null) throw new IOException("Vertex ID is
null");
            if (v.getColor() == null) throw new IOException("Vertex ID is
null");
            if (v.getX() <=0.0) throw new IOException("Vertex ID is null");
            if (v.getY() <= 0.0) throw new IOException("Vertex ID is null");
        }
        for (Edge e : graph.getEdgeList()) {
            int from = e.getSource();
            int to = e.getTarget();
            if (from>graph.getVertexCount() | to > graph.getVertexCount())
throw new IOException("Vertex ID is null");
            if (from<=0 | to<=0) throw new IOException("Vertex ID is null");
            if (e.getEdgeType()==null) throw new IOException("Vertex ID is
null");
            if (e.getColor()==null) throw new IOException("Vertex ID is
null");
            if (graph.getVertexList().get(from-1) == null) throw new
IOException("Edge source vertex not found");
            if (graph.getVertexList().get(to-1)==null) throw new
IOException("Edge target vertex not found");
        }
        graph.sortLists();
        return graph;
    } catch (JsonSyntaxException | JsonIOException e){
        throw new IOException(e);
    }
    catch (IOException e){
        throw new IOException(e);
    }
}
}

```

Main.java:

```

package application;

import algorithm.SCC;

```

```

import algorithm.graph.CustomColor;
import algorithm.graph.Edge;
import algorithm.graph.Graph;
import algorithm.graph.Vertex;
import javafx.animation.PauseTransition;
import javafx.application.Application;
import javafx.geometry.Bounds;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.layout.*;
import javafx.scene.shape.CubicCurve;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import javafx.scene.image.ImageView;
import javafx.util.Duration;

import java.io.File;
import java.io.IOException;
import java.util.*;

import static application.GraphJsonUtils.loadGraph;

public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        settingsInterface(primaryStage);
        primaryStage.show();
    }

    private void settingsInterface(Stage primaryStage){
        primaryStage.setTitle("Kosaraju's Algorithm");
        primaryStage.setWidth(1100);
        primaryStage.setHeight(800);
        primaryStage.setMinWidth(1100);
        primaryStage.setMinHeight(800);

        CustomInterface interfaceApp = new CustomInterface();
        InterfaceController controller = new InterfaceController(interfaceApp,
primaryStage);
        GridPane rootPane = interfaceApp.getRootPane();

        Scene scene = new Scene(rootPane, 900, 700);
        primaryStage.setScene(scene);
    }
}

class InterfaceController{
    Stage primaryStage;
    private CustomInterface interfaceApp;
    private Pane graphCanvas;
    private Button deleteAll;

```



```

private Button createVertex;
private Button createEdge;
private Button deleteSmth;
private Button fastResult;
private Button stepsResult;
private Button saveGraph;
private Button loadGraph;
private Button playPause;
String styleActiveButton;
Button activeButton;
boolean isEditMode = false;
boolean isAlgorithm = false;
boolean isFinishAlgoWithTranspose = false;
private double r = 27;
private AnimationController animationController;
private Graph baseGraph = null;
private int max_vertexes = 20;
private Graph graphForStart;
private Image playImage;
private Image pauseImage;
private boolean stepByStepMode = false;
private boolean isFast = false;

public InterfaceController(CustomInterface interfaceApp, Stage stage){
    this.primaryStage = stage;
    this.interfaceApp = interfaceApp;
    graphCanvas = interfaceApp.getGraphCanvas();
    controlAllButtons();
    animationController = new AnimationController(graphCanvas,
interfaceApp);
}

private void controlAllButtons(){
    unpackButtons();
    controlDeleteAllBtn();
    controlCreateVertexBtn();
    controlCreateEdgeBtn();
    controlDeleteSmthBtn();
    controlLoadGraphBtn();
    controlSaveGraphBtn();
    controlFastResultBtn();
    controlPlayButton();
    controlStepsResultBtn();
}

private void unpackButtons(){
    VBox leftButtons = interfaceApp.getLeftButtons();
    for (Node node : leftButtons.getChildren()) {
        if (node instanceof Button) {
            Button button = (Button) node;
            String text = button.getText();
            if (text.equals("Очистить граф")){
                deleteAll = button;
            }
            else if (text.equals("Добавить вершину")){
                createVertex = button;
            }
            else if (text.equals("Добавить ребро")){
                createEdge = button;
            }
            else if (text.equals("Удалить элемент")){
                deleteSmth = button;
            }
        }
    }
}

```

```

        else if (text.equals("Получить результат")){
            fastResult = button;
        }
        else if (text.equals("Выполнить по шагам")){
            stepsResult = button;
        }
        else if (text.equals("Сохранить граф")){
            saveGraph = button;
        }
        else if (text.equals("Загрузить граф")){
            loadGraph = button;
        }
    }
}
StackPane areaForGraph = interfaceApp.getAreaForGraph();
playPause = (Button) areaForGraph.getChildren().get(1);
}

private void checkEditMode(){
    if (activeButton == null) return;
    String text = activeButton.getText();
    if (isEditMode){
        isEditMode = false;
        if (text.equals("Добавить вершину")){
            createVertex.setStyle(styleActiveButton);
            graphCanvas.setOnMouseClicked(null);
        }
        else if (text.equals("Добавить ребро")){
            createEdge.setStyle(styleActiveButton);
            graphCanvas.setOnMouseClicked(null);
        }
        else if (text.equals("Удалить элемент")){
            deleteSmth.setStyle(styleActiveButton);
            graphCanvas.setOnMouseClicked(null);
        }
    }
}

private void controlPlayButton(){
    ImageView playPauseImageView = (ImageView) playPause.getGraphic();
    playImage = new
Image(Objects.requireNonNull(getClass().getResourceAsStream("play.png")));
    pauseImage = new
Image(Objects.requireNonNull(getClass().getResourceAsStream("pause.png")));

    playPause.setOnAction(e->{
        checkEditMode();
        if (!stepByStepMode){
            if (isFast){
                isFast = false;
            }
            if (!graphCanvas.getChildren().isEmpty()){
                if (isFinishAlgoWithTranspose){
                    isFinishAlgoWithTranspose = false;
                    if (baseGraph != null){
                        animationController.transposeEdges(baseGraph);
                        baseGraph = createGraph(graphCanvas);
                        drawGraphFromLoad(baseGraph);
                    }
                }
            }
            if (isAlgorithm) {
                animationController.togglePause();
                if (animationController.isPaused()) {

```

```

        fastResult.setStyle(interfaceApp.giveResultStyle());
        playPauseImageView.setImage(playImage);
    } else {
        playPauseImageView.setImage(pauseImage);
    }
} else {
    changeColorNon();
    stepsResult.setStyle(interfaceApp.giveResultStyle()+"-
fx-background-color: #b7c2c2;"+"-fx-border-color: #6a6b71;");
    fastResult.setStyle(interfaceApp.giveResultStyle()+"-fx-
background-color: #b7c2c2;"+"-fx-border-color: #6a6b71;");
    String log_message = "";
    Node center1 = interfaceApp.getLog().getCenter();
    if (center1 != null && center1 instanceof TextArea) {
        ((TextArea) center1).setText(log_message);
    }
    String result_message = "";
    Node center2 = interfaceApp.getAnswer().getCenter();
    if (center2 != null && center2 instanceof TextArea) {
        ((TextArea) center2).setText(result_message);
    }
    isAlgorithm = true;
    playPauseImageView.setImage(pauseImage);
    for (Node node : graphCanvas.getChildren()) {
        if (node instanceof StackPane) {
            StackPane stackpane = (StackPane) node;
            if (stackpane.getChildren().size() == 3){
                stackpane.getChildren().remove(2);
            }
        }
    }
    baseGraph= createGraph(graphCanvas);
    graphForStart = createGraph(graphCanvas);
    StringBuilder log = new StringBuilder();
    ArrayList<ArrayList<Integer>> result =
    SCC.find_SCC(baseGraph, log);
    try {

        animationController.visualizeAlgorithm(baseGraph,
500);

    } catch (InterruptedException ex) {
        throw new RuntimeException(ex);
    }

    animationController.getTimeline().setOnFinished(event ->
{

    playPauseImageView.setImage(playImage);
    isAlgorithm = false;
    String result_string = "";
    for (ArrayList<Integer> component : result){
        result_string += component + "\n";
    }
    Node center = interfaceApp.getAnswer().getCenter();
    if (center != null && center instanceof TextArea) {
        ((TextArea) center).setText(result_string);
    }
    isFinishAlgoWithTranspose = true;
    changeColorDefault();

    stepsResult.setStyle(interfaceApp.giveResultStyle());
    fastResult.setStyle(interfaceApp.giveResultStyle());
    });
}

```

```

    }
}

});
}

private void controlStepsResultBtn(){
    String style = stepsResult.getStyle();
    stepsResult.setOnAction(e -> {
        checkEditMode();
        if (!isAlgorithm) {
            if (!graphCanvas.getChildren().isEmpty()) {
                if (isFinishAlgoWithTranspose){
                    isFinishAlgoWithTranspose = false;
                    if (baseGraph != null){
                        animationController.transposeEdges(baseGraph);
                        baseGraph = createGraph(graphCanvas);
                        drawGraphFromLoad(baseGraph);
                    }
                }
            }
            if (isFast){
                isFast = false;
            }

            if (!stepByStepMode){
                changeColorNon();
                playPause.setStyle(interfaceApp.givePlayStyle()+"-fx-
background-color: #b7c2c2;"+ "-fx-border-color: #6a6b71;");
                String log_message = "";
                Node center1 = interfaceApp.getLog().getCenter();
                if (center1 != null && center1 instanceof TextArea) {
                    ((TextArea) center1).setText(log_message);
                }
                String result_message = "";
                Node center2 = interfaceApp.getAnswer().getCenter();
                if (center2 != null && center2 instanceof TextArea) {
                    ((TextArea) center2).setText(result_message);
                }
                for (Node node : graphCanvas.getChildren()) {
                    if (node instanceof StackPane) {
                        StackPane stackpane = (StackPane) node;
                        if (stackpane.getChildren().size() == 3){
                            stackpane.getChildren().remove(2);
                        }
                    }
                }
                stepByStepMode = true;
                stepsResult.setText("Следующий шаг");
                stepsResult.setStyle(style+ "-fx-background-color:
#0575ad;"+ "-fx-border-color: #081459;");

                baseGraph = createGraph(graphCanvas);

                graphForStart = createGraph(graphCanvas);

                StringBuilder log = new StringBuilder();
                SCC.find_SCC(baseGraph, log);

                animationController.resetSteps();
                animationController.prepareStepExecution(baseGraph, 1);

                boolean has_more_steps = true;
                for (int i = 0; i < 4; i++) {

```



```

        if (stepByStepMode)
playPause.setStyle(interfaceApp.givePlayStyle());
        if (isAlgorithm)
stepsResult.setStyle(interfaceApp.giveResultStyle());

        Node answer = interfaceApp.getAnswer().getCenter();
        Node log = interfaceApp.getLog().getCenter();
        if (answer instanceof TextArea && log instanceof TextArea) {
            TextArea answerTextArea = (TextArea) answer;
            TextArea logTextArea = (TextArea) log;
            answerTextArea.clear();
            logTextArea.clear();
        }
        graphCanvas.getChildren().clear();
        drawGraphFromLoad(graphForStart);
        if (isAlgorithm) isAlgorithm = false;
        else if (stepByStepMode) {
            stepByStepMode = false;
            stepsResult.setText("Выполнить по шагам");
            stepsResult.setStyle(stepsResult.getStyle()+"-fx-
background-color: #0c98df;");
        }
        isFinishAlgoWithTranspose= false; //?
    }
    fastResult.setStyle(fastResult.getStyle() + "-fx-background-
color: #0575ad;"+"-fx-border-color: #081459;");
    for (Node node : graphCanvas.getChildren()) {
        if (node instanceof StackPane) {
            StackPane stackpane = (StackPane) node;
            if (stackpane.getChildren().size() == 3) {
                stackpane.getChildren().remove(2);
            }
        }
    }
    graph = createGraph(graphCanvas);
    StringBuilder log = new StringBuilder();
    ArrayList<ArrayList<Integer>> result = SCC.find_SCC(graph, log);
    Node center1 = interfaceApp.getLog().getCenter();
    if (center1 != null && center1 instanceof TextArea) {
        ((TextArea) center1).setText(log.toString());
    }
    String result_string = "";
    for (ArrayList<Integer> component : result) {
        result_string += component + "\n";
    }
    Node center2 = interfaceApp.getAnswer().getCenter();
    if (center2 != null && center2 instanceof TextArea) {
        ((TextArea) center2).setText(result_string);
    }
    List<Vertex> vertexes = graph.getVertexList();
    for (Vertex v : vertexes) {
        Node found = graphCanvas.lookup("#vertex_id" + v.getId());
        if (found instanceof StackPane) {
            StackPane stackpane = (StackPane) found;
            Label numberSCC = new
Label(String.valueOf(v.getSCCNumber()));
            numberSCC.setTextFill(Color.BLACK);
            numberSCC.setStyle(
                "-fx-text-fill: #f538b9;" +
                "-fx-font-size: 1.6em;" +
                "-fx-font-weight: 900;");
            StackPane.setAlignment(numberSCC, Pos.BOTTOM_RIGHT);
            StackPane.setMargin(numberSCC, new Insets(0, 7, 7, 0));

```

```

        stackpane.getChildren().add(numberSCC);
    }
}
isFast = true;
}
});
fastResult.setOnMouseReleased(e -> {
    if (animationController.isPaused() || isFast)
fastResult.setStyle(style);
});
}

private void changeColorDefault(){
    createEdge.setStyle(interfaceApp.giveEditStyle());
    createVertex.setStyle(interfaceApp.giveEditStyle());
    deleteAll.setStyle(interfaceApp.giveEditStyle());
    deleteSmth.setStyle(interfaceApp.giveEditStyle());

    saveGraph.setStyle(interfaceApp.giveFileStyle());
    loadGraph.setStyle(interfaceApp.giveFileStyle());
}

private void changeColorNon(){
    createEdge.setStyle(interfaceApp.giveEditStyle()+"-fx-background-color:
#b7c2c2;"+"-fx-border-color: #6a6b71;");
    createVertex.setStyle(interfaceApp.giveEditStyle()+"-fx-background-
color: #b7c2c2;"+"-fx-border-color: #6a6b71;");
    deleteAll.setStyle(interfaceApp.giveEditStyle()+"-fx-background-color:
#b7c2c2;"+"-fx-border-color: #6a6b71;");
    deleteSmth.setStyle(interfaceApp.giveEditStyle()+"-fx-background-color:
#b7c2c2;"+"-fx-border-color: #6a6b71;");

    saveGraph.setStyle(interfaceApp.giveFileStyle()+"-fx-background-color:
#b7c2c2;"+"-fx-border-color: #6a6b71;");
    loadGraph.setStyle(interfaceApp.giveFileStyle()+"-fx-background-color:
#b7c2c2;"+"-fx-border-color: #6a6b71;");
}
private void controlDeleteAllBtn(){
    String style = deleteAll.getStyle();
    deleteAll.setOnMousePressed(e->{
        if (!isAlgorithm && !stepByStepMode){
            checkEditMode();
            isFinishAlgoWithTranspose = false;
            graphCanvas.getChildren().clear();
            deleteAll.setStyle(deleteAll.getStyle() + "-fx-background-color:
#0575ad;"+"-fx-border-color: #081459;");
            String log_message = "";
            Node center1 = interfaceApp.getLog().getCenter();
            if (center1 != null && center1 instanceof TextArea) {
                ((TextArea) center1).setText(log_message);
            }
            String result_message = "";
            Node center2 = interfaceApp.getAnswer().getCenter();
            if (center2 != null && center2 instanceof TextArea) {
                ((TextArea) center2).setText(result_message);
            }
        }
    });
    deleteAll.setOnMouseReleased(e->{
        if (!isAlgorithm && !stepByStepMode) deleteAll.setStyle(style);
    });
}
}

```

```

private void controlDeleteSmthBtn(){
    deleteSmth.setOnAction(e->{
        if (!isAlgorithm && !stepByStepMode){ // no algo
            if (activeButton != deleteSmth){
                checkEditMode();
                activeButton = deleteSmth;
                styleActiveButton = deleteSmth.getStyle();
            }
            isEditMode = !isEditMode;
            if (isEditMode) {
                deleteSmth.setStyle(deleteSmth.getStyle()+ "-fx-background-
color: #0575ad;"+"-fx-border-color: #081459;");
                graphCanvas.setOnMouseClicked(event ->
                {
                    double x = event.getX();
                    double y = event.getY();
                    List<Node> nodesToRemove = new ArrayList<>();
                    for (Node node : graphCanvas.getChildren()) {
                        if (node instanceof StackPane) {
                            Bounds bounds = node.getBoundsInParent();
                            if (bounds.contains(x, y)) {
                                nodesToRemove.add(node);
                                for (Node edges :
graphCanvas.getChildren()){
                                    if (edges instanceof CubicCurve){
                                        CubicCurve edge = (CubicCurve)
edges;
                                        if
(edge.getProperties().containsKey("source")
                                        &&
edge.getProperties().get("source") == node) {
                                            nodesToRemove.add(edge);
                                        }
                                    }
                                    else if (edges instanceof ClassicEdge){
                                        ClassicEdge edge = (ClassicEdge)
edges;
                                        if
(edge.getProperties().containsKey("source")
                                        &&
edge.getProperties().get("source") == node) {
                                            nodesToRemove.add(edge);
                                        }
                                    }
                                    else if
(edge.getProperties().containsKey("target")
                                        &&
edge.getProperties().get("target") == node){
                                            nodesToRemove.add(edge);
                                        }
                                    }
                                }
                            }
                        }
                    }
                    else if (node.contains(x, y)) {
                        nodesToRemove.add(node);
                    }
                }
                graphCanvas.getChildren().removeAll(nodesToRemove);
            });
        }
    }
    else {
        deleteSmth.setStyle(styleActiveButton);
        activeButton = null;
    }
}

```



```

        graphCanvas.setOnMouseClicked(null);
    }
}
});
}

private void controlCreateVertexBtn(){
    createVertex.setOnAction(e ->
    {
        if (!isAlgorithm && !stepByStepMode){

            if (activeButton != createVertex ){
                checkEditMode();
                activeButton = createVertex;
                styleActiveButton = createVertex.getStyle();
            }

            isEditMode = !isEditMode;
            if (isEditMode) {
                createVertex.setStyle(createVertex.getStyle() + "-fx-
background-color: #0575ad;"+ "-fx-border-color: #081459;");
                graphCanvas.setOnMouseClicked(event ->
                {
                    int[] lb = new int[100];
                    for (int i = 1; i < 101; i++){
                        lb[i-1] = i;
                    }
                    int vertex_counter = 1;
                    for (Node node : graphCanvas.getChildren()) {
                        if (node instanceof StackPane) {
                            if (vertex_counter >= max_vertexes){
                                Alert alert = createAlert("Максимальное
количество вершин = " + max_vertexes + "!");
                                alert.setAlertType(Alert.AlertType.WARNING);
                                alert.setTitle("Превышено ограничение");
                                alert.showAndWait();
                                return;
                            }
                            vertex_counter++;
                            Text textNode = (Text) ((StackPane)
node).getChildren().get(1);

                            String str = textNode.getText();

                            int val = Integer.parseInt(str);
                            lb[val - 1] = 1000;
                        }
                    }
                    OptionalInt new_label = Arrays.stream(lb).min();

                    StackPane container =
createContainerVertex(event.getX(), event.getY(), "" + new_label.getAsInt());

                    Text text = (Text) container.getChildren().get(1);

                    container.getProperties().put("vertex_x", event.getX());
                    container.getProperties().put("vertex_y", event.getY());
                    container.getProperties().put("vertex_label",
text.getText());

                    graphCanvas.getChildren().add(container);
                });
            }
            else {

```

```

        createVertex.setStyle(styleActiveButton);
        activeButton = null;
        graphCanvas.setOnMouseClicked(null);
    }
}
});
}

private void controlCreateEdgeBtn(){
    createEdge.setOnAction(e->{
        if (!isAlgorithm && !stepByStepMode){ // no algo
            if (activeButton != createEdge){
                checkEditMode();
                activeButton = createEdge;
                styleActiveButton = createEdge.getStyle();
            }

            isEditMode = !isEditMode;
            if (isEditMode) {
                createEdge.setStyle(createEdge.getStyle()+ "-fx-background-
color: #0575ad;"+ "-fx-border-color: #081459;");
                final StackPane[] selectedNode = {null};
                graphCanvas.setOnMouseClicked(event ->
                {
                    StackPane vertex = findStackPaneAt(event.getX(),
event.getY());
                    if (vertex != null){
                        String style = "-fx-fill: white; " +
                            "-fx-stroke: black; " +
                            "-fx-stroke-width: 3.0;";
                        if (selectedNode[0] == null) {
                            selectedNode[0] = vertex;
                            vertex.getChildren().get(0).setStyle(style+"-fx-
stroke: #0d3bc5;"+ "-fx-stroke-width: 6.0;");
                        }
                        else {
                            StackPane source = selectedNode[0];
                            StackPane target = vertex;
                            if (source == target){
                                double centerX = source.getLayoutX() +
source.getWidth() / 2;
                                double centerY = source.getLayoutY() +
source.getHeight() / 2;
                                CubicCurve loop = drawLoopEdge(centerX,
centerY, r);
                                loop.getProperties().put("source", vertex);
                                loop.getProperties().put("target", vertex);
                                graphCanvas.getChildren().add(loop);
                            }
                            else{
                                double[] coordinates =
findCoordForEdge(source, target);
                                ClassicEdge edge = new
ClassicEdge(coordinates[0], coordinates[1], coordinates[2], coordinates[3]);
                                edge.getProperties().put("source", source);
                                edge.getProperties().put("target", target);
                                graphCanvas.getChildren().add(edge);
                            }
                        }
                    }
                });
                selectedNode[0].getChildren().get(0).setStyle(style);
                selectedNode[0] = null;
            }
        }
    });
}

```

```

        }
    }
    });
}
else {
    createEdge.setStyle(styleActiveButton);
    activeButton = null;
    graphCanvas.setOnMouseClicked(null);
}
}
});
}

private void controlLoadGraphBtn() {
    String style = loadGraph.getStyle();
    loadGraph.setOnMousePressed(e -> {
        if (!isAlgorithm && !stepByStepMode) {
            checkEditMode();
            PauseTransition pause = new
PauseTransition(Duration.millis(70));
            loadGraph.setStyle(loadGraph.getStyle() + "-fx-background-color:
#0575ad;"+ "-fx-border-color: #081459;");
            pause.setOnFinished(event -> {
                FileChooser fileChooser = new FileChooser();
                fileChooser.setTitle("Выберите файл с графом (json файл)");
                FileChooser.ExtensionFilter filter = new
FileChooser.ExtensionFilter("json файлы", "*.json");
                fileChooser.getExtensionFilters().add(filter);
                File selectedFile =
fileChooser.showOpenDialog(primaryStage);

                if (selectedFile != null) {
                    String filePath = selectedFile.getAbsolutePath();
                    try {
                        Graph graph = loadGraph(filePath);
                        drawGraphFromLoad(graph);
                        isFinishAlgoWithTranspose = false;
                    } catch (Exception ex) {
                        Alert alert = createAlert("Ошибка при
загрузке.\nПопробуйте другой файл.");
                        alert.setAlertType(Alert.AlertType.WARNING);
                        alert.initOwner(primaryStage);
                        alert.showAndWait();
                    }
                }
                loadGraph.setStyle(style);
            });
            pause.play();
        }
    });
}

private void controlSaveGraphBtn() {
    String style = saveGraph.getStyle();
    saveGraph.setOnAction(e -> {
        if (!isAlgorithm && !stepByStepMode) {
            checkEditMode();
            saveGraph.setStyle(saveGraph.getStyle() + "-fx-background-color:
#0575ad;"+ "-fx-border-color: #081459;");
            PauseTransition pause = new
PauseTransition(Duration.millis(70));
            pause.setOnFinished(event -> {
                Graph graph = createGraph(graphCanvas);

```

```

        String filename = "graph";
        for (int i = 0; i < 10; i++){
            int random_num = new Random().nextInt(10);
            filename += random_num;
        }
        filename += ".json";
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Сохранить файл");
        fileChooser.setInitialFileName(filename);

        fileChooser.getExtensionFilters().addAll(
            new FileChooser.ExtensionFilter("json", "*.json")
        );

        File file = fileChooser.showSaveDialog(primaryStage);
        if (file != null) {
            try {
                String fullPath = file.getAbsolutePath();
                GraphJsonUtils.saveGraph(graph, fullPath);
                Alert alert = createAlert("Граф успешно сохранён!");
                alert.initOwner(primaryStage);
                alert.show();
            } catch (IOException ex) {
                Alert alert = createAlert("Ошибка сохранения
файла!");

                alert.setAlertType(Alert.AlertType.WARNING);
                alert.initOwner(primaryStage);
                alert.show();
            }
        }
        saveGraph.setStyle(style);
    });
    pause.play();
}

});
}

private StackPane createContainerVertex(double x, double y, String label){
    Circle circle = new Circle(r);
    circle.setStyle(
        "-fx-fill: white; " +
        "-fx-stroke: black; " +
        "-fx-stroke-width: 3.0;"
    );

    Text text = new Text(label);
    text.setStyle("-fx-font-size: 1.3em;");

    StackPane container = new StackPane();
    container.getChildren().addAll(circle, text);

    container.setLayoutX(x - r);
    container.setLayoutY(y - r);

    container.getProperties().put("vertex_x", x);
    container.getProperties().put("vertex_y", y);
    container.getProperties().put("vertex_label", text.toString());
    return container;
}

private Alert createAlert(String message){
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setHeaderText(null);

```

```

        Text text = new Text(message);
        text.setStyle("-fx-font-size: 14px; -fx-fill: #0575ad; -fx-font-weight:
bold;");
        StackPane container = new StackPane(text);
        container.setPadding(new Insets(10));
        container.setAlignment(Pos.CENTER);
        alert.getDialogPane().setContent(container);
        DialogPane dialogPane = alert.getDialogPane();
        dialogPane.setStyle(
            "-fx-background-color: #cce4ef;" +
            "-fx-border-color: #0575ad;" +
            "-fx-border-width: 3px;"
        );
        ButtonType buttonOk = new ButtonType("Ok", ButtonBar.ButtonData.OTHER);
        alert.getButtonTypes().setAll(buttonOk);
        Button btnOk= (Button) alert.getDialogPane().lookupButton(buttonOk);
        String buttonStyle =
            "-fx-border-color: #0575ad;" +
            "-fx-border-width: 2px;" +
            "-fx-border-radius: 5px;" +
            "-fx-text-fill: #0575ad;" +
            "-fx-font-weight: bold;" +
            "-fx-background-radius: 5px;";
        btnOk.setStyle(buttonStyle);
        btnOk.setFocusTraversable(false);
        return alert;
    }

    private static Graph createGraph(Pane graphCanvas){
        ArrayList<Vertex> vertexes = new ArrayList<>();
        ArrayList<Edge> edges = new ArrayList<>();

        int[] lb = new int[100];
        for (int i = 1; i < 101; i++){
            lb[i - 1] = i;
        }
        int counter = 0;
        for (Node node : graphCanvas.getChildren()){
            if (node instanceof StackPane) {
                counter++;
                StackPane container = (StackPane) node;
                String label = (String)
container.getProperties().get("vertex_label");
                lb[Integer.parseInt(label) - 1] = 0;
            }
        }
        int flag = 0;
        for (int i = 0; i < counter; i++){
            if (lb[i] != 0){
                flag = 1;
                break;
            }
        }

        counter = 0;
        for (Node node : graphCanvas.getChildren()) {
            if (node instanceof StackPane) {
                counter++;
                StackPane container = (StackPane) node;
                double x = (Double) container.getProperties().get("vertex_x");
                double y = (Double) container.getProperties().get("vertex_y");
                String label = (String)
container.getProperties().get("vertex_label");

```

```

        int id = Integer.parseInt(label);
        if (flag == 1){
            id = counter;
        }
        container.getProperties().put("vertex_id", id);
        container.setId("vertex_id"+id);

        Vertex vertex = new Vertex(id, label, x, y);
        vertexes.add(vertex);
    }
}
for (Node node : graphCanvas.getChildren()) {
    if (node instanceof ClassicEdge || node instanceof CubicCurve) {
        StackPane source = (StackPane)
node.getProperties().get("source");
        StackPane target = (StackPane)
node.getProperties().get("target");

        if (source != null && target != null) {
            Integer source_id = (Integer)
source.getProperties().get("vertex_id");
            Integer target_id = (Integer)
target.getProperties().get("vertex_id");
            node.setId("edge_id"+source_id+"_"+target_id);
            Edge edge = new Edge(source_id, target_id,
CustomColor.BLACK);
            edges.add(edge);
        }
    }
}
return new Graph(edges, vertexes);
}

private void drawGraphFromLoad(Graph graph){
    if (!(graphCanvas.getChildren().isEmpty())){
        graphCanvas.getChildren().clear();
    }
    List<Vertex> vertexes = graph.getVertexList();
    List<Edge> edges = graph.getEdgeList();
    List<StackPane> drawVertexes = new ArrayList<StackPane>();
    for (Vertex v : vertexes){
        double x = v.getX();
        double y = v.getY();
        StackPane container = createContainerVertex(x, y, v.getLabel());
        Text text = (Text) container.getChildren().get(1);

        container.getProperties().put("vertex_x", x);
        container.getProperties().put("vertex_y", y);
        container.getProperties().put("vertex_label", text.getText());
        drawVertexes.add(container);
        graphCanvas.getChildren().add(container);
    }

    for (Edge e : edges){
        int target = e.getTarget() - 1;
        int source = e.getSource() - 1;
        Vertex v1 = vertexes.get(source);
        Vertex v2 = vertexes.get(target);
        if (v1 != v2){
            double[] coord = findCoordFroDrawEdge(v1, v2);
            ClassicEdge edge = new ClassicEdge(coord[0], coord[1], coord[2],
coord[3]);
            edge.getProperties().put("source", drawVertexes.get(source));

```

```

        edge.getProperties().put("target", drawVertexes.get(target));
        graphCanvas.getChildren().add(edge);
    }
    else{
        CubicCurve loop = drawLoopEdge(v1.getX(), v1.getY(), r);
        loop.getProperties().put("source", drawVertexes.get(source));
        loop.getProperties().put("target", drawVertexes.get(target));
        graphCanvas.getChildren().add(loop);
    }
}

private double[] findCoordFroDrawEdge(Vertex v1, Vertex v2){
    double dx = v2.getX() - v1.getX();
    double dy = v2.getY() - v1.getY();
    double distance = Math.hypot(dx, dy);
    double unitX = dx / distance;
    double unitY = dy / distance;

    double startX = v1.getX() + unitX * r;
    double startY = v1.getY() + unitY * r;
    double endX = v2.getX() - unitX * r;
    double endY = v2.getY() - unitY * r;

    return new double[]{startX, startY, endX, endY};
}

private StackPane findStackPaneAt(double x, double y) {
    for (Node node : graphCanvas.getChildren()) {
        if (node instanceof StackPane) {
            Bounds bounds = node.getBoundsInParent();
            if (bounds.contains(x, y)) {
                return (StackPane) node;
            }
        }
    }
    return null;
}

private double[] findCoordForEdge(StackPane source, StackPane target){
    double centerX1 = source.getLayoutX() + source.getWidth() / 2;
    double centerY1 = source.getLayoutY() + source.getHeight() / 2;
    double centerX2 = target.getLayoutX() + target.getWidth() / 2;
    double centerY2 = target.getLayoutY() + target.getHeight() / 2;

    double dx = centerX2 - centerX1;
    double dy = centerY2 - centerY1;
    double distance = Math.hypot(dx, dy);
    double unitX = dx / distance;
    double unitY = dy / distance;

    double startX = centerX1 + unitX * r;
    double startY = centerY1 + unitY * r;
    double endX = centerX2 - unitX * r;
    double endY = centerY2 - unitY * r;

    return new double[]{startX, startY, endX, endY};
}

private CubicCurve drawLoopEdge(double centerX, double centerY, double r){
    double angle = 45;
    double startX = centerX - r * Math.cos(Math.toRadians(angle));
    double startY = centerY + r * Math.sin(Math.toRadians(angle));

```

```

        double controlX1 = startX + 50 * Math.cos(Math.toRadians(180));
        double controlY1 = startY - 50 * Math.sin(Math.toRadians(180));

        double controlX2 = startX + 50 * Math.cos(Math.toRadians(angle + 180 +
45));
        double controlY2 = startY - 50 * Math.sin(Math.toRadians(angle + 180 +
45));

        CubicCurve loop = new CubicCurve(
            startX, startY,
            controlX1, controlY1,
            controlX2, controlY2,
            startX, startY
        );
        loop.setStroke(Color.BLACK);
        loop.setFill(null);
        loop.setStrokeWidth(4);
        return loop;
    }
}

class ClassicEdge extends Group{
    private Line line;
    private Line leftWing;
    private Line rightWing;
    private double startX;
    private double startY;
    private double endX;
    private double endY;

    public ClassicEdge(double startX, double startY, double endX, double endY){
        this.startX = startX;
        this.startY = startY;
        this.endX = endX;
        this.endY = endY;
        line = new Line(startX, startY, endX, endY);
        line.setStroke(Color.BLACK);
        line.setStrokeWidth(4);

        double angle = Math.atan2(endY - startY, endX - startX);
        double arrowLength = 10;
        double arrowAngle = Math.toRadians(30);

        double x1 = endX - arrowLength * Math.cos(angle - arrowAngle);
        double y1 = endY - arrowLength * Math.sin(angle - arrowAngle);

        double x2 = endX - arrowLength * Math.cos(angle + arrowAngle);
        double y2 = endY - arrowLength * Math.sin(angle + arrowAngle);

        leftWing = new Line(endX, endY, x1, y1);
        rightWing = new Line(endX, endY, x2, y2);

        leftWing.setStroke(Color.BLACK);
        rightWing.setStroke(Color.BLACK);

        leftWing.setStrokeWidth(4);
        rightWing.setStrokeWidth(4);
        getChildren().addAll(line, leftWing, rightWing);
    }

    public void setAnotherColorLines(Color color){
        line.setStroke(color);
    }
}

```



```
        leftWing.setStroke(color);
        rightWing.setStroke(color);
    }

    public double getStartX(){
        return startX;
    }

    public double getStartY(){
        return startY;
    }

    public double getEndX(){
        return endX;
    }

    public double getEndY(){
        return endY;
    }
}
```