

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

ОТЧЕТ
по учебной практике
Тема: Визуализация алгоритмов на графах на языке Java

Студент гр. 7303	_____	Ковалёв К.А.
Студент гр. 7303	_____	Романенко М.В.
Студент гр. 7303	_____	Державин Д.П.
Руководитель	_____	Размочаева Н.В.

Санкт-Петербург
2019

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Ковалёв К.А. группы 7303

Студент Романенко М.В. группы 7303

Студент Державин Д.П. группы 7303

Тема практики: Визуализация алгоритмов на графах на языке Java

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на языке Java с графическим интерфейсом.

Алгоритм: Косарайю

Сроки прохождения практики: 01.07.2019 – 14.07.2019

Дата сдачи отчета: 10.07.2019

Дата защиты отчета: 10.07.2019

Студент	_____	Ковалёв К.А.
---------	-------	--------------

Студент	_____	Романенко М.В.
---------	-------	----------------

Студент	_____	Державин Д.П.
---------	-------	---------------

Руководитель	_____	Размочасева Н.В.
--------------	-------	------------------

АННОТАЦИЯ

В ходе выполнения задания учебной практики была реализована программа, осуществляющая поиск компонент сильной связности в ориентированном графе с помощью алгоритма Косарайю. Программа разработана в среде IntelliJ IDEA. Язык, используемый для написания программы – Java. В проекте используется Swing — библиотека для создания графического интерфейса для программ на языке Java [5]. Разработанная программа детально показывает этапы работы алгоритма.

SUMMARY

During the educational practice there has been created the program that searches strongly connected components using Kosaraju's algorithm. The program has been developed in IDE IntelliJ IDEA. The language of program is Java. There has been used Swing in a project. It is a library for writing GUI in Java. The developed program shows in detail the stages of the algorithm Kosaraju.

СОДЕРЖАНИЕ

	Введение	5
1.	Постановка задачи	6
1.1.	Теоретические сведения	6
1.2.	Реализуемый алгоритм	6
2.	Спецификация программы	8
2.1.	Требования к входным данным программы	8
2.2.	Описание интерфейса	8
3.	План разработки и распределение ролей в бригаде	11
3.1.	План разработки	11
3.2.	Распределение ролей в бригаде	11
4.	Особенности реализации	12
4.1	Архитектура программы	12
5.	Процесс тестирования	13
5.1	Тестирование графического интерфейса	13
5.2	Тестирование кода алгоритма	18
	Заключение	24
	Список используемых источников	25

ВВЕДЕНИЕ

Целью данной учебной практики является развитие следующих навыков:

- программирование на языке Java;
- работа над проектом в команде;
- использование системы контроля версий.

Таким образом, данный отчет посвящен выполнению учебного проекта, выданного в соответствии с указанными требованиями. Основой для изучения языка Java, стал курс на образовательной платформе Stepik[1], и руководство Гербера Шилдта[5].

Формулировка задания: требуется разработать программу на языке Java, визуализирующую алгоритм поиска компонент сильной связности в ориентированном графе.

1. ПОСТАНОВКА ЗАДАЧИ

1.2. Теоретические сведения

Дан ориентированный или неориентированный граф. Граф называется сильно связным, если любые две его вершины сильно связаны. Две вершины s и t любого графа сильно связны, если существует ориентированный путь из s в t и ориентированный путь из t в s . Компонентами сильной связности орграфа называются его максимальные по включению сильно связные подграфы.

Поиск в глубину (англ. depth-first search, DFS) – это рекурсивный алгоритм обхода вершин графа. Алгоритм поиска: перебираем все исходящие из рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось рёбер, которые ведут в нерассмотренную вершину. Если после завершения алгоритма не все вершины были рассмотрены, то необходимо запустить алгоритм от одной из нерассмотренных вершин.

1.2. Реализуемый алгоритм

Алгоритм Косарайю содержит три этапа: поиск в глубину с фиксированием времени выхода вершин, транспонирование рёбер графа и непосредственно сам поиск компонент связности [4].

- *Первый этап.* Поиск в глубину с фиксированием времени выхода вершин. Обозначить все вершины графа не посещёнными и завести два стека: для DFS и списка времени выхода вершин графа. Применить к графу поиск в глубину. Если из текущей вершины нельзя попасть в новую, положить её в список времени выхода вершин графа. Если стек для DFS пуст, проверить, существует ли не посещённая вершина. Существует - положить её в стек для

DFS и применить к графу действия первого этапа, иначе перейти к следующему этапу.

- *Второй этап.* Транспонирование рёбер графа. Необходимо сменить ориентацию каждого ребра графа.

- *Третий этап.* Нахождение компонент сильной связности. Завести стек для DFS. Пока список времени выхода вершин графа не пуст, снять вершину со списка времени выхода вершин графа и поиском в глубину искать очередную вершину и добавлять в компоненту сильной связности. Если стек для DFS пуст, снять со списка времени выхода вершин графа вершину и начать искать новую компоненту сильной связности.

2. СПЕЦИФИКАЦИЯ ПРОГРАММЫ

2.1. Требования к входным данным программы

Доступно 2 способа ввода:

- Вручную. Пользуясь кнопками панели инструментов и/или Меню → Действие можно построить граф вручную.
- Ввод из файла. Пункт меню Файл → Сохранить, либо по комбинации клавиш **ctrl+O**. Граф хранится в виде списка инцидентности. Каждый список содержит вершину, её координаты, если они даны, и связанные с этой вершиной вершины.

2.2. Описание интерфейса

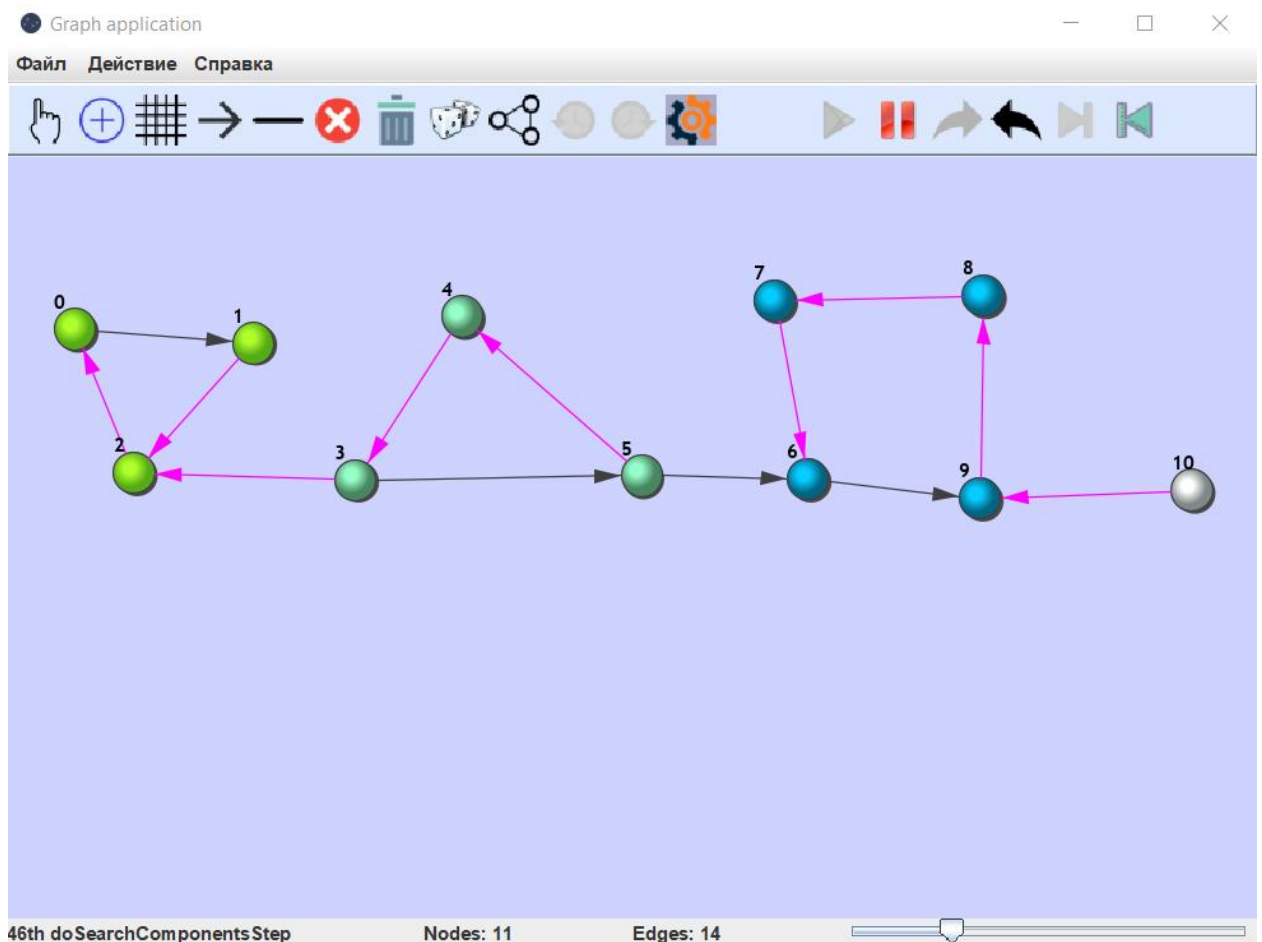




















Рисунок 1. Схема графического интерфейса пользователя

В нижней части окна находится статус бар с текстовым отображением выполняемого в данный момент действия, количеством вершин и ребер текущего графа и слайдер для изменения скорости работы алгоритма.

На панели инструментов сверху располагаются кнопки для создания и редактирования графа, а также кнопки для управления состоянием выполнения алгоритма.

1.  – перемещение вершин мышью.
2.  – добавить вершину;
3.  – соединить все вершины неориентированными ребрами;
4.  – добавить ориентированное ребро;
5.  – добавить неориентированное ребро;
6.  – удалить ребро/рёбра и вершины;
7.  – очистить графическую панель;
8.  – сгенерировать случайный граф;
9.  – создать граф в виде триангуляционной сети;
10.  – отменить последнее совершенное действие, также доступно по комбинации клавиш `ctrl+Z`;
11.  – вернуть последнее отмененное действие, также доступно по комбинации клавиш `ctrl+Y`;
12.  – начать выполнение выбранного алгоритма с выбора вершины;
13.  – запустить выполнение алгоритма;

14.  – приостановить выполнение алгоритма;
15.  – шаг вперед для выполняемого алгоритма;
16.  – шаг назад для выполняемого алгоритма;
17.  – переместиться к концу выполнения алгоритма;
18.  – переместиться к началу выполнения алгоритма;

Также для пользователя реализованы следующие возможности взаимодействия с программой:

- Перемещение сцены мышью с зажатым колёсиком;
- Изменение масштаба сцены по кручению колёсика мыши;
- Регулирование скорости работы алгоритма;
- Сохранение графа в файл. Пункт меню Файл → Сохранить, либо по комбинации клавиш ctrl+S;

3. РАЗРАБОТКИ И РЕСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

3.1. План разработки

Таблица 1 – план разработки

№	Задача	Планируемый срок сдачи	Срок сдачи
1	Добавить спецификацию	02.07.2019	02.07.2019
2	Представить прототип	02.07.2019	02.07.2019
3	Представить 1 версию	08.07.2019	05.07.2019
4	Представить 2 версию	10.07.2019	08.07.2019
5	Представить финальную версию	12.07.2019	10.07.2019

3.2. Распределение ролей в бригаде

Державин Д.П. Реализация пользовательского интерфейса и алгоритма Косарайю на языке Java, тестирование программы.

Ковалёв К.А. Разработка и реализация архитектуры хранения данных, визуализация алгоритма.

Романенко М.В. Разработка и реализация ввода графа из файла и его сохранение в файл, дизайн пользовательского интерфейса.

4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

4.1. Архитектура программы

Как говорил мой дед, «Разделяй и властвуй». Именно поэтому программа спроектирована таким образом, чтобы классы обработки программы не имели доступа визуальной составляющей (Но не везде). Классы программы связывались таким образом, чтобы каждый из них занимался исключительно своей задачей и передавал информацию о своем состоянии другим. Данная архитектура позволила свободно добавлять в код новые классы для визуализации алгоритмов.

Для отображения графа на графическую панель был создан класс DrawGraph, который получает информацию о вершинах и ребрах графа, берет из этих данных классы «обертки», и передает им управление на отрисовку всех составляющих графа.

Класс Graph хранит в себе список вершин и список ребер. Данные списки необходимы для предоставления всех компонент визуализатору.

Для быстрого доступа к ребрам, которые связаны с конкретной вершиной, класс Node содержит в себе список экземпляров класса Edge, который в свою очередь хранит в себе пару экземпляров Node. Благодаря такой архитектуре хранения данных графа можно быстро итерировать по любым вершинам и ребрам графа.

Для того, чтобы граф не занимался обработкой команд, поступающих от GUI программы, был разработан класс GraphEventManager. Данный класс принимает информацию от кнопок графической панели, а также события нажатия клавиши и обрабатывает их в зависимости от текущего состояния графа(режим добавления вершины, удаления ребра и т.д.). Класс GraphEventManager, после обработки полученных данных, передает графу команды на добавление, удаление вершин и\или ребер.

Гвоздем данной программы являются классы алгоритмов. Созданный абстрактный класс Algorithm содержит основные методы для работы любого алгоритма. Для добавления нового (в последний раз пишу это слово)

алгоритма создается наследник класса `Algorithm`, где переопределяются методы базового класса.

Для хранения экземпляров алгоритмов был добавлен отдельный класс со статическим списком доступных для визуализации алгоритмов.

Менеджеров много не бывает, поэтому в проект был добавлен класс `AlgorithmEventManager`, который, как и его предшественник, получает данные от GUI, обрабатывает их и передает результат текущему алгоритму, которые тот исполняет и возвращает свой статус как ответ. По результатам работы алгоритма обновляются элементы GUI и перерисовывается граф.

5. ПРОЦЕСС ТЕСТИРОВАНИЯ

5.1. Тестирование графического интерфейса

Запускаем программный модуль.

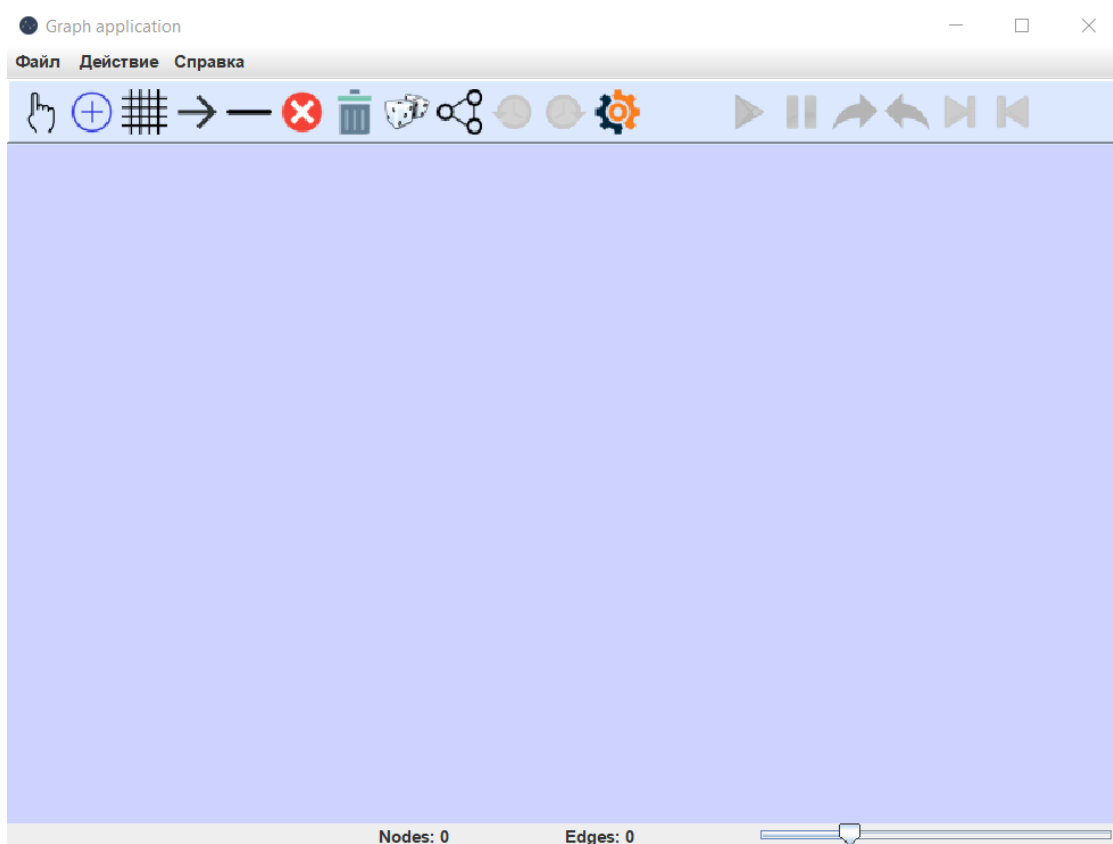


Рисунок 2 – Внешний вид интерфейса программы после запуска

С помощью панели инструментов добавим немного вершин, а после соединим их всех друг с другом при помощи соответствующей кнопки.

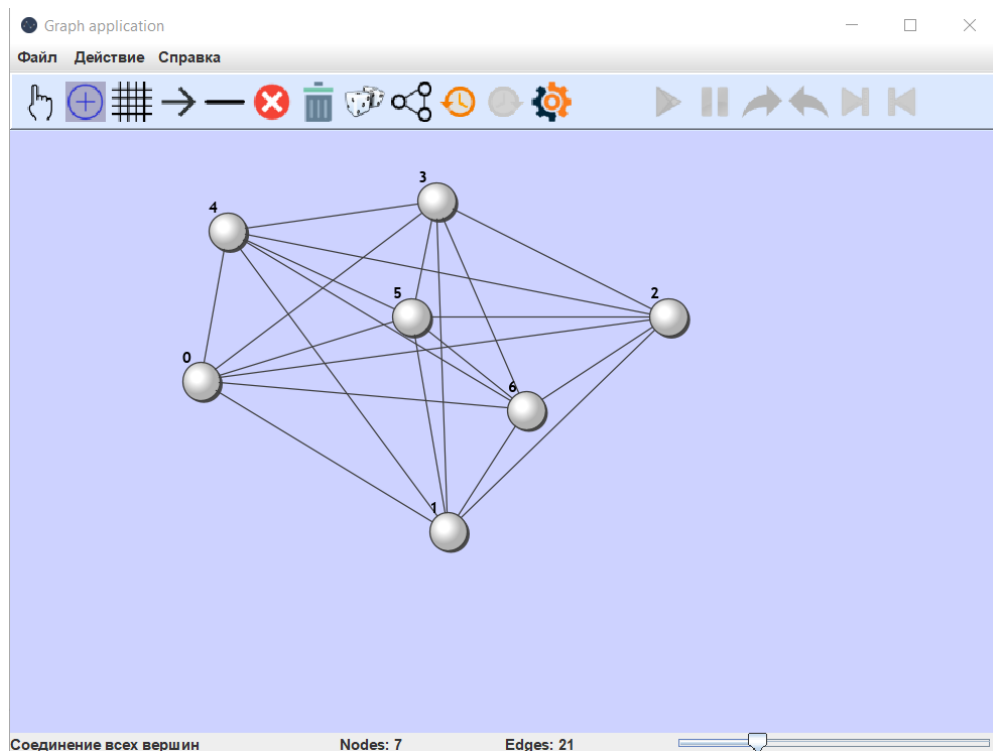


Рисунок 3 – Внешний вид созданного графа

После удалим некоторые ребра.

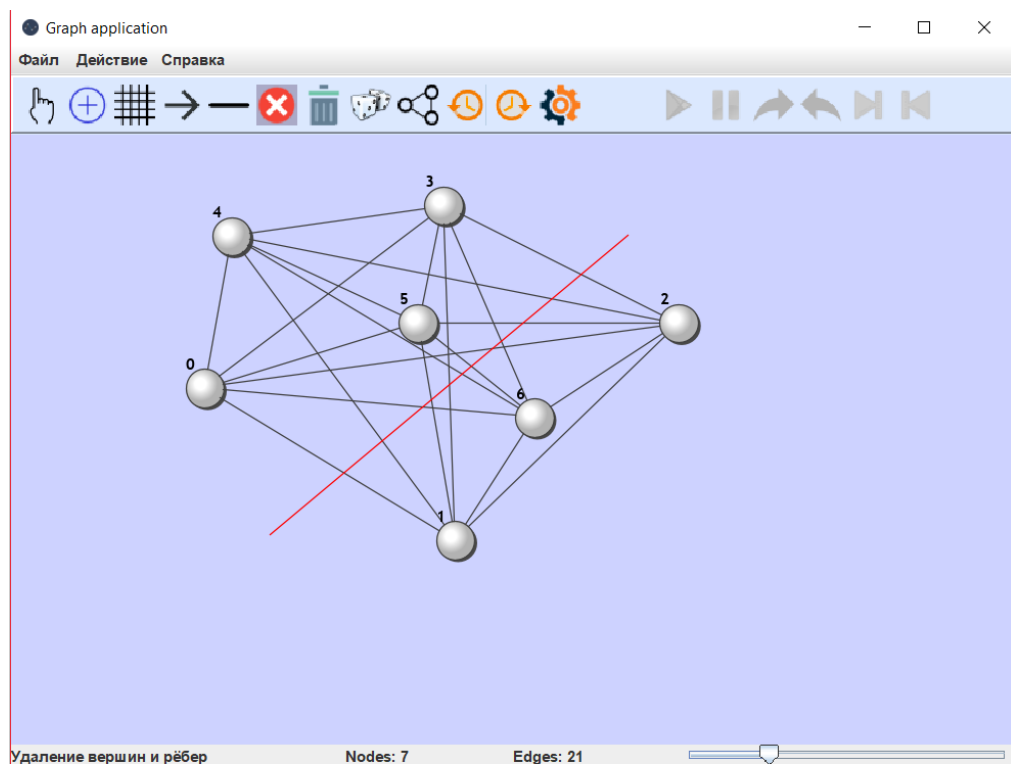


Рисунок 4 – Процесс удаления ребер

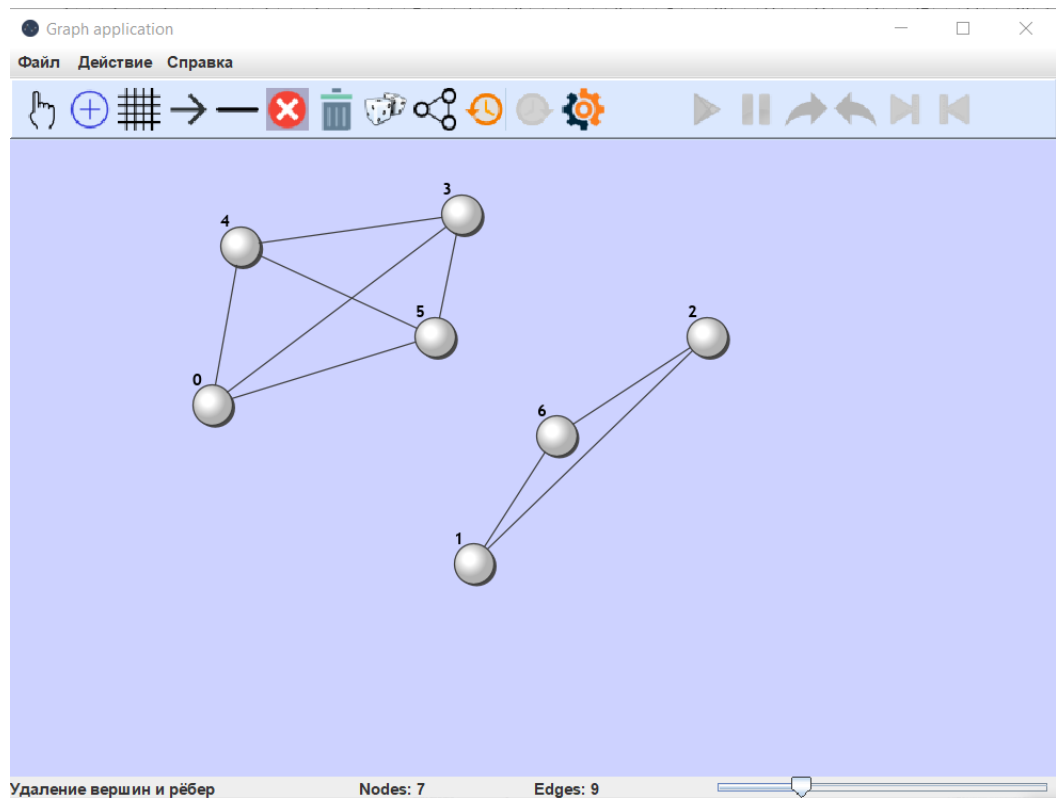


Рисунок 5 – После удаления ребер

Очистим сцену, добавим новые вершины и создадим триангуляционную сеть. После удалим некоторые ребра, что бы получилось 4 отдельных графа.

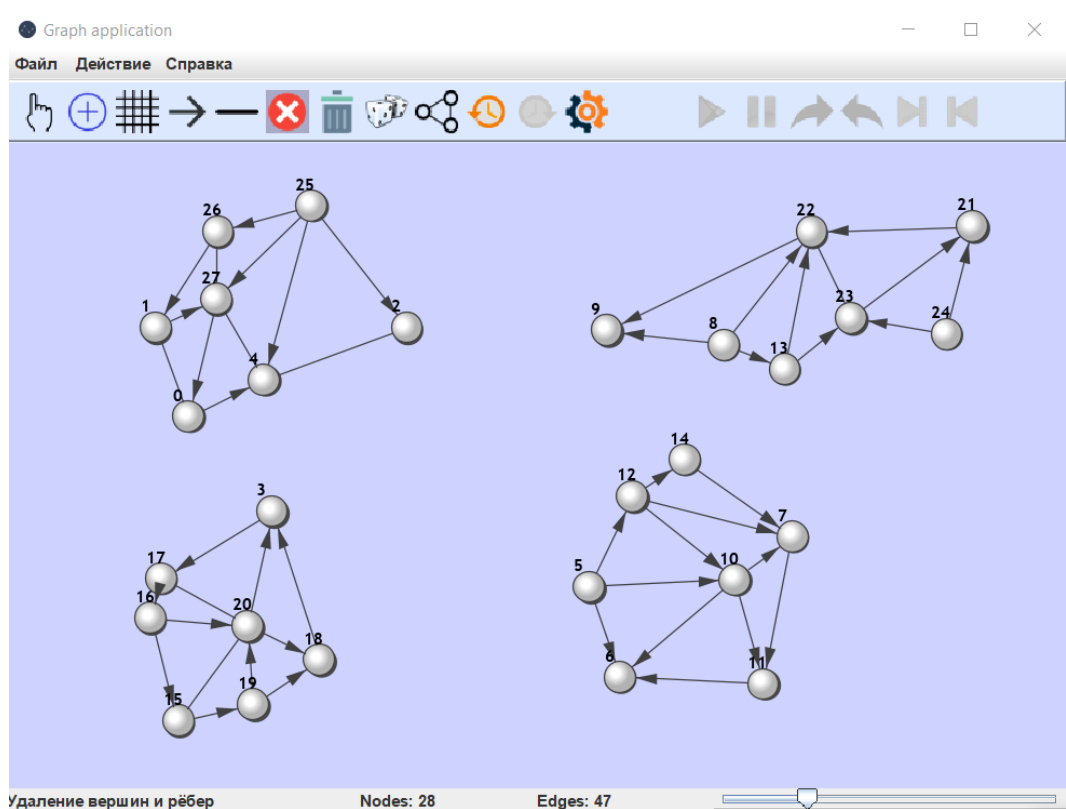


Рисунок 6 – Новый граф

Запустим для него алгоритм Косарайю. Выберем вершину и нажмем пуск. В процессе выполнения можно изменять скорость, приостанавливать и выполнять пошагово.

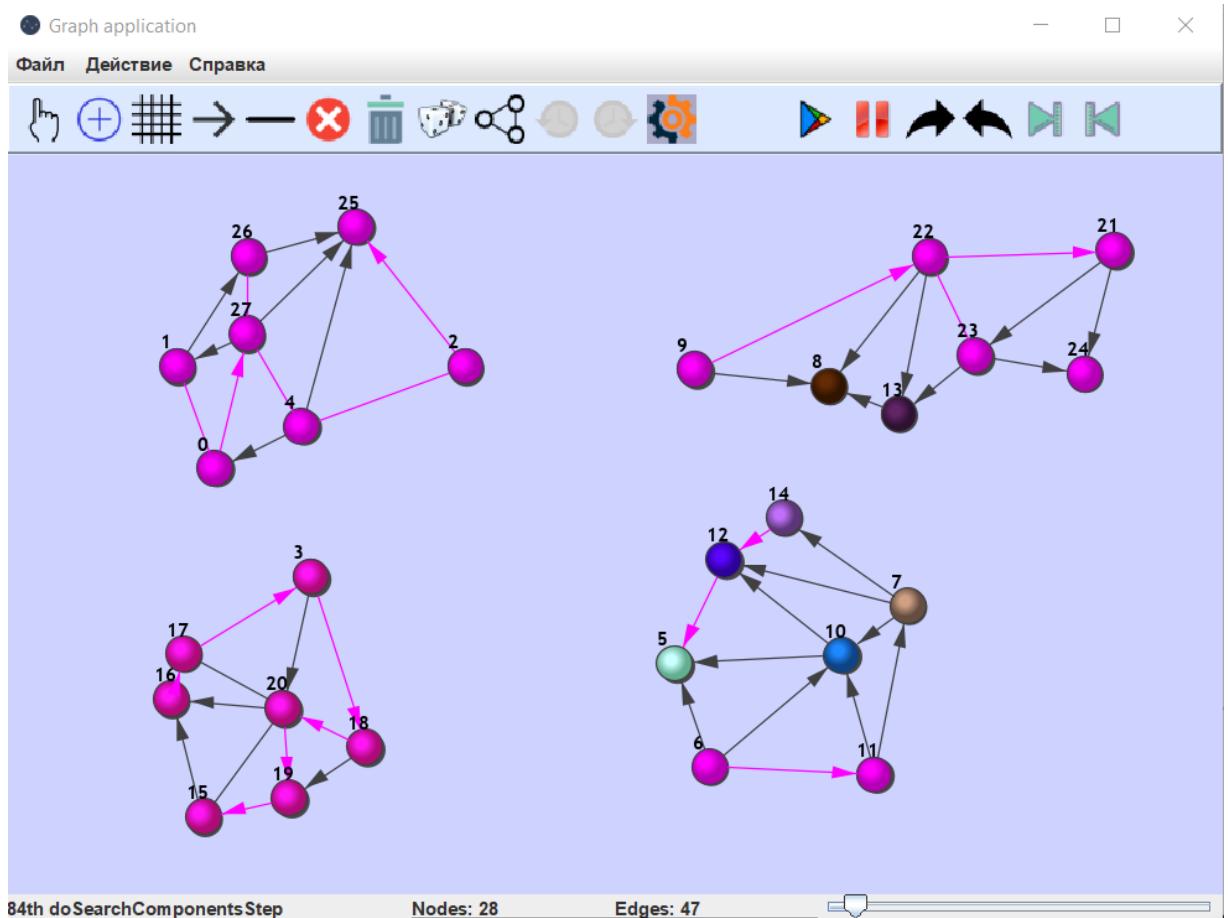


Рисунок 7 – Пауза при выполнении алгоритма

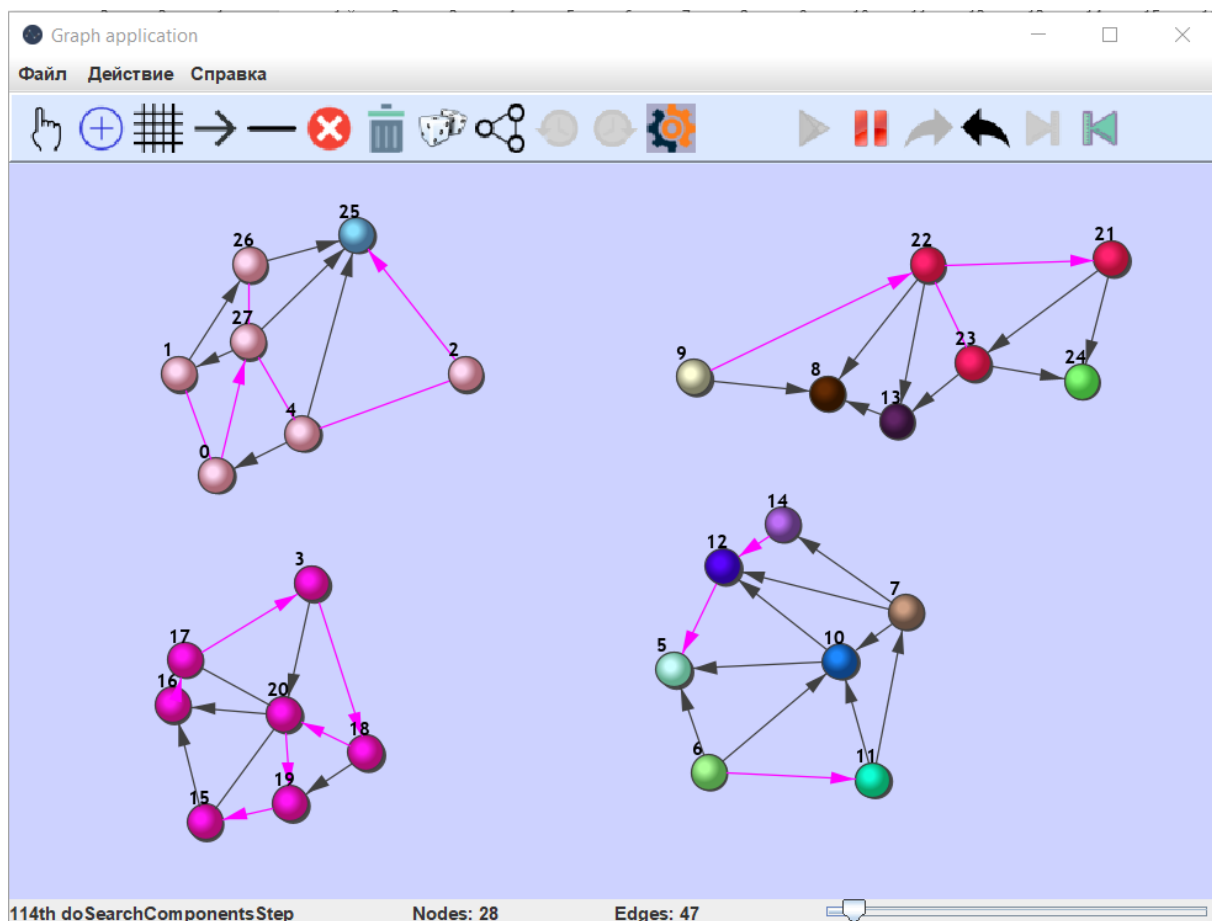


Рисунок 8 – Результат работы алгоритма

Тестирование графического интерфейса пройдено успешно.

5.2. Тестирование алгоритма

Для тестирования алгоритма был написан класс `AlgorithmKosarajuTest`, который симулировал графическую панель и кнопки для работы с алгоритмом. В каждом тесте программа считывала граф из файла и запускала алгоритм Косарайю. Для проверки результата работы алгоритма был написан метод, который проверял, раскрашены ли вершины, принадлежащие одной компоненте сильной связности, в один цвет и отличаются ли цвета каждой компоненты.

Первый тест – тестирование графа, в котором на этапе DFS стек не посещённых вершин несколько раз становится пустым. Программа должна

находить четыре компоненты связности. На рис. 9 приведён соответствующий граф и результат работы алгоритма.

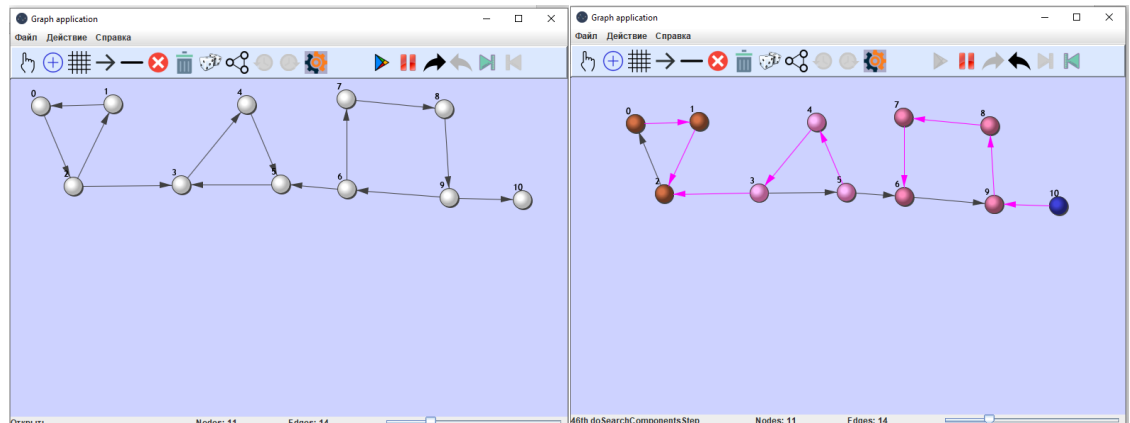


Рисунок 9 – Тестирование графа, стек которого несколько раз становится пустым на этапе DFS

Второй тест – тестирование графа с одной вершиной. Программа должна находить одну компоненту связности. На рис. 10 приведён соответствующий граф и результат работы алгоритма.

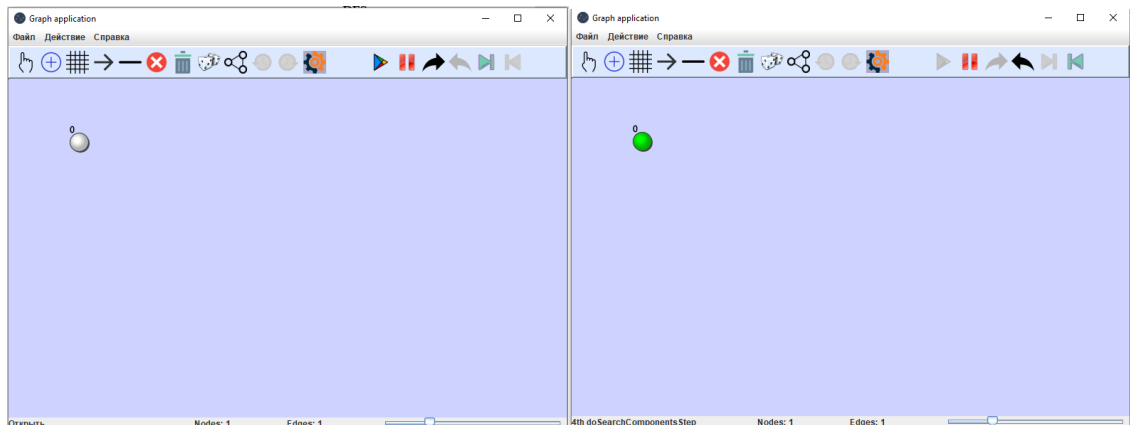


Рисунок 10 – Тестирование графа, с одной вершиной

Третий тест – тестирование графа, состоящего из нескольких одиночных вершин. Программа должна находить пять компонент связности. На рис. 11 приведён соответствующий граф и результат работы алгоритма.

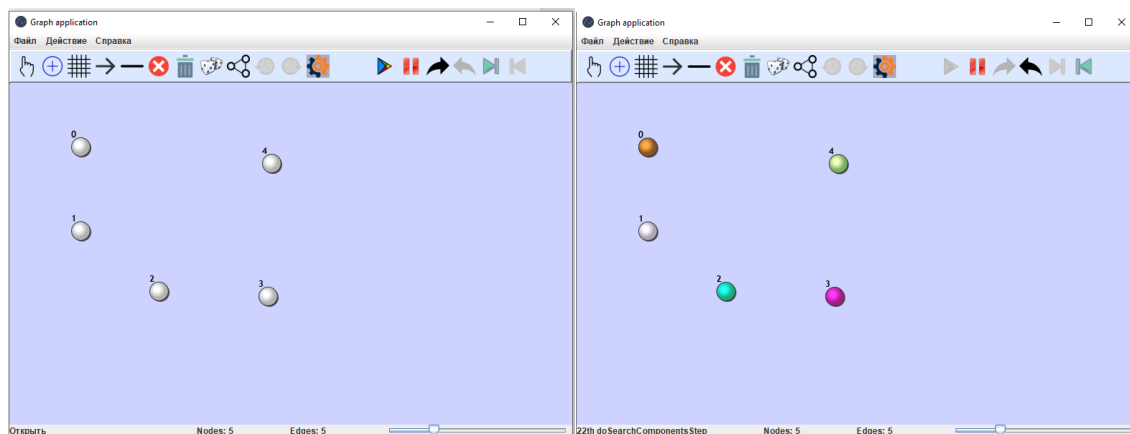


Рисунок 11 – Тестирование графа, состоящего из нескольких
одиночных вершин

Четвёртый тест – тестирование графа, состоящего из двух не связанных компонент связности. Программа должна находить две компоненты связности. На рис. 12 приведён соответствующий граф и результат работы алгоритма.

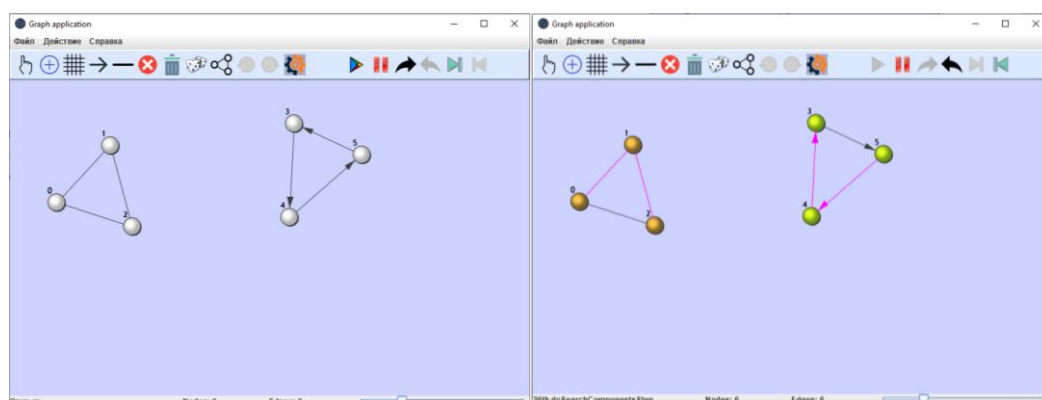


Рисунок 12 – Тестирование графа, состоящего из двух не связанных
компонент связности

Пятый тест – тестирование графа, компоненты связности которого соединены ориентированными рёбрами. Компоненты и соединяющие их рёбра не должны образовывать цикл. Программа должна находить три компоненты связности. На рис. 13 приведён соответствующий граф и результат работы алгоритма.

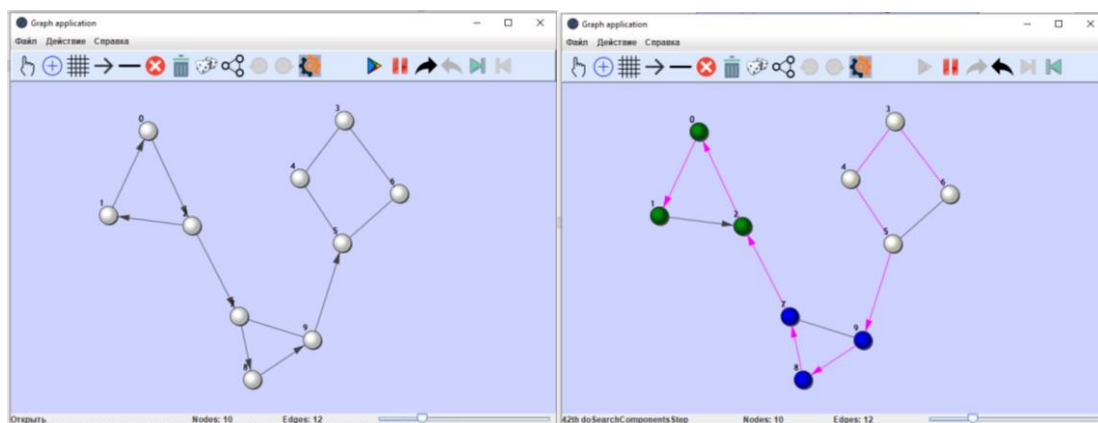


Рисунок 13 – Тестирование графа, компоненты связности которого соединены ориентированными рёбрами

Шестой тест – тестирование графа, компоненты связности которого соединены двумя ориентированными рёбрами. Компоненты и соединяющие их рёбра не должны образовывать цикл. Программа должна находить две компоненты связности. На рис. 14 приведён соответствующий граф и результат работы алгоритма.

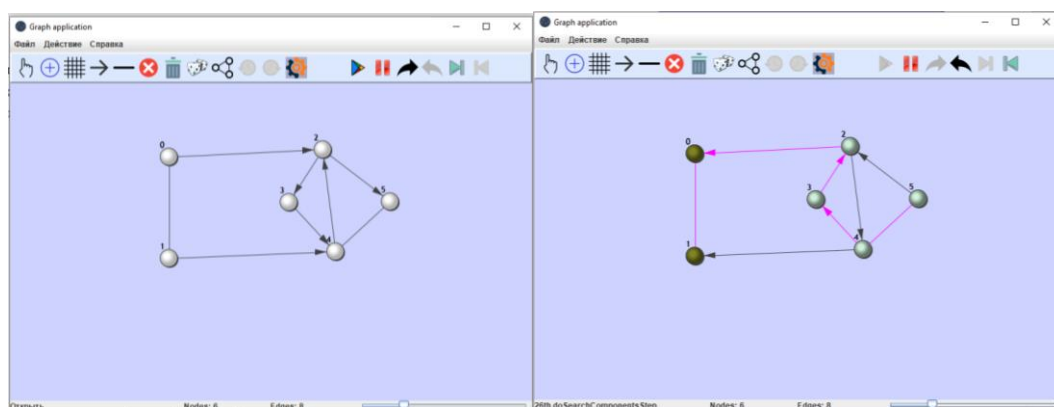


Рисунок 14 – тестирование графа, компоненты связности которого соединены двумя ориентированными рёбрами

Седьмой тест – тестирование графа, все вершины которого связаны между собой. Программа должна находить одну компоненты связности. На рис. 15 приведён соответствующий граф и результат работы алгоритма.

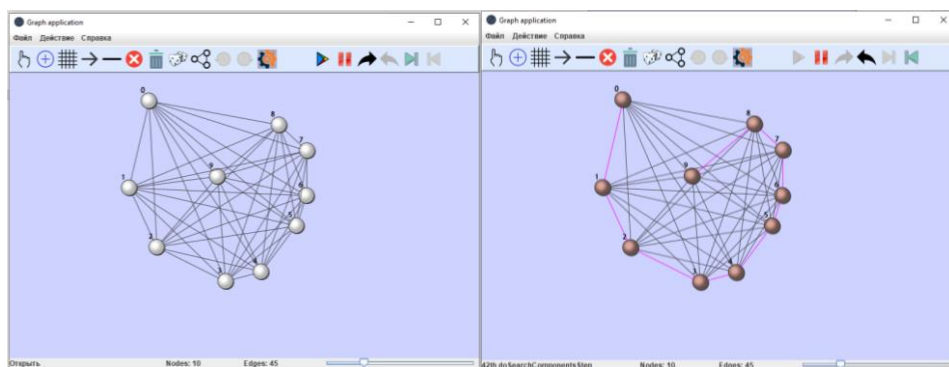


Рисунок 15 – Тестирование графа, все вершины которого связаны между собой

Восьмой тест – тестирование графа, который на первый взгляд может показаться состоящим из трёх компонент сильной связности. Компоненты сильной связности и рёбра, соединяющие их, не должны образовывать цикл. Программа должна находить одну компоненту связности. На рис. 16 приведён соответствующий граф и результат работы алгоритма.

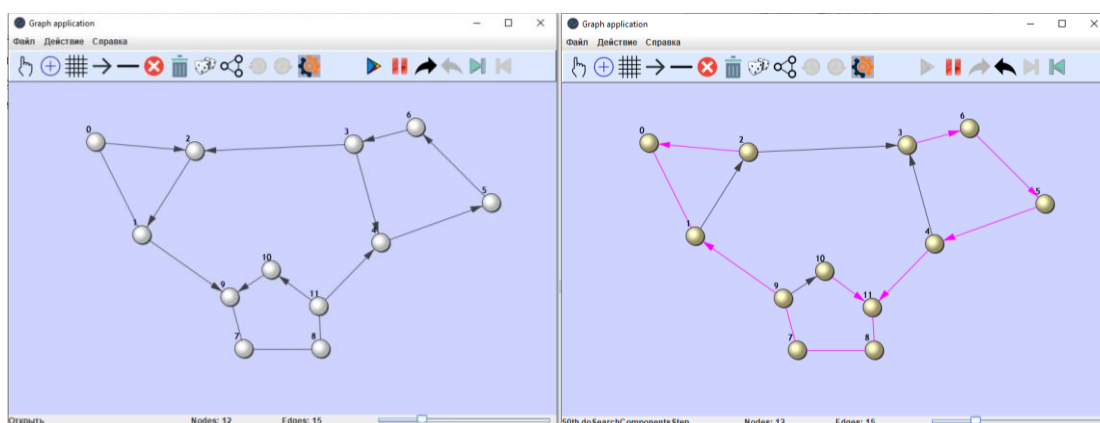


Рисунок 16 – Тестирование графа, который на первый взгляд может показаться состоящим из трёх компонент сильной связности

Девятый тест – тестирование графа, вершины и рёбра которого образуют незамкнутую цепь. Программа должна находить пять компонент связности. На рис. 17 приведён соответствующий граф и результат работы алгоритма.

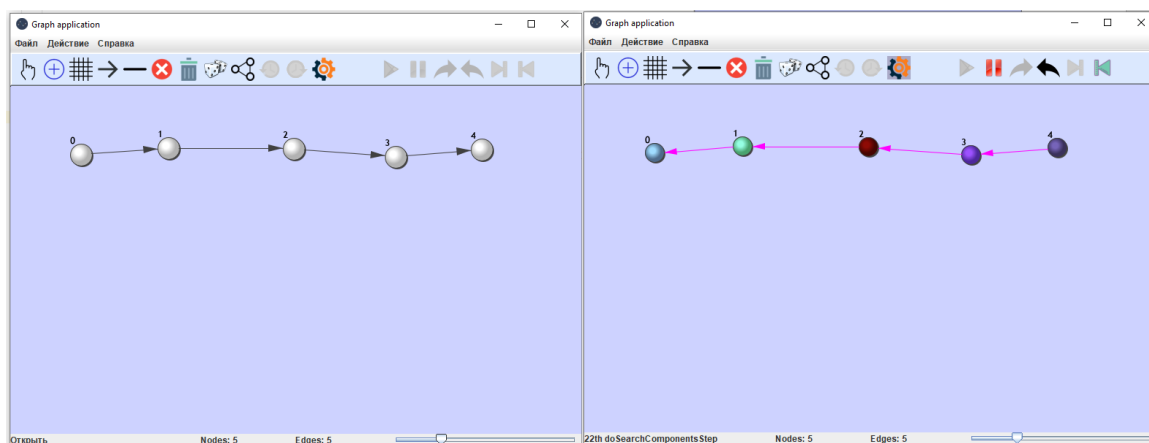


Рисунок 17 – Тестирование графа, вершины и рёбра которого образуют незамкнутую цепь

Девятый тест – тестирование графа, вершины и рёбра которого образуют замкнутую цепь. Программа должна находить одну компоненту связности. На рис. 18 приведён соответствующий граф и результат работы алгоритма.

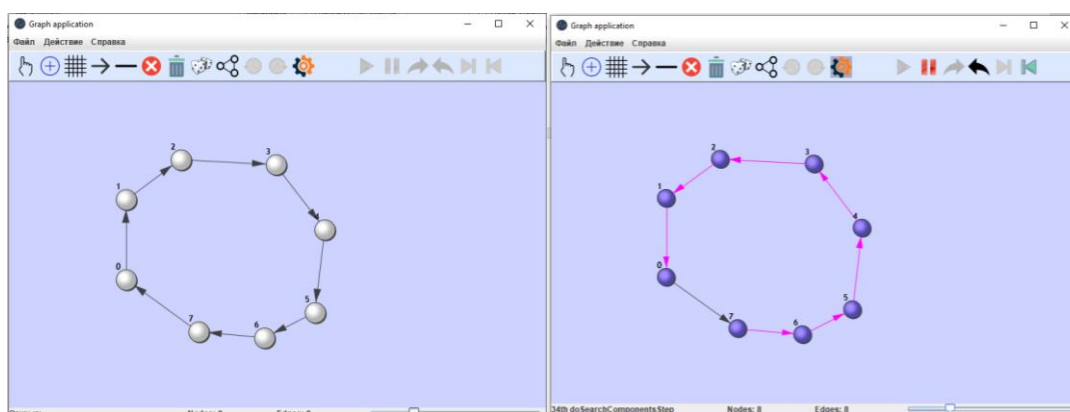


Рисунок 18 – Тестирование графа, вершины и рёбра которого образуют замкнутую цепь

ЗАКЛЮЧЕНИЕ

В результате выполнения учебной практики была разработана и протестирована программа, реализующая алгоритм поиска компонент сильной связанности в графе и наглядно демонстрирующая принцип его работы.

В ходе работы было проведено тестирование с целью выявления возможных ошибок. По результатам было выяснено, что алгоритм работает корректно.

Выполнение данного проекта позволило приобрести навыки, необходимые для будущей профессии, тесно связанной с ИТ. Это навыки:

- разработки в среде программирования Java;
- работы в команде;
- использования известной системы контроля версий GitHub;
- использования библиотеки Swing для реализации графического интерфейса.

Таким образом, цели практики успешно достигнуты, и по окончании разработки получен корректно работающий визуализатор алгоритма Косарайю на языке программирования Java.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java. Базовый курс. // Stepik. URL: <https://stepik.org/course/Java-Базовый-курс-187/syllabus> .
2. Брюс Эккель. Философия Java. СПб.: Питер, 2009.
3. Дейтел Х.М., Дейтел П.Дж. Как программировать на Java. Кн. 1: Основы программирования. / пер. с англ. А.В. Козлова. 4-е изд. М.: Бином, 2003;
4. Седжвик Р., Уэйн К. Алгоритмы на Java, 4-е изд.: Пер. с англ. – М.: ООО “И.Д.Вильямс”, 2013. – 848 с.
5. Шилдт Г. - Java 8. Полное руководство. 9-е изд.: Пер. с англ. – М.: ООО “И.Д.Вильямс”, 2015. – 1376 с.