**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по учебной практике**

**Тема: Визуализация алгоритмов на языке Java**

|  |  |  |
| --- | --- | --- |
| Студент  гр. 5304 |  | Якушкин А.Э. |
| Студентка гр. 5304 |  | Гайдук М.А. |
| Студентка гр. 5304 |  | Федорова М.Д. |
| Руководитель |  | Томша А.Э. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент группы 5304 Якушкин А.Э. | | |
| Студентка группы 5304 Гайдук М.А. | | |
| Студентка группы 5304 Федорова М.Д. | | |
| Тема практики: визуализация алгоритмов на языке Java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: поиск компонент сильной связности в ориентированном графе. | | |
| Сроки прохождения практики: 21.06.2017 – 04.07.2017 | | |
| Дата сдачи отчета: | | |
| Дата защиты отчета: | | |
|  | | |
| Студент гр.5304 |  | Якушкин А.Э. |
| Студентка гр.5304 |  | Гайдук М.А. |
| Студентка гр.5304 |  | Федорова М.Д. |
| Руководитель |  | Томша А.Э. |

**АННОТАЦИЯ**

В данной работе изучается и разрабатывается визуализатор алгоритма поиска сильно связных компонент в ориентированном графе. Программа разработана в среде IntelliJ IDEA. Язык, используемый для написания программы – Java. В проекте используется Qt Jambi — библиотека Java, представляющая собой обёртку Qt, каркаса графических приложений, которая позволяет Java разработчикам использовать Qt в своих проектах.

Программа подробно показывает процесс нахождения сильно связных компонент ориентированного графа. Для визуализации в программе предусмотрен графический интерфейс, описание которого приведено в данном отчёте. Работоспособность программы проверена рядом примеров.

**SUMMARY**

In this work, an algorithm for searching for strongly connected components in an oriented graph is studied and developed. The program was created in IntelliJ IDEA. The language, which used for writing the program, is Java. In this project it is used Qt Jambi  - a Java binding of the cross-platform application framework [Qt](https://en.wikipedia.org/wiki/Qt_(software)). It enables Java developers to use Qt within Java programming language.

The program shows process of searching for strongly connected components in an oriented graph in details. For visualization, the program provides a graphical interface, the description of which is given in this report. The working capacity of the program is checked by a number of tests.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc486340018)

[1. ОПИСАНИЕ АЛГОРИТМА 6](#_Toc486340019)

[2. СПЕЦИФИКАЦИЯ ПРОГРАММЫ 7](#_Toc486340020)

[2.1. Структура классов 7](#_Toc486340021)

[2.2. Описание интерфейса 8](#_Toc486340022)

[3. ТРЕБОВАНИЯ К ПРОГРАММЕ 9](#_Toc486340023)

[4. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ 10](#_Toc486340024)

[5. оСОБЕННОСТИ РЕАЛИЗАЦИИ 11](#_Toc486340025)

[6. ТЕСТИРОВАНИЕ 11](#_Toc486340026)

[6.1 Тестирование графического интерфейса 11](#_Toc486340027)

[6.2 Тестирование кода алгоритма 11](#_Toc486340028)

[ЗАКЛЮЧЕНИЕ 11](#_Toc486340029)

[СПИСОК ИСПОЛЬЗОВАННых ИСТОЧНИКОВ 11](#_Toc486340030)

[пРИЛОЖЕНИЕ а 11](#_Toc486340031)

# ВВЕДЕНИЕ

**Формулировка задания:** требуется разработать программу, визуализирующую алгоритм поиска компонент сильной связности (алгоритм Косорайю) в ориентированном графе. При этом должен присутствовать графический интерфейс.

Дан ориентированный граф *G*, множество вершин которого *V* и множество рёбер — *E*. Петли и кратные рёбра допускаются. Обозначим через *n* количество вершин графа, через *m* — количество рёбер.

Компонентой сильной связности (strongly connected component) называется такое (максимальное по включению) подмножество вершин C, что любые две вершины этого подмножества достижимы друг из друга, т.е. для любых *u,v* ϵ *C*:

*u → v, v → u*

где символом → здесь и далее мы будем обозначать достижимость, т.е. существование пути из первой вершины во вторую.

1. **ОПИСАНИЕ АЛГОРИТМА**

Для поиска компонент сильной связности используется алгоритм Косорайю:

1 шаг. Запустить серию обходов в глубину графа G, которая возвращает вершины в порядке увеличения времени выхода {\rm tout}, т.е. некоторый список \rm order.

2 шаг. Построить транспонированный граф G^T. Запустить серию обходов в глубину/ширину этого графа в порядке, определяемом списком \rm order (а именно, в обратном порядке, т.е. в порядке уменьшения времени выхода). Каждое множество вершин, достигнутое в результате очередного запуска обхода, и будет очередной компонентой сильной связности.

1. **СПЕЦИФИКАЦИЯ ПРОГРАММЫ**

2.1. Структура классов

В рамках данной работы будет использоваться структура классов, схожая с MVC-парадигмой. Моделью в данной работе выступает граф, реализованный с помощью отдельного класса-контейнера. Этот класс содержит в себе информацию о вершинах в виде списка инцидентности и должен поддерживать операцию транспонирования. Контроллером в данной схеме является класс, реализующий исследуемый алгоритм. Этот класс должен реагировать на события графического интерфейса пользователя и реализовывать алгоритм Косорайю как в пошаговом, так и в автоматическом режиме. Видом в данной структуре будет являться класс окна, содержащий в себе все элементы графического интерфейса пользователя, включая поле графического вывода графа. На рисунке 1 представлена UML-диаграмма проекта.

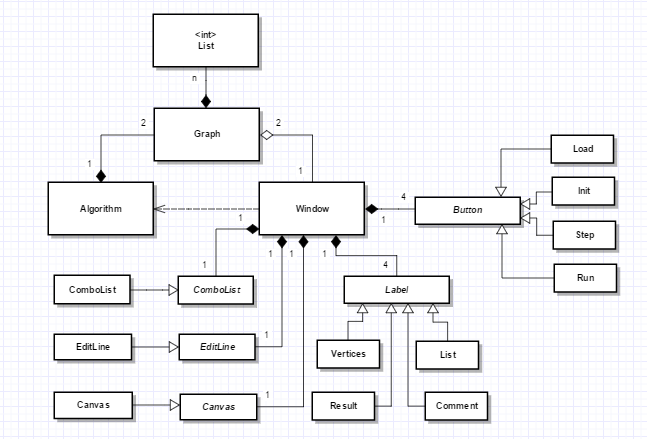


Рисунок 1. UML-диаграмма проекта

2.2. Описание интерфейса

Графический интерфейс пользователя состоит из 8 основных компонентов (см. рисунок 2).

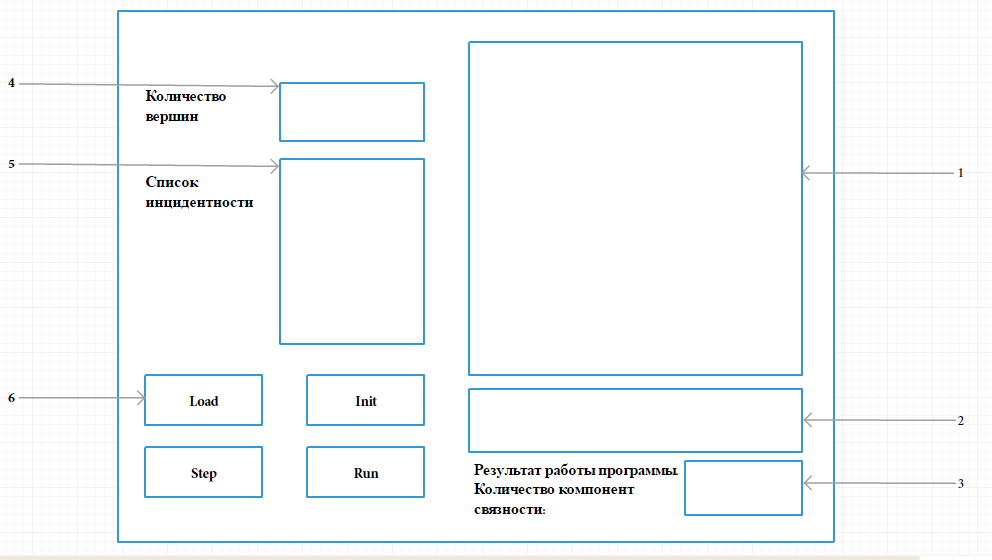


Рисунок 2. Схема графического интерфейса пользователя

Программа может поддерживать как ручной ввод пользователем, так и автоматический ввод из файла. Для того, чтобы ввести граф из файла, необходимо нажать кнопку Load в блоке 6 на рисунке 2. Если пользователь хочет ввести граф самостоятельно, то он должен ввести количество вершин графа в поле 4, после чего ввести список инцидентности в список 5. Для того, чтобы начать выполнение программы, необходимо нажать на кнопку Init в блоке 6, загрузив введенный граф в память. После этого кнопки Step и Run позволяют выполнить один шаг алгоритма и весь алгоритм полностью соответственно. Во время выполнения программы в окне 1 будет выводиться графическая интерпретация работы алгоритма с кратким пояснением в надписи 2; также в надписи 3 выводится итог работы алгоритма в виде количества найденных компонент сильной связности.

1. **ТРЕБОВАНИЯ К ПРОГРАММЕ**

Входные данные:

Граф, записанный в следующем формате: в первой строке количество вершин (целое число), каждая последующая *i-*я строка содержит последовательность вершин, инцидентных к (*i-2*)-й вершине. Отсчет вершин начинается с нуля.

Выходные данные:

Визуализация работы алгоритма Косорайю с краткими пояснениями.

1. **ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ**

Роли в бригаде были распределены следующим образом:

* архитектор-аналитик – Якушкин А.Э.;
* разработчик – Гайдук М.А.;
* тестировщик – Федорова М.Д.

Календарный план разработки приведен в таблице 1.

Таблица 1. Календарный план разработки

|  |  |  |
| --- | --- | --- |
| Дата | Этап работы | Ответственное лицо |
| 21 июня | Согласование темы проекта;  подготовка к работе (создание репозитория, настройка окружения и т.д.);  изучение теоретического материала. | Совместно |
| 22 июня | Составление спецификации проекта и планирование цикла разработки | Якушкин А.Э. |
| 23 июня | Согласование спецификации с руководителем практики и корректировка плана разработки | Якушкин А.Э. |
| 24-25 июня | Разработка графического интерфейса и основных классов | Гайдук М.А. |
| 25 июня | Тестирование и сборка прототипа | Федорова М.Д. |
| 26 июня | Согласование прототипа с руководителем практики | Совместно |
| 27 июня | Завершение реализации класса алгоритма и графа | Гайдук М.А. |
| 27 июня | Тестирование вывода графа на экран | Федорова М.Д. |
| 28 июня | Согласование рабочей версии проекта с руководителем практики | Совместно |
| 29 июня | Отладка проекта | Федорова М.Д. |
| 30 июня | Демонстрация и защита финальной версии | Совместно |

1. **теоретические сведения о графах**

*Ориентированный граф* – пара множеств *V* и *E*, где *V* – множество вершин, а *E* – множество *упорядоченных* пар вершин (*u*, *v*), называемых ребрами. Для краткости ориентированный граф также называется *орграфом*.

*Направленным* называется граф, в котором любые две вершины соединены не более, чем одним ребром (то есть направленный граф получен из неориентированного простой ориентацией ребер).

Неориентированный граф также можно считать орграфом, в котором каждому ребру (*u*, *v*) также соответствует ребро (*v*, *u*).

*Исток* графа – такая вершина, *в* которую не входит ни одно ребро. *Сток* – вершина, *из* которой не выходит ни одно ребро.

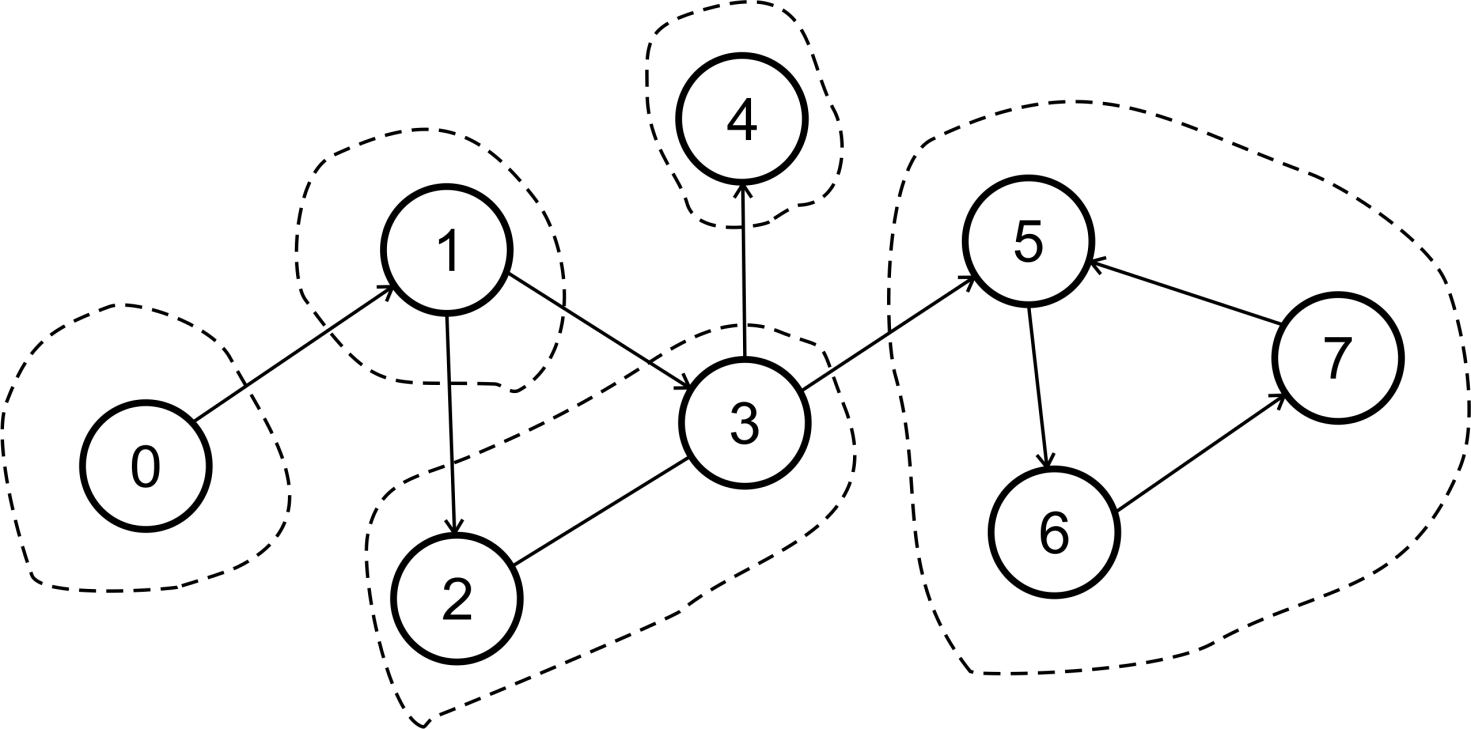


Рисунок 3. Пример ориентированного графа с выделенными компонентами сильной связности.

*Компонентой сильной связности* называется такое множество вершин *U*, в котором выполняется следующее условие: из любой вершины *u∈U* можно добраться по ребрам графа в любую другую вершину *v∈U*. Изолированная вершина также является компонентой сильной связности.

В неориентированном графе задача поиска компонент сильной связности (понятие слабой связности для неориентированного графа не имеет смысла) решается простым обходом в глубину с пометками вершин. Однако, в орграфе выбор начальной вершины сильно влияет на решение задачи: например, в графе на рис. 3, обход с нулевой вершины скажет, что весь граф является компонентой сильной связности, с первой – то, что все, кроме нулевой вершины, сильно связно.

Таким образом, использовать простой обход в глубину нельзя. Однако, можно заметить, что если выделить сток графа (или множество вершин, при объединении вырождающиеся в сток; здесь и далее под этими терминами будет пониматься именно множество вершин), то обход в глубину гарантированно даст одну компоненту сильной связности. Поэтому нужно выделить порядок для обходов в глубину, в котором начальные вершины расположены от стока к истоку.

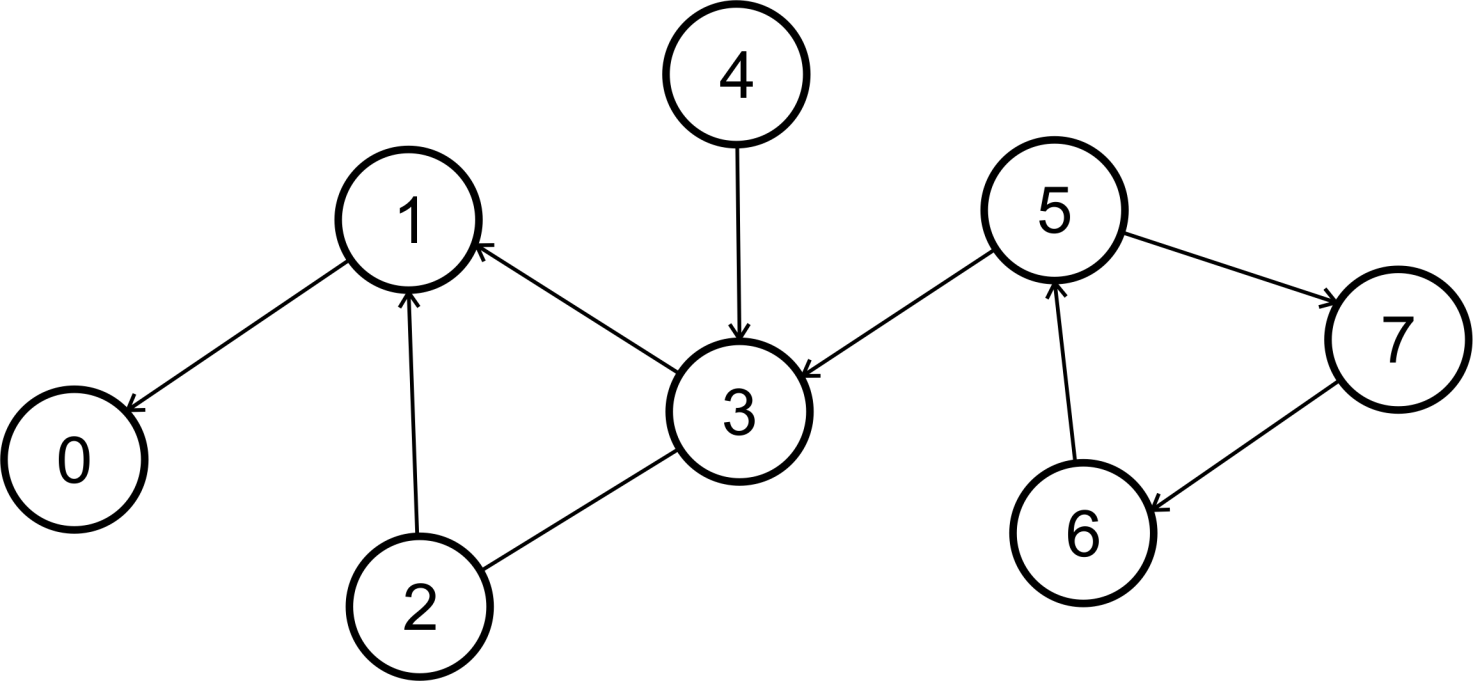


Рисунок 4. Транспонированный граф из рисунка 1.

Исток графа имеет интересное свойство, которое можно использовать: время его обработки при обходе в глубину всегда максимально в данной компоненте связности. Кроме того, исток графа является стоком в транспонированном графе (см. рис. 4), и наоборот. Отсюда можно вывести следующий алгоритм поиска компонент связности:

1. Найти последовательность истоков транспонированного графа путем обхода в глубину.
2. Провести обход в глубину по порядку вершин, найденных на шаге 1. Все вершины, найденные в течение одного обхода, будут являться компонентами сильной связности.
3. **оСОБЕННОСТИ РЕАЛИЗАЦИИ**
   1. **Описание классов**

* *class MyGraph*

Поля:

public ArrayList<Integer> IncidList[] – список инцидентности графа  
public int numV – количество вершин  
public int numE – количество ребер  
public int[] component – массив компонент сильной связности графа  
int componentNum – количество компонент связности в орграфе

Методы:

public MyGraph Transpose(MyGraph Graph) – транспонирование графа

* *class Algorithm*

Поля:

private MyGraph GraphT – список инцидентности транспонированного орграфа (массив списков)  
private boolean usedV[] – массив для хранения информации о пройденных и не пройденных вершинах  
private ArrayList<Integer> topSort – топологически упорядоченная перестановка номеров вершин графа

Методы:

private void dfs(int v, ArrayList<Integer>[] Graph) –обход графа в глубину

private void ccs(int v, int componentID, int[] component) – поиск компонент сильной связности орграфа

public void runInit (MyGraph Graph, boolean onestep) – инициализация компонентов для работы алгоритма поиска (создание вспомогательных массивов и транспонирование графа)

public int runCycle (MyGraph Graph, boolean onestep) – реализует циклическую часть алгоритма поиска компонент сильной связности, а именно: обход в глубину, просмотр вспомогательных массивов, при этом реализована возможность выполнения только одного шага алгоритма.

public int run(MyGraph Graph, boolean onestep) – объединяет runInit и runCycle в 1 метод

* *class Canvas*

Методы:

public void init – инициализация графа

private void adjustDisplaySettings( JGraph jg ) – настройки свойств экрана

public void paintComponent(Graphics g) – вывод компонент графа

* *class InputOutput*

Методы:

public MyGraph getData(MyGraph Graph, BufferedReader cin) –

ввод данных

public void printData(MyGraph Graph) –вывод результата. передаем в метод кол-во компонент связности, кол-во вершин, массив компонет связности - данные, которые надо вывести.

* *class JGraphAdapterDemo*

Методы:

private void adjustDisplaySettings( JGraph jg ) – настройки свойств экрана

* *class Window*

Методы:

private void setEdgeList(int amount) –заполнение таблицы инцидентности

public static void main(String[] args) –главная функция

* *class MyGUIForm*

Поля:

private JLabel descLabel – поле для комментариев алгоритма  
private JLabel resLabel – поля для вывода результата   
private JTextField graphEdit – поле для ввода входных данных  
private JButton buttonLoad – кнопка Load  
private JButton buttonInit – кнопка Init  
private JButton buttonStep – кнопка Step  
private JButton buttonRun – кнопка Run

Методы:

public static void main(String[] args) – главная функция

public void onButtonLoadPressed(Boolean clicked) – метод обработки при нажатии кнопки Load

buttonInit.addActionListener(new ActionListener() – открытие файла для чтения

buttonLoad.addActionListener(new ActionListener() – инициализация графа

* 1. **Алгоритм работы**

Для начала работы алгоритма необходимо ввести исходные данные: количество вершин, количество ребер, список инцидентности вершин. Это можно сделать, считав данные из файла, нажав на кнопку Load и выбрав нужный файл; либо можно данные ввести в окно в поле Graph data. Далее, нажав кнопку Init, загружается в память и выводится на экран.

Нажимая на кнопки Step и Run, алгоритм будет выполняться в пошаговом режиме и весь полностью. На данном этапе в окне будет выводиться графическая интерпретация алгоритма и краткие комментарии. После окончания работы на экран будет выведен результат работы программы: количество компонент сильной связности ориентированного графа.

1. **ТЕСТИРОВАНИЕ**

# Тестирование графического интерфейса

* 1. **Тестирование кода алгоритма**

**ЗАКЛЮЧЕНИЕ**

В ходе данной учебной практики была написана и протестирована программа, реализующая поиск компонент сильной связности ориентированного графа. Для реализации был использован алгоритм Косорайю. Программа позволяет, как демонстрировать пошаговую работу алгоритма, так и автоматически, с выводом кратким описанием.

В ходе написания данной работы были получены следующие знания и навыки:

* Основы программирования на языке Java
* Реализация графического интерфейса
* Работа с распределённой системой управления версиями Git и репозиторием GitHub

**СПИСОК ИСПОЛЬЗОВАННых ИСТОЧНИКОВ**

1. Java. Базовый курс. // Stepik. URL: [https://stepik.org/course/Java-Базовый-курс-187/syllabus](https://stepik.org/course/Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81-187/syllabus) (дата обращения: 22.06.2017).
2. Алгоритм Косарайю.// Википедия. URL: <https://ru.wikipedia.org/wiki/Алгоритм_Косорайю> (дата обращения 24.06.2017)
3. Брюс Эккель. Философия Java. СПб.: Питер, 2009.

**пРИЛОЖЕНИЕ а**