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

ОТЧЕТ

по учебной практике

Тема: Сильно связные компоненты орграфа

Студент гр. 7303 _____ Петров С.А.

Студент гр. 7303 _____ Юсковец А.В.

Студент гр. 7303 _____ Шаломов А.Д.

Руководитель _____ Размочаева Н.В.

Санкт-Петербург
2019

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Петров С.А. группы 7303

Студент Юсковец А.В. группы 7303

Студент Шаломов А.Д. группы 7303

Тема практики: Сильно связные компоненты орграфа.

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Косарайю.

Сроки прохождения практики: 01.07.2019 – 14.07.2019

Дата сдачи отчета: 00.07.2019

Дата защиты отчета: 00.07.2019

Студент _____ Петров С.А.

Студент _____ Юсковец А.В.

Студент _____ Шаломов А.Д.

Руководитель _____ Размочаева Н.В.

АННОТАЦИЯ

Целью прохождения данной учебной практики является выполнение мини-проекта, подразумевающего разработку визуализации алгоритма с графическим интерфейсом на объектно-ориентированном языке программирования Java. В данном варианте реализуется визуализация алгоритма Косарайю для поиска сильных компонент связности в ориентированном графе. Данный отчет содержит: описание реализации алгоритма и использованной структуры данных, описание основных методов, а также результаты тестирования графического интерфейса и кода алгоритма.

SUMMARY

The purpose of this educational practice is to perform a mini-project, which implies the development of a visualization of the algorithm with a graphical interface in the object-oriented programming language Java. In this case the Kosaraju's algorithm to find strong connected components in the directed graph visualization is implemented. This report contains a description of the implementation of the algorithm and the data structure used in project, a description of the main methods, as well as the results of testing the graphical interface and the code of the algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной учебной практики является разработка программы на языке Java с графическим интерфейсом, выполняющей визуализацию алгоритма. Графический интерфейс предоставляет пользователю удобный способ работы с программой: возможность выбора исходных данных, наглядное исполнение алгоритма.

В качестве конкретного алгоритма был выбран алгоритм Косарайю — алгоритм поиска областей сильной связности в ориентированном графе. Чтобы найти области сильной связности, сначала выполняется поиск в глубину (DFS) на обращении исходного графа (то есть против дуг), вычисляя порядок выхода из вершин. Затем используется обращение этого порядка, чтобы выполнить поиск в глубину на исходном графе (в очередной раз берём вершину с максимальным номером, полученным при обратном проходе).

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требование к входным данным

Граф хранится в виде списка инцидентности в формате:

<вершина> x y <смежная вершина> … <смежная вершина>

1.1.2. Требования к визуализации

При визуализации первой стадии алгоритма - поиска в глубину в инвертированном графе последовательно красить пройденные вершины, на второй стадии – красить найденные компоненты связности в разные цвета.

1.2. Уточнение требований после сдачи прототипа

Необходимо добавить возможность пошагового исполнения алгоритма.

1.3. Уточнение требований после сдачи 1-ой версии

Добавить проверку входных данных, на усмотрение команды перейти к хранению графа в формате json.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

- Спецификация 05.07.19
- Граф 08.07.19
- Алгоритм 08.07.19
- Интерфейс 10.07.19
- Тестирование 12.07.19

2.2. Распределение ролей в бригаде

- Петров Сергей – граф, первая стадия алгоритма
- Юсковец Антай – организация вики, вторая стадия алгоритма
- Шаломов Артем – интерфейс, архитектура проекта

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структура данных

Алгоритм Косарайю осуществляет поиск компонент связности в ориентированном графе, поэтому была реализована структура данных, содержащая представление графа – класс Graph. Класс содержит список вершин графа и список ребер, для данных список присутствуют геттеры, имеются методы добавления/удаления ребер и вершин, в классе переопределен метод toString(). Для удобства осуществления алгоритма присутствует метод invert(), который отвечает за инвертирование графа. Проект подразумевает отрисовку графа, поэтому частично реализуется паттерн Observer – в графе хранится ссылка на объект, реализующий интерфейс Listener, которому граф сообщает об изменении своего состояния.

Каждая вершина хранит в себе список исходящих ребер, координаты, цвет и имя. Каждое ребро содержит две ссылки: на начальную и конечную вершины ребра.

Код реализованной структуры данных представлен в приложении А.

3.2. Основные методы

3.2.1. Алгоритм

Для реализации алгоритма был заведен класс GraphAlgo, содержащий ссылку на граф, и выполняющий над ним необходимые действия. Так же класс содержит массив состояний, позволяющий осуществить как автоматическое исполнение алгоритма, так и пошаговое. Класс содержит три основных метода – DFS1_step() и DFS2_step(), для двух стадий алгоритма Косарайю, и главный метод Kosaraju(), который вызывает предыдущие два. Первый метод осуществляет поиск в глубину на инвертированном графе, замеряя время выхода из вершины. Второй метод осуществляет поиск в глубину на неинвертированном графе, каждый раз выбирая вершины с наибольшим временем выхода, полученным на первой стадии. Полученные компоненты

связности – сильные компоненты. В графе присутствуют вспомогательные метода для поиска в глубину и создания состояний графа.

3.2.2. Интерфейс

Графический пользовательский интерфейс разработан, используя классы стандартной библиотеки в пакете javax.swing и java.awt. Основной класс пользовательского интерфейса – MainWindow. Внешний вид окна представлен на рис. *.1.

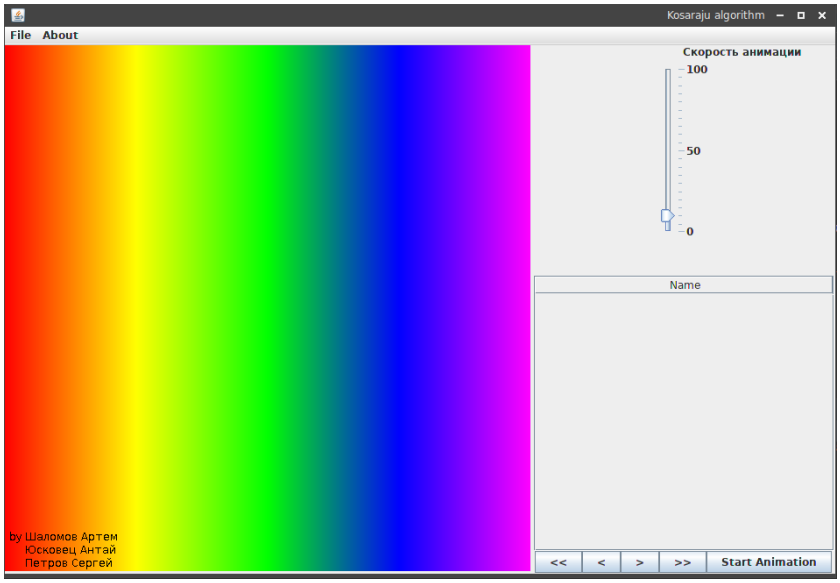

Рисунок 1 – Графический интерфейс программы

В левой части окна располагается граф (на рис. *.1 граф пустой), в правой сверху вниз – слайдер для выбора скорости демонстрации работы алгоритма, список вершин в порядке их обхода в алгоритме Косарайю, кнопки для переключения между этапами выполнения алгоритма: в начало, шаг вперед, шаг назад, в конец, включить/выключить автоматический переход. На рис. *.2 представлен пример графа с полностью завершенным алгоритмом Косарайю, каждая компонента связности окрашена в собственный цвет.

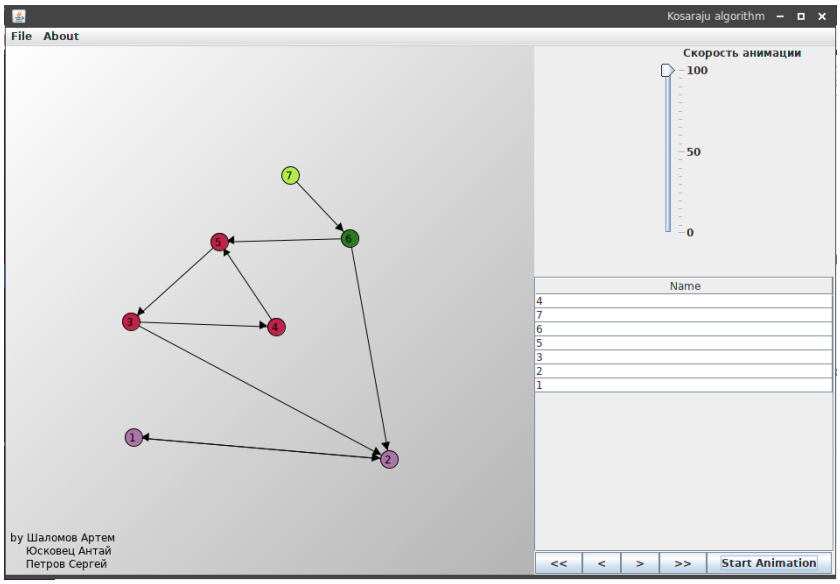

Рисунок 2 – Результат работы алгоритма

Так же предусмотрена возможность изменения графа. Например при нажатии правой кнопки мыши на панель отрисовки графа открывается меню в котором можно создать вершину или очистить весь граф. Вид этого меню изображен на рис. *.3.

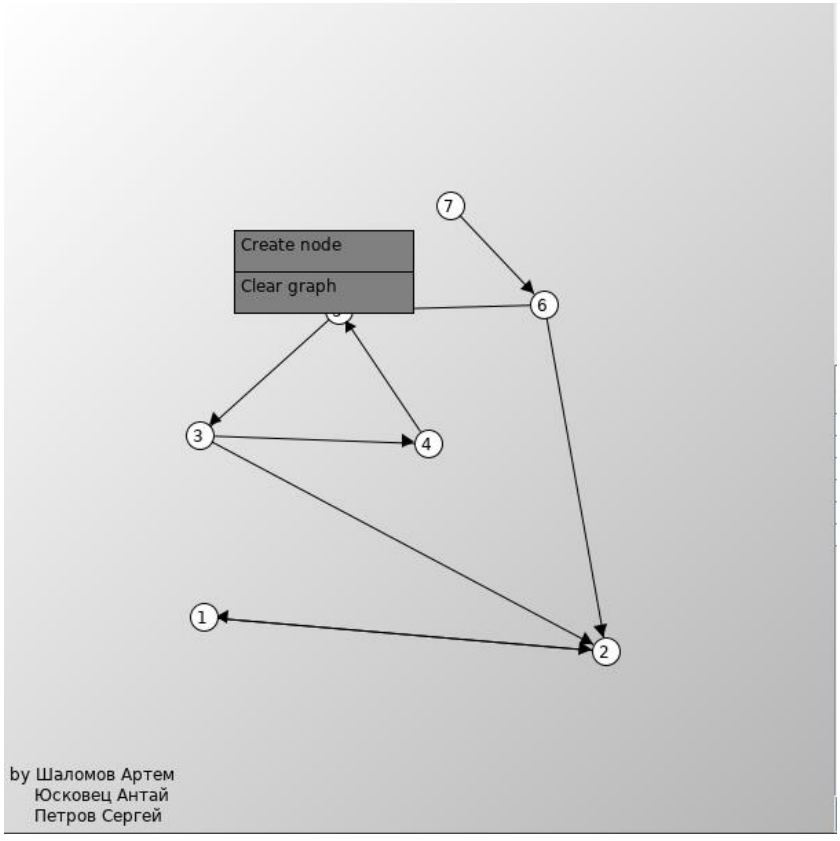

Рисунок 3 – Всплывающее меню панели графа

Также возможно изменение непосредственно вершины графа. При нажатии правой кнопкой мыши на вершину также открывается меню, изображенное на рис. *.4, позволяющее добавлять/удалять ребра, удалять и переименовывать саму вершину.

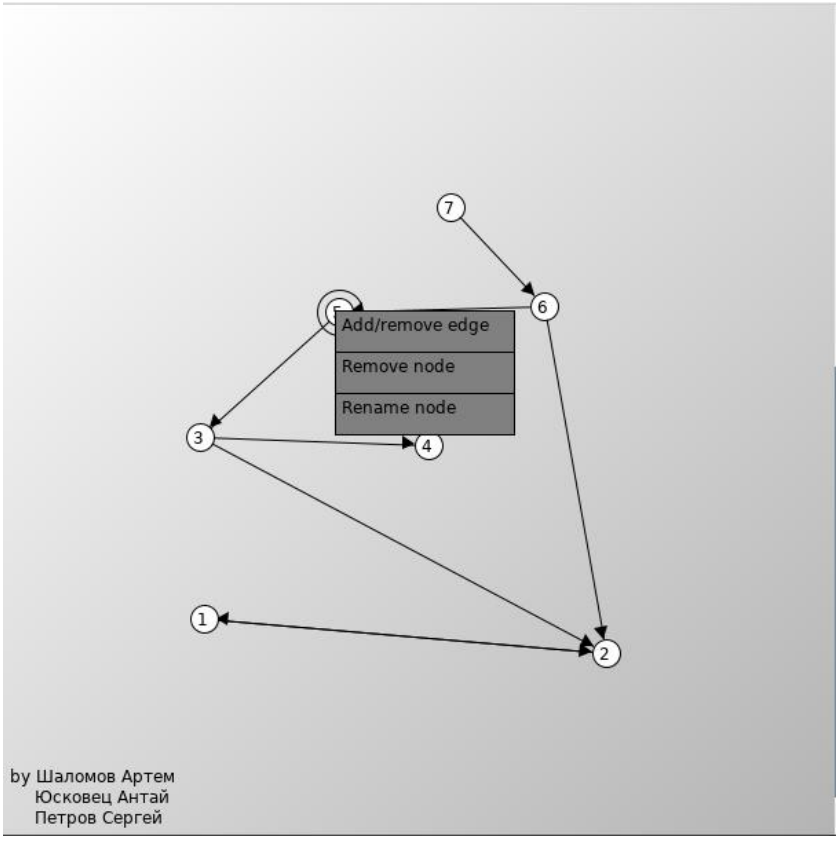

Рисунок 4 – Всплывающее меню вершины

Единственное ограничение при изменении графа: вершины должны иметь уникальные имена. При попытке именовать вершину уже имеющимся в графе названием появится уведомление, изображенное на рис. *5.

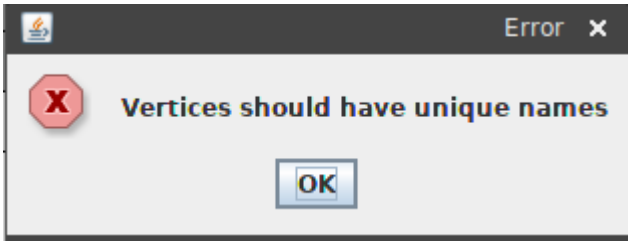

Рисунок 5 – Некорректное именование вершины

Возможно сохранение и чтение графа в виде файла, хранящегося в формате JSON. Вызвать эти функции пользователь может из меню File или при помощи сочетания клавиш Ctrl+O для чтения и Ctrl+S для сохранения.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Пользовательский интерфейс был протестирован в различных условиях. Корректно обрабатываются ошибки при неправильном формате входных данных, а также ситуации ошибочного поведения пользователя (например, создание вершины с уже существующим именем).

4.2. Тестирование кода алгоритма

Код алгоритма и основные методы графа были протестированы с помощью библиотеки JUnit в классе GraphAlgoTest.

5. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

5.1. Начальное состояние

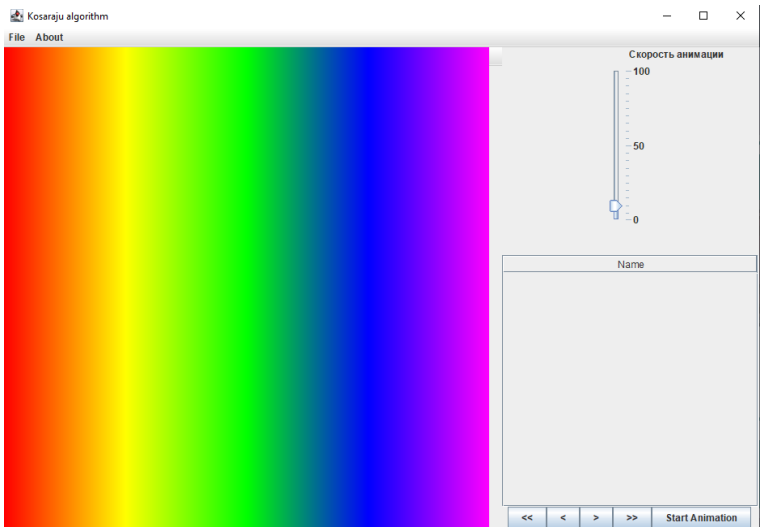

Рисунок 6 – начальное состояние программы

5.2. Создание вершин и ребер

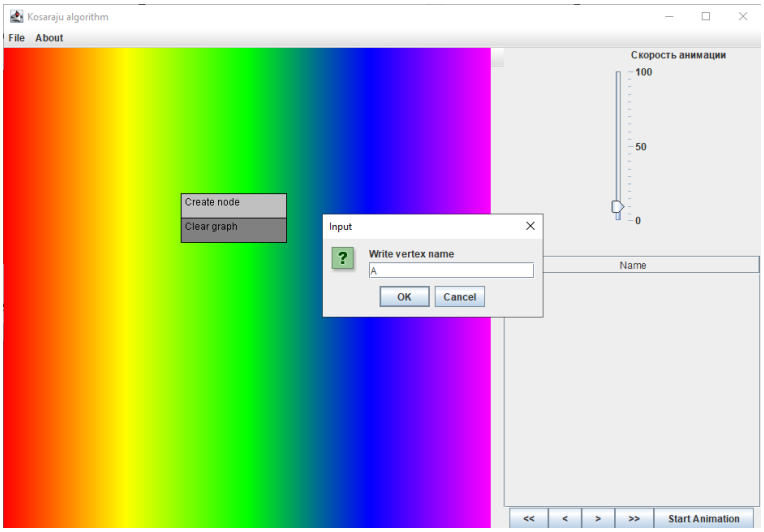

Рисунок 7 – меню выбора по клику правой кнопки мыши

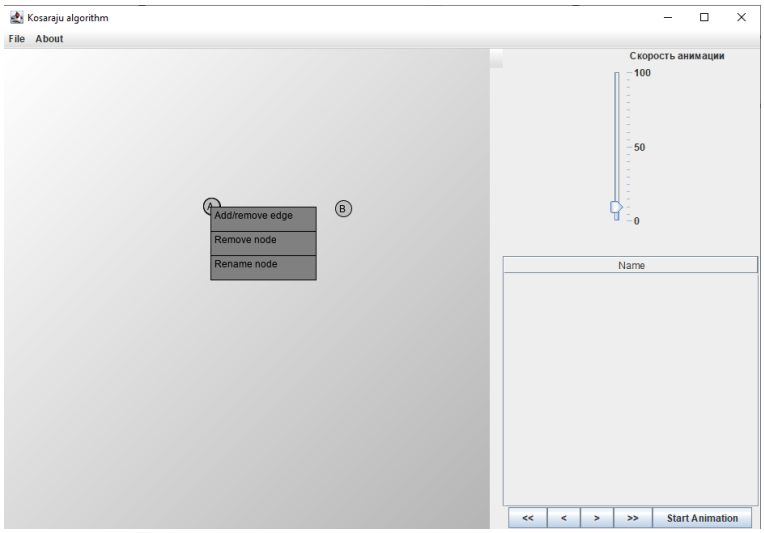

Рисунок 8 – создание ребра(выбор)

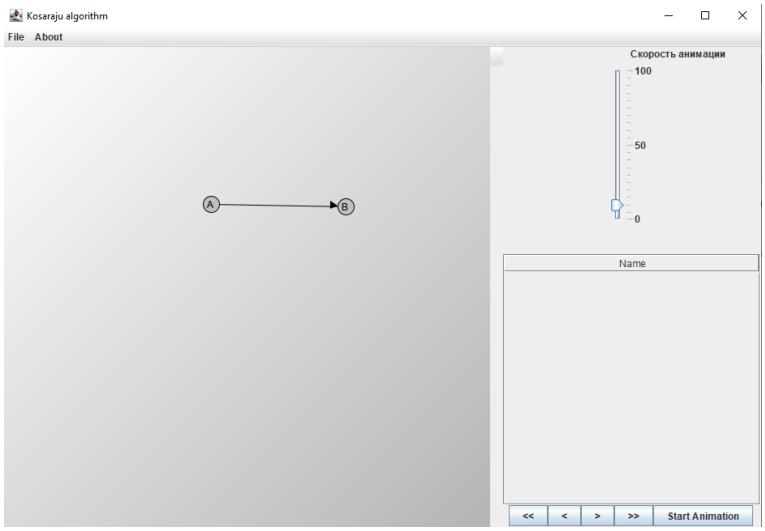

Рисунок 9 – создание ребра(полученное ребро)

5.3. Работа алгоритма

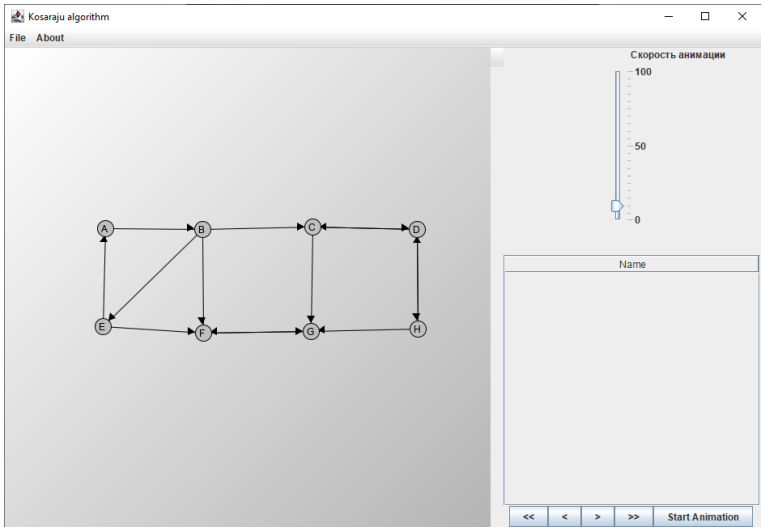

Рисунок 10 – начальное состояние графа

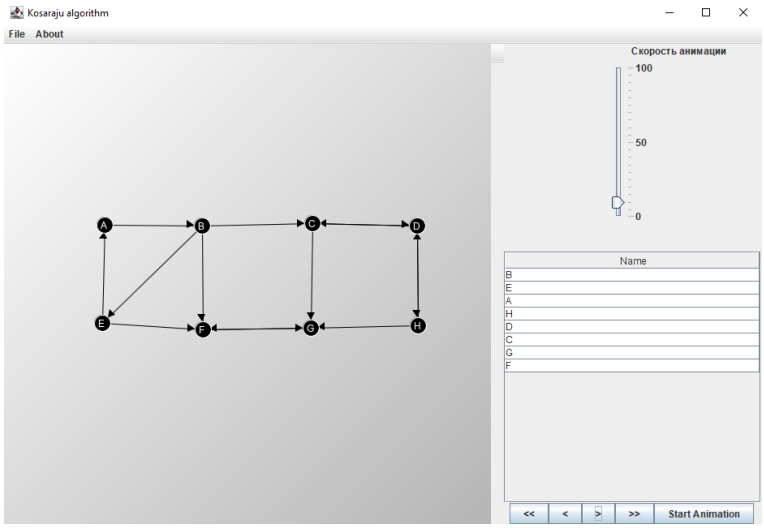

Рисунок 11 – состояние графа после первой стадии алгоритма

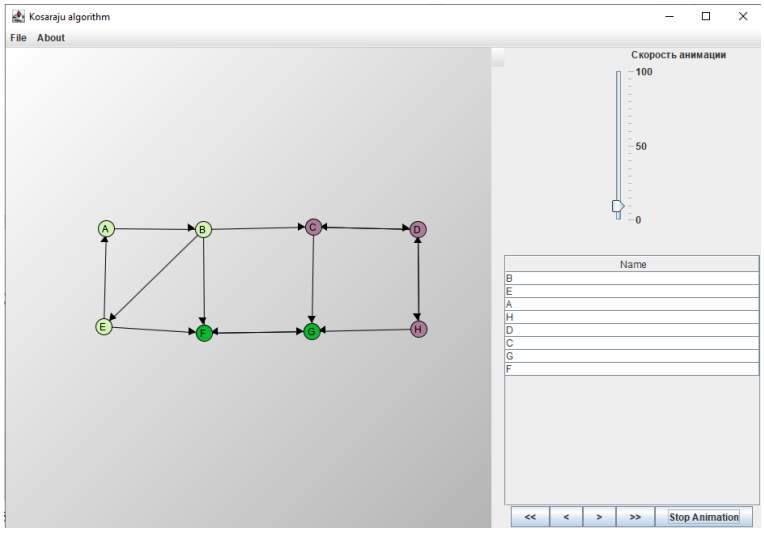

Рисунок 12 – финальное состояние графа

5.4. Пошаговое исполнение алгоритма алгоритма

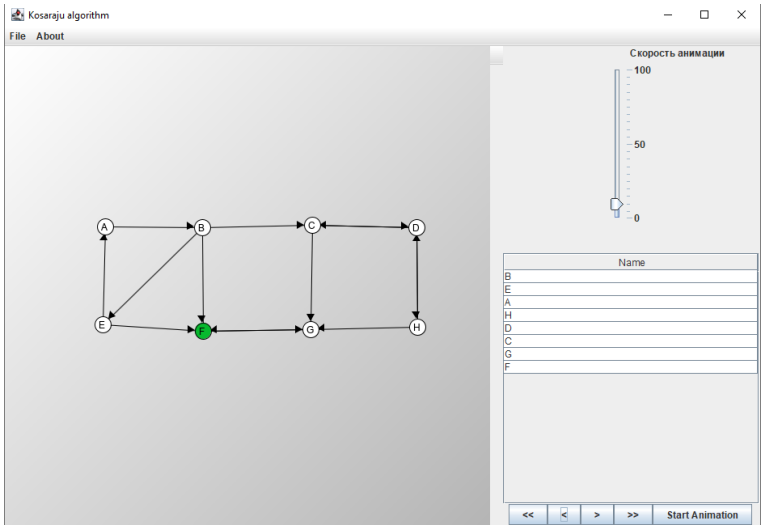

Рисунок 13 – состояние на n-м шаге

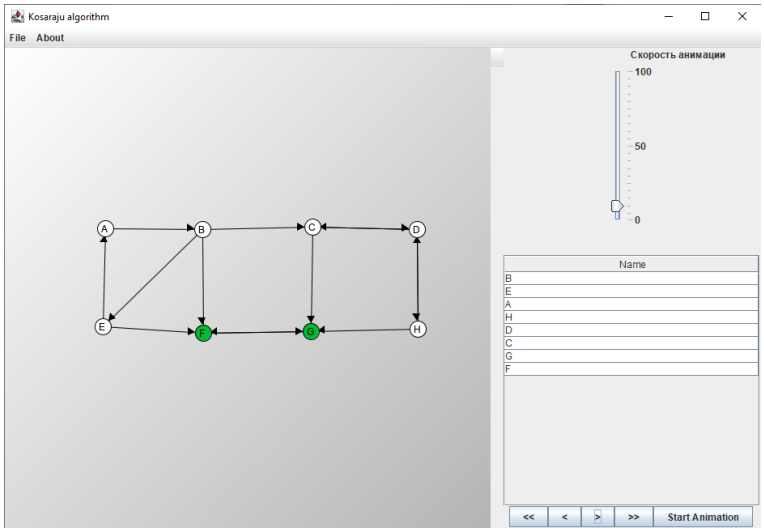

Рисунок 14 – состояние на n+1-м шаге

ЗАКЛЮЧЕНИЕ

В результате прохождения учебной практики был выполнен мини-проект по созданию программы на языке Java, осуществляющая визуализацию алгоритма Косарайю для нахождения сильных компонент связности в ориентированном графе. Программа предоставляет пользователю графический интерфейс, который позволяет выбрать или задать входные данные (граф), сохранить полученный граф, выполнить алгоритм в автоматическом или пошаговом режиме. Граф визуализируется в наглядном для пользователя виде, в результате работы алгоритма каждая компонента будет покрашена в отдельный цвет.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шилдт, Герберт Java 8. Руководство для начинающих / Герберт Шилдт. - М.: Вильямс, 2015. - 720 с.

2. Седжвик Роберт , Уэйн Кевин Алгоритмы на Java / Роберт Седжвик, Кевин Уэйн. - М.: Вильямс, 2016. - 848 с.

3. Официальная документация по Java // Oracle Help Center. URL: https://docs.oracle.com/en/.

4. Роберт Лафоре "Структуры данных и алгоритмы в Java", 2-е издание, (2013, PDF) // URL: https://vk.com/doc10903696_336361025?hash=175de31599461c95a4&dl=d07edbfb9aec9b6ceb

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Graph.java

```
package graph;

import java.util.*;

public class Graph {
    private List<Vertex> vertices;
    private List<Edge> edges;
    private Listener listener;

    private boolean isDirected = true;

    public Graph() {
        edges = new ArrayList<>();
        vertices = new ArrayList<>();
    }
    public Graph(boolean isDirected) {
        this();
        this.isDirected = isDirected;
    }

    public void onModify(Listener listener) {
        this.listener = listener;
    }

    @Deprecated
    public Graph(Graph g) {
        isDirected = g.isDirected();
        for (Vertex v : g.getVertices())
            this.vertices.add(new Vertex(v));

        for (Vertex v : this.getVertices()) {
            this.edges.addAll(v.getEdges());
        }
    }
    @Deprecated
    public Graph(Collection<Vertex> vertices, Collection<Edge> edges) {
        this(true, vertices, edges);
    }
    @Deprecated
    public Graph(boolean isDirected, Collection<Vertex> vertices,
Collection<Edge> edges) {
        this(isDirected);

        this.vertices.addAll(vertices);
        this.edges.addAll(edges);

        for (Edge e : edges) {
            final Vertex from = e.getSource();
            final Vertex to = e.getDest();

            if (!this.vertices.contains(from) || !this.vertices.contains(to))
                continue;

            from.addEdge(e);
            if (!this.isDirected) {
                Edge reciprical = new Edge(to, from);
                to.addEdge(reciprical);
```

```
            this.edges.add(reciprical);
        }
    }
}

public boolean createVertex(String name, float x, float y){
    if(getVertex(name) == null){
        Vertex v = new Vertex(name, x, y);
        vertices.add(v);
        listener.performAction();
        return true;
    }
    return false;
}

public boolean createVertex(String name) {
    return createVertex(name, 0, 0);
}

public boolean createEdge(String src, String dest){
    Vertex from = getVertex(src);
    Vertex to = getVertex(dest);
    if(from.isConnected(to)) return false;
    Edge e = new Edge(from, to);
    from.addEdge(e);
    edges.add(e);
    listener.performAction();
    return true;
}

public boolean createEdge(Vertex src, Vertex dest){
    if(src.isConnected(dest)) return false;
    Edge e = new Edge(src, dest);
    src.addEdge(e);
    edges.add(e);
    listener.performAction();
    return true;
}
public boolean removeEdge(Vertex src, Vertex dest){
    if(!src.isConnected(dest))return false;
    Edge delEdge = src.getEdge(dest);
    src.getEdges().remove(delEdge);
    edges.remove(delEdge);
    listener.performAction();
    return true;
}
public boolean removeVertex(Vertex v){
    if(!vertices.contains(v)) return false;
    vertices.remove(v);
    edges.removeIf(e -> e.getDest() ==v || e.getSource() == v);
    listener.performAction();
    return true;
}

public boolean isDirected() {
    return isDirected;
}

public List<Vertex> getVertices() {
    return vertices;
}

public List<Edge> getEdges() {
```

```
        return edges;
    }

    public Vertex getVertex(String name){
        for(Vertex v : vertices){
            if(v.getName().equals(name))
                return v;
        }
        return null;
    }

    public void invert(){
        for(Edge e : edges){
            e.invert();
        }
    }

    public void clear() {
        edges.clear();
        vertices.clear();
    }

    @Override
    public int hashCode() {
        int code = ((Boolean)this.isDirected).hashCode() + this.vertices.size()
+ this.edges.size();
        for (Vertex v : vertices)
            code *= v.hashCode();
        for (Edge e : edges)
            code *= e.hashCode();
        return 31 * code;
    }

    @Override
    public String toString() {
        final StringBuilder builder = new StringBuilder();
        for (Vertex v : vertices)
            builder.append(v.toString());
        return builder.toString();
    }
}
```

Файл Vertex.java

```
package graph;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;

public class Vertex {

    private float x,y;
    private String name;
    private List<Edge> edges = new ArrayList<>();
    private Color color;

    Vertex(String name) {
        this.name = name;
    }

    Vertex(String name, float x, float y) {
        this.name = name;
        this.x = x;
        this.y = y;
```

```
    this.color = Color.WHITE;
}
@Deprecated
Vertex(Vertex vertex) {
    this(vertex.name, vertex.x, vertex.y);
    this.edges.addAll(vertex.edges);
}

public String getName() {
    return name;
}
public void setName(String name){
    this.name = name;
}

public void addEdge(Edge e) {
    edges.add(e);
}

public List<Edge> getEdges() {
    return edges;
}

public Edge getEdge(Vertex v) {
    for (Edge e : edges) {
        if (e.getDest().equals(v))
            return e;
    }
    return null;
}

public float getX() {
    return x;
}

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

public float getY() {
    return y;
}

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

public Color getColor() {
    return color;
}

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

public boolean isConnected(Vertex v) {
    for (Edge e : edges) {
        if (e.getDest().equals(v))
            return true;
    }
    return false;
}
```

```
    @Override
    public int hashCode() {
        final int code = this.name.hashCode() + this.edges.size();
        return 31 * code;
    }

    @Override
    public String toString() {
        final StringBuilder builder = new StringBuilder();
        builder.append("Value=").append(name)
                .append(" x: ").append(this.x)
                .append(" y: ").append(this.y)
                .append("\n");
        for (Edge e : edges)
            builder.append("\t").append(e.toString());
        return builder.toString();
    }
}
```

Файл Edge.java

```
package graph;

public class Edge{
    private Vertex source,dest;

    Edge(Vertex source, Vertex dest) {
        if (source == null || dest == null)
            throw (new NullPointerException("Both 'dest' and 'source' vertices
need dest be non-NULL."));
        this.source = source;
        this.dest = dest;
    }
    @Deprecated
    public Edge(Edge e) {
        this(e.source, e.dest);
    }

    public Vertex getSource() {
        return source;
    }

    public Vertex getDest() {
        return dest;
    }

    public void invert() {
        source.getEdges().remove(this);
        dest.getEdges().add(this);

        Vertex temp = dest;
        dest = source;
        source = temp;
    }

    @Override
    public int hashCode() {
        final int cost = (this.getSource().hashCode() *
this.getDest().hashCode());
        return 31 * cost;
    }

    @Override
    public String toString() {
        return "[" + source.getName() + "]" + " -> " +
```

```
                "[" + dest.getName() + "]" + "\n";
    }
}
```

Файл Listener.java

```
package graph;

public interface Listener {
    void performAction();
}
```

Файл VerticesList.java

```
package graph;

import javax.swing.table.DefaultTableModel;
import java.util.ArrayList;
import java.util.List;

public class VerticesList {
    private DefaultTableModel model;
    private List<Vertex> vertices = new ArrayList<>();

    public VerticesList(DefaultTableModel model) {
        this.model = model;
    }

    void append(Vertex vertex) {
        model.addRow(new Object[]{vertex.getName()});

        vertices.add(vertex);
    }

    Vertex getNext() {
        int lastElIndex = vertices.size() - 1;

        Vertex next = vertices.get(lastElIndex);
        vertices.remove(lastElIndex);

        return next;
    }

    boolean hasNext() {
        return !vertices.isEmpty();
    }

    void remove(Vertex vertex) {
        vertices.remove(vertex);
    }

    void clear() {
        for (int rowIndex = model.getRowCount()-1; rowIndex >= 0; --rowIndex) {
            model.removeRow(rowIndex);
        }

        vertices.clear();
    }
}
```

Файл GraphAlgo.java

```
package graph;

import java.awt.*;
import java.util.ArrayList;
import java.util.Random;
import java.util.function.Supplier;
```

```
public class GraphAlgo {
    private ArrayList<ArrayList<Color>> states;
    private Graph graph;

    public GraphAlgo() { }

    private void DFS1_step(Graph graph, VerticesList list) {
        //красит только из белого в черный
        this.graph = graph;
        states.add(createState());

        for(Vertex v : graph.getVertices()){
            if(v.getColor() == Color.WHITE)
                visit(v, list);
        }
    }


    private void DFS2_step(Graph graph, VerticesList list) {
        states.add(createState());
        this.graph = graph;

        ArrayList<Color> usedColors = new ArrayList<>();
        Supplier<Color> randomColor = () -> {
            Random rand = new Random();

            do {
                float r = rand.nextFloat();
                float g = rand.nextFloat();
                float b = rand.nextFloat();

                Color color = new Color(r, g, b);
                if (!usedColors.contains(color)) {
                    usedColors.add(color);
                    return color;
                }
            } while (true);
        };

        while (list.hasNext()) {
            Color color = randomColor.get();

            Vertex v = list.getNext();
            if (v.getColor() == Color.WHITE) {
                visit(v, color, list);
            }
        }
    }

    public ArrayList<ArrayList<Color>> Kosaraju(Graph graph, VerticesList list){
        states = new ArrayList<>();
        list.clear();

        for(Vertex v : graph.getVertices()){
            v.setColor(Color.WHITE);
        }

        graph.invert();
        DFS1_step(graph, list);
        graph.invert();

        for(Vertex v : graph.getVertices()){
```

```
            v.setColor(Color.WHITE);
        }
        DFS2_step(graph, list);

        return states;
    }


    private void visit(Vertex v, Color color, VerticesList list) {
        v.setColor(color);
        list.remove(v);

        states.add(createState());
        for (Edge e : v.getEdges()) {
            if(e.getDest().getColor() == Color.WHITE) {
                visit(e.getDest(), color, list);
            }
        }
    }

    private void visit(Vertex v, VerticesList list) {
        v.setColor(Color.BLACK);
        states.add(createState());
        for (Edge e : v.getEdges()) {
            if(e.getDest().getColor() == Color.WHITE) {
                visit(e.getDest(), list);
            }
        }

        list.append(v);
    }

    private ArrayList<Color> createState(){
        ArrayList<Color> state = new ArrayList<>();
        graph.getVertices().forEach(el->state.add(el.getColor()));
        return state;
    }
}
```

Файл GraphAlgoTest.java

```
package graph;

import junit.framework.TestCase;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.util.List;
import java.util.*;

public class GraphAlgoTest extends TestCase {
    private Graph randomGraph;
    private Graph graph;
    private GraphAlgo algo;

    @Before
    public void setUp() throws Exception {
        algo = new GraphAlgo();

        // randomGraph
        randomGraph = new Graph();
        Random r = new Random();
```

```
        int min = 5;
        int max = 100;
        int vert = min + r.nextInt(max-min+1);
        int from, to;
        for(int i = 0; i < vert; i++)
            randomGraph.createVertex(String.valueOf(i));

        randomGraph.createEdge(String.valueOf(0), String.valueOf(1));
        for(int i = 1; i < vert; i++){
            from = r.nextInt(vert);
            to = r.nextInt(vert);
            randomGraph.createEdge(String.valueOf(from), String.valueOf(to));
        }

        // special graph
        graph = new Graph();
        graph.createVertex("A");
        graph.createVertex("B");
        graph.createVertex("C");
        graph.createVertex("D");
        graph.createVertex("E");
        graph.createVertex("F");
        graph.createVertex("G");
        graph.createVertex("H");

        graph.createEdge("A", "B");

        graph.createEdge("B", "C");
        graph.createEdge("B", "E");
        graph.createEdge("B", "F");

        graph.createEdge("C", "D");
        graph.createEdge("C", "G");

        graph.createEdge("D", "H");
        graph.createEdge("D", "C");

        graph.createEdge("E", "A");
        graph.createEdge("E", "F");

        graph.createEdge("F", "G");

        graph.createEdge("G", "F");

        graph.createEdge("H", "G");
        graph.createEdge("H", "D");
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testDFS1VisitAllVertices() {
        DefaultTableModel model = new DefaultTableModel();
        model.addColumn("Name");
        VerticesList list = new VerticesList(model);

        for(Vertex v : randomGraph.getVertices()){
            v.setColor(Color.WHITE);
        }
        algo.DFS1_step(randomGraph, list);
        for(Vertex v : randomGraph.getVertices()){
```

```
            assertEquals(v.getColor(), Color.BLACK);
        }
    }

    @Test
    public void testDFS1Timeout() {
        DefaultTableModel model = new DefaultTableModel();
        model.addColumn("Name");
        VerticesList list = new VerticesList(model);

        for(Vertex v : graph.getVertices()){
            v.setColor(Color.WHITE);
        }
        algo.DFS1_step(graph, list);
        List<String> expected = Arrays.asList("A", "B", "E", "C", "D", "H", "G",
"F");
        Iterator it = expected.iterator();
        while(list.hasNext() && it.hasNext()){
            assertEquals(it.next(), list.getNext().getName());
        }
    }

    @Test
    public void testDFS2() {
        DefaultTableModel model = new DefaultTableModel();
        model.addColumn("Name");
        VerticesList list = new VerticesList(model);

        ArrayList<ArrayList<Color>> states = algo.Kosaraju(graph, list);
        ArrayList<Color> lastState = states.get(states.size()-1);

        // (A, B, E) (C, D, H) (F, G)
        for (int i = 0; i != lastState.size(); ++i) {
            graph.getVertices().get(i).setColor(lastState.get(i));
        }

        Color aColor = graph.getVertex("A").getColor();
        Color bColor = graph.getVertex("B").getColor();
        Color cColor = graph.getVertex("C").getColor();
        Color dColor = graph.getVertex("D").getColor();
        Color eColor = graph.getVertex("E").getColor();
        Color fColor = graph.getVertex("F").getColor();
        Color gColor = graph.getVertex("G").getColor();
        Color hColor = graph.getVertex("H").getColor();

        // (A, B, E)
        assertEquals(aColor, bColor);
        assertEquals(bColor, eColor);

        // (C, D, H)
        assertEquals(cColor, dColor);
        assertEquals(dColor, hColor);

        // (F, G)
        assertEquals(fColor, gColor);
    }

    @Test
    public void testClear() {
        randomGraph.clear();
        assertEquals(0, randomGraph.getVertices().size());
        assertEquals(0, randomGraph.getEdges().size());
    }
```

```
    @Test
    public void testAddVertex() {
        String name = "Vertex #" + randomGraph.getVertices().size() + 1;
        assertTrue(randomGraph.createVertex(name));
        assertFalse(randomGraph.createVertex(name));
    }

    @Test
    public void testRemoveVertex() {
        Random r = new Random();
        Vertex v =
randomGraph.getVertices().get(r.nextInt(randomGraph.getVertices().size()));
        assertTrue(randomGraph.removeVertex(v));
        assertFalse(randomGraph.removeVertex(v));
    }

    @Test
    public void testAddEdge() {
        Random r = new Random();
        String name = randomGraph.getVertices().size()+"";
        randomGraph.createVertex(name + 1);
        randomGraph.createVertex(name + 2);
        assertTrue(randomGraph.createEdge(name + 1, name + 2));
        assertFalse(randomGraph.createEdge(name + 1, name + 2));
    }

    @Test
    public void testRemoveEdge() {
        Random r = new Random();
        Vertex from = null;
        for(Vertex v : randomGraph.getVertices()){
            if(v.getEdges().size() > 0){
                from = v;
                break;
            }
        }
        Vertex to = from.getEdges().get(0).getDest();
        assertTrue(randomGraph.removeEdge(from, to));
        assertFalse(randomGraph.removeEdge(from, to));
    }
}
```