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 программирования Java

Студент гр. 7304 _____ Давыдов А.А.

Студент гр. 7304 _____ Пэтайчук Н.Г.

Студентка гр. 7383 _____ Чемова К.А.

Руководитель _____ Размочаева Н.В.

Санкт-Петербург
2019

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Давыдов А.А. группы 7304

Студент Пэтайчук Н.Г. группы 7304

Студентка Чемова К.А. группы 7383

Тема практики: Визуализация алгоритмов на языке программирования Java

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: алгоритм поиска сильных компонент связности в ориентированном графе (алгоритм Косарайю).

Сроки прохождения практики: 01.07.2019 – 14.07.2019

Дата сдачи отчёта: 10.07.2019

Дата защиты отчёта: 12.07.2019

Студент гр. 7304	_____	Давыдов А.А.
Студент гр. 7304	_____	Пэтайчук Н.Г.
Студентка гр. 7383	_____	Чемова К.А.
Руководитель	_____	Размочаева Н.В.

АННОТАЦИЯ

Целью данной практической работы является совершенствование навыков программирования, обучение созданию программ, обладающих графическим интерфейсом пользователя, а также работе в команде.

Основным заданием практической работы является разработка в бригаде из 3 человек программы, визуализирующей работу одного из алгоритмов на графах. Наша бригада в качестве визуализируемого алгоритма выбрала алгоритм поиска компонент сильной связности в ориентированном графе, а именно алгоритм Косарайю.

SUMMARY

The purpose of this practical work is to improve programming skills, training in creating programs with a graphical user interface, as well as teamwork.

The main task of practical work is the development of a program in a brigade of 3 people that visualizes the work of one of the algorithms on graphs. Our team has chosen the algorithm for finding strongly connected components in a directed graph as the visualized algorithm, namely the Kosaraju algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной практики является обучение работе в команде для разработки полноценных графических приложений в итеративном режиме.

Задачами данной практики являются:

1. Прохождение вводного задания, связанного с освоением языка программирования Java;
2. Создание мини-проекта в бригаде из 3 человек, представляющего из себя программу, визуализирующую работу одного из алгоритмов на графах. Мини-проект должен быть выполнен на языке программирования Java;

Нашей бригадой в качестве визуализируемого алгоритма выбран алгоритм Косарайю, решающий задачу поиска компонент сильной связности в ориентированном графе. Описание входных/выходных данных и требований к программе, графического интерфейса, особенностей реализации программы и тестирования как интерфейса, так и кода, будет приведено в дальнейших разделах.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Программа должна представлять из себя оконное приложение с возможностью рисования графа, на котором будет применяться алгоритм, и управляющими кнопками для управления визуализацией данного алгоритма.

1.1.1. Требования к входным данным

Программа должна уметь считывать нарисованный пользователем граф и суметь визуализировать алгоритм Косарайю.

1.1.2. Требования к выходным данным

Программа должна визуализировать сильные компоненты связности, раскрашивая их в разные цвета. Также информация о процессе визуализации должна выводиться в лог, оформленный в виде текстового поля.

1.2. Уточнение требований после сдачи прототипа

- Программа должна уметь открывать файлы с графами в формате .json и обрабатывать их;
- В программе должна присутствовать возможность просмотреть справку о том, как пользоваться программой, и контактную информацию о создателях данного приложения;
- Снизу программы должна находиться надпись с именами создателей программы;
- Не должно быть кнопок, отвечающих за рисование графа: граф должен рисоваться только с помощью мыши в окошке поля графа;

1.3. Уточнение требований после сдачи 1-й версии программы

Уточнения требований не было, необходимо реализовать все предыдущие требования.

1.4. Уточнение требований после сдачи 2-й версии программы

Уточнений требований не было, вторая версия признана финальной версией программы.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

- 5 июля - создание прототипа программы (графического интерфейса приложения);
- 8 июля - создание 1-й версии программы (альфа-версии), реализующей основной функционал программы: рисование графа при помощи левой и правой кнопок мыши, пошаговая визуализация алгоритма и управление ею при помощи кнопок;
- 10 июля - создание 2-й версии программы (бета-версии), добавляющей вспомогательный функционал программы: считывание графа из файла формата .json, использование чёрного текстового поля в качестве лога, реализация раздела меню «Справка» и «Контакты»; составление отчёта по практике;
- 12 июля - создание финальной версии программы, со всем требуемым функционалом и полностью протестированный и избавленный от багов; составление отчёта по практике;

2.2. Распределение ролей в бригаде

- Давыдов А.А. - разработка основной логики программы, а именно представление графа и самого алгоритма Косарайю; разработка класса-адаптера между логикой и интерфейсом;
- Пэтайчук Н.Г. - разработка графического интерфейса программы; разработка класса-адаптера между логикой и интерфейсом;
- Чемова К.А. - тестирование графического интерфейса и логики программы, создание юнит-тестов: помощь в разработке всех компонентов программы;

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Использованные структуры данных

В качестве структуры данных, на которой основывается граф, используется хэш-таблица, реализующая интерфейс «ключ-значение» (HashMap), где в качестве ключей выступают имена вершин, а в качестве значений - списки вершин, куда можно прийти напрямую из вершины-ключа. Помимо этого, используются также массив строк для реализации пошагового исполнения алгоритма Косарайю, а также списки для хранения компонент связности, для хранения очереди, по которой нужно просматривать граф поиском глубину во второй раз, для хранения непосещённых вершин и трека поиска в глубину. Для реализации поиска в глубину также использовалась такая структура данных, как стек.

3.2. Основные методы

1) Class Graph:
   a) ***public Graph transposeGraph()*** - метод, транспонирующий граф. В качестве возвращаемого значения выступает объект класса Graph, являющийся транспонированной версией того графа, который вызвал этот метод;
2) Class KosarajuAlgorithm:
   a) ***private List<String> tOutDepthTraversal(Graph g)*** - метод, выполняющий первую часть работы алгоритма Косарайю, а именно первый обход в глубину транспонированного графа и создание очереди, по которой нужно проходить во второй раз обходом в глубину. В качестве аргумента принимает объект типа Graph, который считается транспонированной версией графа, в котором ищутся компоненты сильной связности. Возвращает данный метод

список, по которому нужно проходить во второй раз обходом в глубину (инвертированный список вершин в порядке их выхода из DFS);

b) ***private List<String> depthFirstTraversal(List<String> list_p)*** - метод, выполняющий второй обход в глубину уже изначального графа по списку, в котором вершины нужно обходить. Данный список является аргументом, который принимает данный метод. Возвращаемое значение - список вершин, разделённых значением null по признаку вхождения в одну компоненту сильной связности;

c) ***public void createGraph(List<String> vertexes, List<String[]> edges)*** - метод, инициализирующий граф (объект класса Graph) в объекте класса KosarajuAlgorithm, над которым в дальнейшем будет выполняться алгоритм Косарайю. Аргументы функции - список имён вершин и список рёбер (массивов строк размера 2, где первая строка является именем вершины начала, а вторая - вершины конца);

3) Class VisualKosarajuAlgorithmProxy:

a) ***private void prepareGraphData()*** - метод, преобразующий структуру данных хранения графа из класса VisualKosarajuWindow в те структуры данных, из которых можно создать граф для объекта KosarajuAlgorithm. Ничего не принимает, ничего не возвращает;

b) ***String[] createAlgorithmStepTrace()*** - метод, который преобразует два трека выполнения поисков в глубину в один единый трек, необходимый для визуализации алгоритма. В качестве возвращаемого значения выступает массив строк, являющийся единым треком (если поиск в глубину вышел из графа, то в шаге алгоритма будет пустая вершина, признаком транспонирования и конца работы алгоритма является значение null в массиве);

4) Class VisualKosarajuWindow:

a) ***private void createGraphComponent()*** - метод, создающий в окне поле для отображения и рисования графа, из которого можно извлечь

данные о самом нарисованном графе. Ничего не принимает, ничего не возвращает;

b) ***private void addVertex(String vertexName, int x, int y)*** - метод, добавляющий вершину в граф. Переменная vertexName - имя вершины, x и y - координаты, куда добавить вершину. Ничего не возвращает;

c) ***private void addEdge(String sourceVertex, String targetVertex)*** - метод, добавляющий ребро в граф. Переменные sourceVertex и targetVertex - имена вершины начала и вершины конца ребре соответственно. Ничего не возвращает;

d) ***private void deleteGraph()*** - метод удаления всего графа. Ничего не принимает, ничего не возвращает;

e) ***private void deleteVertex(Object vertexToDelete)*** - метод удаления вершины в графе. Принимает в качестве аргумента объект, инкапсулирующий в себе вершину графа. Ничего не возвращает;

f) ***private void changeColor(Object graphElement, String color)*** - метод, раскрашивающий вершину в некоторый цвет. Аргументы - объект, инкапсулирующий в себе вершину графа, и строка в формате «#HHHHHH», где HHHHHH - шестнадцатеричный код цвета, в который нужно окрасить вершину. Ничего не возвращает;

g) ***private void graphTransposition()*** - метод, транспонирующий нарисованный граф. Ничего не принимает, ничего не возвращает;

h) ***private void colorConnectivityComponents()*** - метод, раскрашивающий компоненты связности в различные цвета. Ничего не принимает, ничего не возвращает;

i) **private void discolorVertexes()** - метод, который возвращает вершинам стандартный цвет. Ничего не принимает, ничего не возвращает;

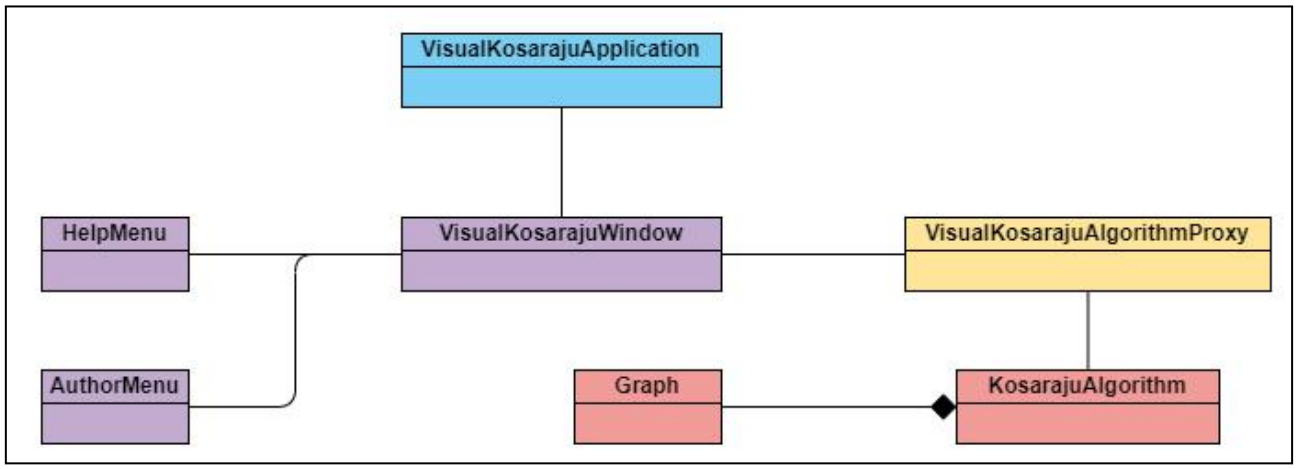

Рис 3.1 - UML-диаграмма классов проекта

4. ТЕСТИРОВАНИЕ ПРОГРАММЫ

Было проведено тестирование мини-проекта на корректность работы. Для этого были написаны тесты, проверяющие работу функций алгоритма, а также проверена работа графического интерфейса.

Тесты представляют собой набор юнит-тестов на разных исходных данных. Перед каждым юнит-тестом происходит инициализация графа `tka` (`tka` – является объектом `KosarajuAlgorithm`) при указании метки `@Before`. В юнит-тестах были использована функция `assertEquals()` из `org.junit`. `testTranposeGraph()` проверяет правильность транспонирования графа. `testTOutDepthTraversal()` проверяет, корректно ли возвращается список вершин по времени их выхода из обхода графа поиском в глубину. `testDepthFirstTraversal()` проверяет правильность возвращения списка компонент связности, разделителем компонент является null. `testGetTranspositionStepTrace()` проверят список обхода вершин в транспонированном графе, а `testGetOriginalStepTrace()` – в исходном.

Все тесты были успешно пройдены, это показано на рис. 1–4.

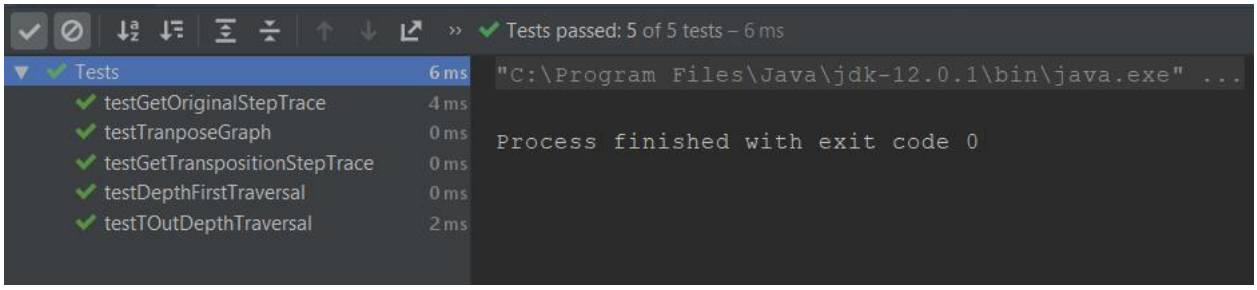

Рисунок 4.1 – Запуск Tests

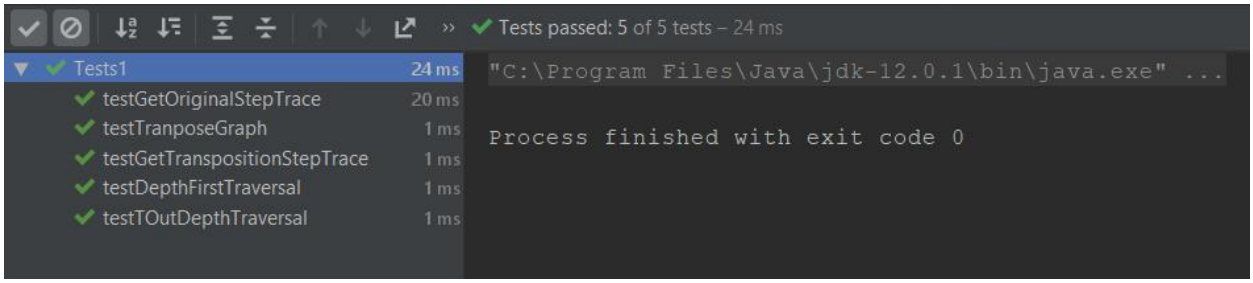

Рисунок 4.2 – Запуск Tests1

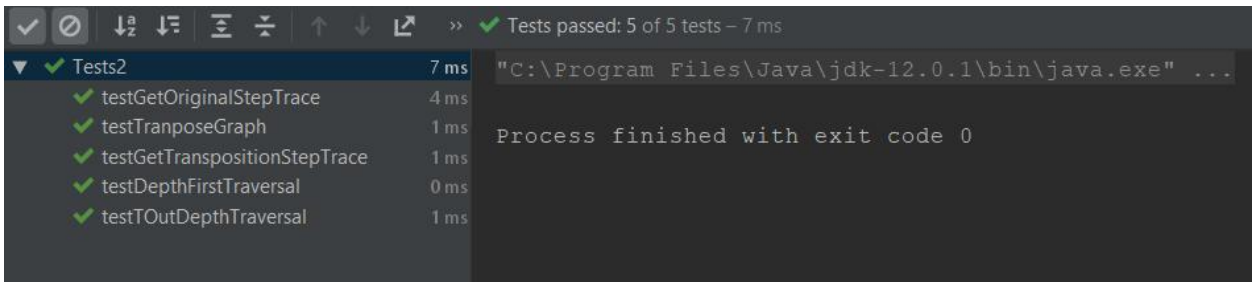

Рисунок 4.3 – Запуск Tests2

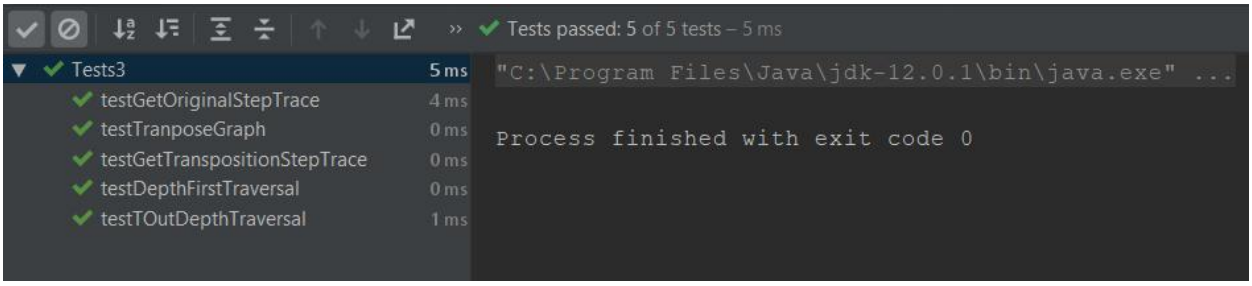

Рисунок 4.4 – Запуск Tests3

Также была проверена работа алгоритма при открытии графа из файла с расширением .json. При открытии корректного файла программа работает правильно. Если файл имеет другое расширение, то выводится сообщение об ошибке. Также при попытке открытия файла с неверными данными выводится сообщение об ошибке. Программа не позволяет открывать файл во время работы алгоритма, для его завершения необходимо нажать «Сброс», и после этого начать работу с файлом. На рис. 5 показаны сообщения об ошибках.

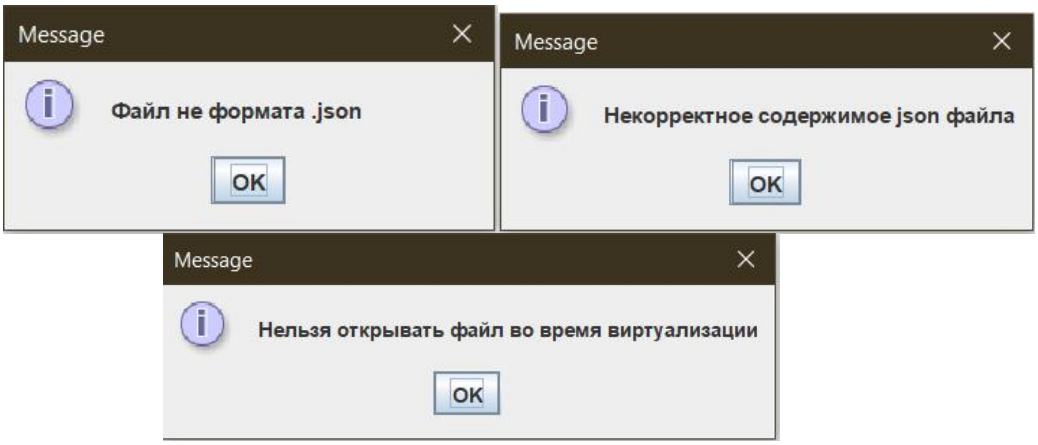

Рисунок 5 – Сообщения об ошибках

ЗАКЛЮЧЕНИЕ

В ходе практической работы были получены следующие результаты:

- Были изучены основы языка программирования Java за счёт выполнения вводного задания;
- Была разработана программа на языке программирования Java, визуализирующая алгоритм на графах (в данном случае алгоритм Косарайю поиска сильных компонент связности в ориентированном графе);
- Были получены навыки работы в команде и сопровождения программы в рамках группы разработчиков;

Кроме того, были получены навыки работы со сторонними библиотеками, которые можно найти в Интернете и которые созданы для облегчения решения какой-либо задачи (в данном проекте из подобных библиотек использовались JGraphX для визуализации и рисования графов и JSON simple для обработки файлов формата .json).

Можно утверждать, что основные цели данной практической работы были достигнуты, а потому можно сказать, что данная практическая работа была успешной.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Герасимова Т.В. Лабораторный практикум по программированию на языке JAVA СПБ.: Изд-во СПбГЭТУ «ЛЭТИ», 2012. 52 с.;

2. Герберт Шилдт Java 8. Руководство для начинающих, 6-е издание: Пер. с англ. — М.: Издательский дом "Вильямс", 2015. 712 с.;

3. Хорстманн К. С, Корнелл Г. Библиотека профессионала. Java 2. Том 1. Основы.: Пер. с англ. — М.: Издательский дом "Вильямс", 2003. 848 с;

4. Java. Методы программирования : уч.-мет. пособие / И.Н. Блинов, В.С. Романчик. — Минск : издательство «Четыре четверти», 2013. 896 с.;

5. Блох, Джошуа Java: эффективное программирование, 3-е изд.: Пер. с англ. — СПб. : ООО “Диалектика”, 2019. 464 с;

ПРИЛОЖЕНИЕ А

VISUAL KOSARAJU - ИСХОДНЫЙ КОД

- Graph.java

```
package VisualKosarajuLogic;

import java.util.*;

class Pair {
    private String l;
    private String r;

    public Pair(String l,String r) {
        this.l = l;
        this.r = r;
    }

    public String getL(){ return l; }
    public String getR(){ return r; }
    public void setL(String l){ this.l = l; }
    public void setR(String r){ this.r = r; }

    @Override
    public boolean equals(Object obj) {
        if(this == obj)
            return true;
        else if(obj instanceof Pair){
            Pair other = (Pair) obj;
            if(l == other.l && r == other.r)
```

```
            return true;
        }

        return false;
    }

    @Override
    public int hashCode()
    {
        return r.hashCode() + l.hashCode();
    }
}

public class Graph {

    private HashMap<String, List<String>> vertexMap;

    public Graph()
    {
        vertexMap = new HashMap<>();
    }

    public Graph(HashMap<String, List<String>> g)
    {
        vertexMap = g;
    }

    public HashMap<String, List<String>> getGraph()
    {
        return vertexMap;
```

```
    }

    public void addVertex(String vertexName) {
        if (!hasVertex(vertexName)) {
            vertexMap.put(vertexName, new ArrayList<>());
        }
    }

    public boolean hasVertex(String vertexName) {
        return vertexMap.containsKey(vertexName);
    }

    public boolean hasEdge(String vertexName1, String vertexName2) {
        if (!hasVertex(vertexName1)) return false;
        List<String> edges = vertexMap.get(vertexName1);
        return Collections.binarySearch(edges, vertexName2) != -1;
    }

    public void addEdge(String vertexName1, String vertexName2) {
        if (!hasVertex(vertexName1)) addVertex(vertexName1);
        if (!hasVertex(vertexName2)) addVertex(vertexName2);
        List<String> edges1 = vertexMap.get(vertexName1);
        edges1.add(vertexName2);
        Collections.sort(edges1);
    }

    public Graph transposeGraph() {
        HashMap<String, List<String>> new_g = new HashMap<String,
List<String>>();
```

```
        for (String vertex : vertexMap.keySet()) {
            new_g.put(vertex, new ArrayList<>());
        }
        for (String source : new_g.keySet()) {
            for (String target : vertexMap.keySet()) {
                if (vertexMap.get(target).contains(source)) {
                    new_g.get(source).add(target);
                }
            }
        }

        return new Graph(new_g);
    }

    public List<String> getNeighbours(String label) {
        return vertexMap.get(label);
    }
}
```

- KosarajuAlgorithm.java

```
package VisualKosarajuLogic;

import java.util.*;

public class KosarajuAlgorithm {
    private Graph graph = new Graph();
    private List<String> transpositionStepTrace = new ArrayList<>();
    private List<String> strongConnectivityComponents;
    private List<String> originalStepTrace = new ArrayList<>();
```

```
public void createGraph(List<String> vertexes, List<String[]> edges) {
    for(String v : vertexes)
        graph.addVertex(v);
    for(String[] e : edges)
        graph.addEdge(e[0], e[1]);
}

public void Algorithm() {
    Graph t_graph = graph.transposeGraph();
    List<String> priority_list = tOutDepthTraversal(t_graph);
    strongConnectivityComponents = depthFirstTraversal(priority_list);
}

private List<String> tOutDepthTraversal(Graph g) {
    List<String> non_visited = new ArrayList<>(g.getGraph().keySet());
    if (non_visited.isEmpty()) {
        return new ArrayList<>();
    }

    List<String> dfsTrace = new ArrayList<>();
    Stack<String> stack = new Stack<>();
    List<String> t_out_list = new ArrayList<>();
    String[] vertexes = g.getGraph().keySet().toArray(new String[0]);
    stack.push(vertexes[0]);

    while (!non_visited.isEmpty() || !stack.empty()) {
        if(stack.isEmpty())
        {
            dfsTrace.add("");
```

```
            stack.push(non_visited.get(0));
            non_visited.remove(0);
        }

        String vertex = stack.peek();
        dfsTrace.add(vertex);
        non_visited.remove(vertex);
        int stack_size = stack.size();

        for(String u : g.getNeighbours(vertex))
            if(!dfsTrace.contains(u))
            {
                stack.push(u);
                break;
            }

        if(stack_size == stack.size()) {
            t_out_list.add(stack.pop());
        }
    }

    Collections.reverse(t_out_list);
    transpositionStepTrace.addAll(dfsTrace);
    return t_out_list;
}

private List<String> depthFirstTraversal(List<String> list_p) {
    if (list_p.isEmpty()) {
        return new ArrayList<>();
    }
```

```
List<String> non_visited = new ArrayList<String>(graph.getGraph().keySet());
Stack<String> stack = new Stack<String>();
List<String> c_c = new ArrayList<String>();
stack.push(list_p.get(0));
list_p.remove(0);

while (!non_visited.isEmpty() || !stack.empty()) {
   if (stack.isEmpty()) {
      String first = list_p.get(0);
      stack.push(first);
      non_visited.remove(first);
      originalStepTrace.add("");
      c_c.add(null);
   }

   String vertex = stack.peek();
   originalStepTrace.add(vertex);
   non_visited.remove(vertex);
   if (!c_c.contains(vertex)) {
      list_p.remove(vertex);
      c_c.add(vertex);
   }

   int stackSize = stack.size();
   for (String v : graph.getNeighbours(vertex)) {
      if (non_visited.contains(v)) {
         stack.push(v);
         break;
      }
```

```
            }
            if (stackSize == stack.size()) {
                stack.pop();
            }
        }
        return c_c;
    }

    public List<String> getTranspositionStepTrace() {
        return transpositionStepTrace;
    }

    public List<String> getStrongConnectivityComponents() {
        return strongConnectivityComponents;
    }

    public List<String> getOriginalStepTrace() {
        return originalStepTrace;
    }
}
```

- AuthorsMenu.java

```
package VisualKosarajuGUI;

import javax.swing.*;
import java.awt.*;

class AuthorsMenu extends JFrame {
    AuthorsMenu() {
```

```
        super("Visual Kosaraju - Контакты");
        getContentPane().setPreferredSize(new Dimension(330,165));
        getContentPane().setLayout(new FlowLayout(FlowLayout.LEFT));

        JLabel nameDavydov = new JLabel("1) Давыдов А.А. - тимлид, старший
разработчик");
        JLabel emailDavydov = new JLabel("Почта - casha_davydov@mail.ru");
        JLabel nameChemova = new JLabel("2) Чемова К.А. - разработчик,
тестировщик");
        JLabel emailChemova = new JLabel("Почта - chemovaks@mail.ru");
        JLabel namePet = new JLabel("3) Пэтайчук Н.Г. - разработчик, дизайнер");
        JLabel emailPet = new JLabel("Почта - pet.ai.4.uk@yandex.ru");
        JSeparator firstSeparator = new JSeparator();
        firstSeparator.setPreferredSize(new Dimension(320, 10));
        JSeparator secondSeparator = new JSeparator();
        secondSeparator.setPreferredSize(new Dimension(320, 10));

        getContentPane().add(nameDavydov);
        getContentPane().add(emailDavydov);
        getContentPane().add(firstSeparator);
        getContentPane().add(nameChemova);
        getContentPane().add(emailChemova);
        getContentPane().add(secondSeparator);
        getContentPane().add(namePet);
        getContentPane().add(emailPet);
        pack();
        setVisible(true);
    }
}
```

- HelpMenu.java

```
package VisualKosarajuGUI;

import javax.swing.*;
import java.awt.*;

class HelpMenu extends JFrame {
   HelpMenu() {
      super("Visual Kosaraju - Помощь");
      setSize(new Dimension(550, 300));
      getContentPane().setLayout(new FlowLayout());

      String contentOfTextArea = createHelpMenuText();
      JTextArea textArea = new JTextArea();
      textArea.setText(contentOfTextArea);
      textArea.setEditable(false);
      textArea.setBackground(Color.WHITE);
      textArea.setForeground(Color.BLACK);
      textArea.setFont(new Font("Arial", Font.PLAIN, 14));
      JScrollPane scrollPane = new JScrollPane(textArea);

scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR
_AS_NEEDED);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_
NEEDED);
      scrollPane.setPreferredSize(new Dimension(530, 280));
      getContentPane().add(scrollPane);
      pack();
```

```
        setVisible(true);
    }

    private String createHelpMenuText() {
        return "РИСОВАНИЕ ГРАФА:\n\n" +
                "-) Левая кнопка мыши - перемещение элементов, создание рёбер\n" +
                "   (нажатие на середину вершины, провод стрелки до нужной
вершины);\n" +
                "-) Правая кнопка мыши - удаление элемента, либо создание
вершины,\n" +
                "   если нажать на пустом месте;\n\n" +
                "ОТКРЫТИЕ ГРАФА ИЗ ФАЙЛА:\n\n" +
                "Графы должны храниться в файлах .json в следующем формате:\n" +
                "{\n" +
                "    \"Vertexes\": [list of vertexes],\n" +
                "    \"Edges\": {\n" +
                "        \"vertex_from_list\": [list of neighbor vertexes],\n" +
                "        ...\n" +
                "    }\n" +
                "}\n\n" +
                "УПРАВЛЕНИЕ ВИЗУАЛИЗАЦИЕЙ:\n\n" +
                "-) Начать визуализацию - начать пошаговую работу алгоритма над\n"
+
                "   данным графом;\n" +
                "-) Следующий шаг - переход на следующий шаг алгоритма;\n" +
                "-) Предыдущий шаг - возврат на предыдущий шаг алгоритма;\n" +
                "-) Доработать до конца - переход на конечный шаг работы
алгоритма;\n" +
                "-) Вернуться в начало - переход на начальный этап работы
алгоритма;\n" +
```

```
            "-) Сброс - прервать выполнение алгоритма/выход из режима
визуализации;\n\n" +
            "В чёрном текстовом поле будет показано краткое описание
текущего\n" +
            "шага алгоритма.";
    }
}
```

- VisualKosarajuAlgorithmProxy.java

```
package VisualKosarajuGUI;

import VisualKosarajuLogic.*;
import com.mxgraph.model.mxCell;
import com.mxgraph.view.mxGraph;

import java.util.ArrayList;
import java.util.List;

class VisualKosarajuAlgorithmProxy extends VisualKosarajuWindow {
    private List<String> vertexes = new ArrayList<>();
    private List<String[]> edges = new ArrayList<>();
    private List<String> transpositionStepTrace;
    private List<String> strongConnectivityComponents;
    private List<String> originalStepTrace;

    VisualKosarajuAlgorithmProxy() {
        KosarajuAlgorithm mainLogic = new KosarajuAlgorithm();

        prepareGraphData();
```

```
        mainLogic.createGraph(vertexes, edges);
        mainLogic.Algorithm();
        transpositionStepTrace = mainLogic.getTranspositionStepTrace();
        strongConnectivityComponents =
mainLogic.getStrongConnectivityComponents();
        originalStepTrace = mainLogic.getOriginalStepTrace();
    }

    private void prepareGraphData() {
        mxGraph graph = this.getGraph();
        graph.clearSelection();
        graph.selectAll();
        Object[] cells = graph.getSelectionCells();
        for (Object cell : cells) {
            mxCell graphElement = (mxCell) cell;
            if (graphElement.isVertex()) {
                vertexes.add(graphElement.getValue().toString());
            } else if (graphElement.isEdge()) {
                String source = graphElement.getSource().getValue().toString();
                String target = graphElement.getTarget().getValue().toString();
                edges.add(new String[]{source, target});
            }
        }
        graph.clearSelection();
    }

    String[] createAlgorithmStepTrace() {
        List<String> algorithmStepTrace = new ArrayList<>();
        algorithmStepTrace.add("");
        algorithmStepTrace.addAll(transpositionStepTrace);
```

```
        algorithmStepTrace.add("");
        algorithmStepTrace.add(null);
        algorithmStepTrace.addAll(originalStepTrace);
        algorithmStepTrace.add("");
        algorithmStepTrace.add(null);
        return algorithmStepTrace.toArray(new String[0]);
    }

    List<String> getStrongConnectivityComponents() {
        return strongConnectivityComponents;
    }
}
```

- VisualKosarajuWindow.java

```
package VisualKosarajuGUI;

import com.mxgraph.model.mxCell;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.view.mxGraph;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
```

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Optional;

class IncorrectJSONFileContentException extends Exception {
    IncorrectJSONFileContentException() {
        super("Incorrect content of JSON graph file.");
    }
}

class VisualKosarajuWindow extends JFrame {
    private static mxGraph graph  = new mxGraph();
    private HashMap<String, Object> VertexHashMap = new HashMap<>();
    private String[] algorithmStepTrace;
    private List<String> strongConnectivityComponents;
    private int indexOfAlgorithmStep;
    private int indexOfTransposition;
    private boolean isEditMode;

    private String[] colors;
    private static final String STANDARD_COLOR = "#C3D9FF";
    private static final String CHOSEN_COLOR = "#FFD9C3";

    private mxGraphComponent graphComponent;
    private JTextArea messageArea;
```

```
VisualKosarajuWindow() {
    super("Visual Kosaraju");
    isEditMode = true;
    colors = new String[]{"#88fb88", "#ffff66", "#eebef1",
                          "#ffffff", "#ffa375", "#c284e0"};
    initializeGUI();
}

static mxGraph getGraph() {
    return graph;
}

private void initializeGUI() {
    setSize(new Dimension(630, 680));
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    getContentPane().setLayout(new FlowLayout(FlowLayout.LEFT));

    createMenuBar();
    createGraphComponent();
    createStartButton();
    createNextStepButton();
    createPrevStepButton();
    createToTheEndButton();
    createToTheBeginButton();
    createResetButton();
    createMessageArea();
    createAuthorsLabel();
}
```

```
    private void createMenuBar() {
        JMenuBar menuBar = new JMenuBar();
        JMenu menu = new JMenu("Программа");
        JMenuItem openFileMenu = new JMenuItem(new openGraphJSONFile());
        JMenuItem helpMenu = new JMenuItem(new openHelpMenuAction());
        JMenuItem authorsMenu = new JMenuItem(new openAuthorsMenuAction());

        menu.add(openFileMenu);
        menu.add(new JSeparator());
        menu.add(helpMenu);
        menu.add(authorsMenu);
        menuBar.add(menu);
        getContentPane().add(menuBar);
    }

    private void createGraphComponent() {
        graphComponent = new mxGraphComponent(graph);
        graphComponent.setPreferredSize(new Dimension(600,365));
        graphComponent.getGraphControl().addMouseListener(new MouseListener() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getButton() == MouseEvent.BUTTON3) {
                    if (isEditMode) {
                        Object editingGraphElement = graphComponent.getCellAt(e.getX(),
e.getY());
                        if (editingGraphElement == null) {
                            String vertexName = JOptionPane.showInputDialog("Введите имя
вершины");
                            addVertex(vertexName, e.getX(), e.getY());
```

```
                    } else {
                        deleteVertex(editingGraphElement);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Нельзя модифицировать граф в данный момент");
                }
            }
        }

        @Override
        public void mousePressed(MouseEvent e) {}

        @Override
        public void mouseReleased(MouseEvent e) {}

        @Override
        public void mouseEntered(MouseEvent e) {}

        @Override
        public void mouseExited(MouseEvent e) {}
    });
    getContentPane().add(graphComponent);
}

private void createAuthorsLabel() {
    JLabel authorsLabel = new JLabel("Создатели: Давыдов А.А., Пэтайчук Н.Г., Чемова К.А.; " +
            "Компания \"Davydov & Co.\"");
    authorsLabel.setPreferredSize(new Dimension(600,15));
```

```
        getContentPane().add(authorsLabel);
    }

    private void createStartButton() {
        JButton StartButton = new JButton("Начать визуализацию");
        StartButton.setPreferredSize(new Dimension(196, 25));
        StartButton.addActionListener((ActionEvent e) -> {
            if (isNoGraph()) {
                JOptionPane.showMessageDialog(null, "Нет графа");
                return;
            }
            if (isEditMode) {
                isEditMode = false;
                VisualKosarajuAlgorithmProxy algorithmProxy = new
VisualKosarajuAlgorithmProxy();
                strongConnectivityComponents =
algorithmProxy.getStrongConnectivityComponents();
                algorithmStepTrace = algorithmProxy.createAlgorithmStepTrace();
                indexOfAlgorithmStep = 0;
                findIndexOfTransposition();
                graphTransposition();
                logMessage("АКТИВАЦИЯ РЕЖИМА ВИЗУАЛИЗАЦИИ\n\n");
            }
        });
        getContentPane().add(StartButton);
    }

    private void createNextStepButton() {
        JButton NextStepButton = new JButton("Следующий шаг");
        NextStepButton.setPreferredSize(new Dimension(197, 25));
```

```
        NextStepButton.addActionListener((ActionEvent e) -> {
            if (isEditMode) {
                JOptionPane.showMessageDialog(null, "Нельзя использовать эту кнопку
в Режиме Редактирования");
                return;
            } else if (indexOfAlgorithmStep == algorithmStepTrace.length - 1) {
                return;
            }
            goToNextStep();
            logAlgorithmStep(indexOfAlgorithmStep);
        });
        getContentPane().add(NextStepButton);
    }

    private void createPrevStepButton() {
        JButton PrevStepButton = new JButton("Предыдущий шаг");
        PrevStepButton.setPreferredSize(new Dimension(196, 25));
        PrevStepButton.addActionListener((ActionEvent e) -> {
            if (isEditMode)
            {
                JOptionPane.showMessageDialog(null, "Нельзя использовать эту кнопку
в Режиме Редактирования");
                return;
            } else if (indexOfAlgorithmStep == 0) {
                return;
            }
            goToPrevStep();
            logAlgorithmStep(indexOfAlgorithmStep);
        });
        getContentPane().add(PrevStepButton);
```

```
    }

    private void createToTheEndButton() {
        JButton ToTheEndButton = new JButton("Доработать до конца");
        ToTheEndButton.setPreferredSize(new Dimension(196, 25));
        ToTheEndButton.addActionListener((ActionEvent e) -> {
            if (isEditMode)
            {
                JOptionPane.showMessageDialog(null, "Нельзя использовать эту кнопку
в Режиме Редактирования");
                return;
            }
            if (indexOfAlgorithmStep < indexOfTransposition) {
                graphTransposition();
            }
            colorConnectivityComponents();
            indexOfAlgorithmStep = algorithmStepTrace.length - 1;
            logAlgorithmStep(indexOfAlgorithmStep);
        });
        getContentPane().add(ToTheEndButton);
    }

    private void createToTheBeginButton() {
        JButton ToTheBeginButton = new JButton("Вернуться в начало");
        ToTheBeginButton.setPreferredSize(new Dimension(197, 25));
        ToTheBeginButton.addActionListener((ActionEvent e) -> {
            if (isEditMode)
            {
                JOptionPane.showMessageDialog(null, "Нельзя использовать эту кнопку
в Режиме Редактирования");
```

```
            return;
        }
        if (indexOfAlgorithmStep >= indexOfTransposition) {
            graphTransposition();
        }
        discolorVertexes();
        indexOfAlgorithmStep = 0;
        logAlgorithmStep(indexOfAlgorithmStep);
    });
    getContentPane().add(ToTheBeginButton);
}

private void createResetButton() {
    JButton ResetButton = new JButton("Сброс");
    ResetButton.setPreferredSize(new Dimension(196, 25));
    ResetButton.addActionListener((ActionEvent e) -> {
        if (!isEditMode) {
            isEditMode = true;
            discolorVertexes();
            if (indexOfAlgorithmStep < indexOfTransposition) {
                graphTransposition();
            }
            logMessage("\nВЫХОД ИЗ РЕЖИМА ВИЗУАЛИЗАЦИИ\n\n");
        }
    });
    getContentPane().add(ResetButton);
}

private void createMessageArea() {
    messageArea = new JTextArea();
```

```
        messageArea.setEditable(false);
        messageArea.setBackground(new Color(0x000000));
        messageArea.setForeground(new Color(0xFFFFFF));
        messageArea.setFont(new Font("CourierNew", Font.PLAIN, 14));
        JScrollPane scrollPane = new JScrollPane(messageArea);

scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR
_AS_NEEDED);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_
NEEDED);
        scrollPane.setPreferredSize(new Dimension(600, 150));
        getContentPane().add(scrollPane);
    }

    private boolean isNoGraph() {
        graph.clearSelection();
        graph.selectAll();
        Object[] cells = graph.getSelectionCells();
        graph.clearSelection();
        return cells.length == 0;
    }

    private void addVertex(String vertexName, int x, int y) {
        if (vertexName != null) {
            if (!vertexName.equals("")) {
                if (!VertexHashMap.containsKey(vertexName)) {
                    graph.getModel().beginUpdate();
                    Object parent = graph.getDefaultParent();
```

```
                Object vertex = graph.insertVertex(parent, null, vertexName, x, y, 50, 50,
"shape=ellipse");
                VertexHashMap.put(vertexName, vertex);
                graph.getModel().endUpdate();
            } else {
                JOptionPane.showMessageDialog(null, "Вершина с таким именем уже
существует");
            }
        } else {
            JOptionPane.showMessageDialog(null, "Нельзя создать вершину без
имени");
        }
    }

    private void addEdge(String sourceVertex, String targetVertex) {
        Object parent = graph.getDefaultParent();
        Object source = VertexHashMap.get(sourceVertex);
        Object target = VertexHashMap.get(targetVertex);
        graph.insertEdge(parent, null, "", source, target);
    }

    private void deleteGraph() {
        String[] vertexes = VertexHashMap.keySet().toArray(new String[0]);
        for (String vertex : vertexes) {
            deleteVertex(VertexHashMap.get(vertex));
        }
    }

    private void deleteVertex(Object vertexToDelete) {
```

```
        String nameOfDeletedVertex = ((mxCell) vertexToDelete).getValue().toString();
        VertexHashMap.remove(nameOfDeletedVertex);
        graph.getModel().remove(vertexToDelete);
        deleteUnusedEdges();
    }

    private void changeColor(Object graphElement, String color) {
        if (graphElement != null) {
            String style = "fillColor=" + color + ";shape=ellipse";
            graph.getModel().beginUpdate();
            Object[] cellArray = new Object[1];
            cellArray[0] = graphElement;
            graph.setCellStyle(style, cellArray);
            graph.getModel().endUpdate();
        }
    }

    private void deleteUnusedEdges() {
        graph.clearSelection();
        graph.selectAll();
        Object[] cells = graph.getSelectionCells();
        for (Object cell : cells) {
            mxCell graphComponent = (mxCell) cell;
            if (graphComponent.isEdge()) {
                String source = (String) graphComponent.getSource().getValue();
                String target = (String) graphComponent.getTarget().getValue();
                if (!(VertexHashMap.containsKey(source) &&
VertexHashMap.containsKey(target))) {
                    graph.getModel().remove(graphComponent);
                }
```

```
        }
    }
    graph.clearSelection();
}

private void graphTransposition() {
    graph.clearSelection();
    graph.selectAll();
    Object[] cells = graph.getSelectionCells();
    for (Object cell : cells) {
        mxCell graphComponent = (mxCell) cell;
        if (graphComponent.isEdge()) {
            Object source = graphComponent.getSource();
            Object target = graphComponent.getTarget();
            Object parent = graph.getDefaultParent();
            graph.getModel().remove(graphComponent);
            graph.insertEdge(parent, null, "", target, source);
        }
    }
    graph.clearSelection();
}

private void colorConnectivityComponents() {
    int colorIndex = 0;
    for (String elem : strongConnectivityComponents) {
        if (elem == null) {
            colorIndex++;
            colorIndex %= 6;
        } else {
            changeColor(VertexHashMap.get(elem), colors[colorIndex]);
```

```
        }
    }
}

private void discolorVertexes() {
    graph.clearSelection();
    graph.selectAll();
    Object[] cells = graph.getSelectionCells();
    for (Object cell : cells) {
        mxCell graphComponent = (mxCell) cell;
        if (graphComponent.isVertex()) {
            changeColor(cell, "#C3D9FF");
        }
    }
    graph.clearSelection();
}

private void findIndexOfTransposition() {
    for (int i = 0; i < algorithmStepTrace.length; i++) {
        if (algorithmStepTrace[i] == null) {
            indexOfTransposition = i;
            break;
        }
    }
}

private void goToNextStep() {
    String nowVertex = algorithmStepTrace[indexOfAlgorithmStep];
    if (nowVertex != null && !nowVertex.equals("")) {
        changeColor(VertexHashMap.get(nowVertex), STANDARD_COLOR);
```

```
    }
    indexOfAlgorithmStep++;
    String nextVertex = algorithmStepTrace[indexOfAlgorithmStep];
    if (indexOfAlgorithmStep == algorithmStepTrace.length - 1) {
        colorConnectivityComponents();
    } else if (nextVertex == null) {
        graphTransposition();
    } else if (!nextVertex.equals("")) {
        changeColor(VertexHashMap.get(nextVertex), CHOSEN_COLOR);
    }
}

private void goToPrevStep() {
    String nowVertex = algorithmStepTrace[indexOfAlgorithmStep];
    if (indexOfAlgorithmStep == algorithmStepTrace.length - 1) {
        discolorVertexes();
    } else if (nowVertex == null) {
        graphTransposition();
    } else if (!nowVertex.equals("")) {
        changeColor(VertexHashMap.get(nowVertex), STANDARD_COLOR);
    }
    indexOfAlgorithmStep--;
    String prevVertex = algorithmStepTrace[indexOfAlgorithmStep];
    if (prevVertex != null && !prevVertex.equals("")) {
        changeColor(VertexHashMap.get(prevVertex), CHOSEN_COLOR);
    }
}

private void logAlgorithmStep(int indexOfAlgorithmStep) {
    if (indexOfAlgorithmStep == algorithmStepTrace.length - 1) {
```

```
            String strongConnectivityComponentsReport =
createStrongConnectivityComponentsReport();
            logMessage("Окончание работы алгоритма. Компоненты сильной
связности:\n");
            logMessage(strongConnectivityComponentsReport);
        } else if (indexOfAlgorithmStep == indexOfTransposition) {
            logMessage("Транспонирование графа.\n");
        } else if (indexOfAlgorithmStep == 0) {
            logMessage("Начало работы алгоритма.\n");
        } else if (algorithmStepTrace[indexOfAlgorithmStep].equals("")) {
            logMessage("Выход из ветки DFS.\n");
        } else {
            String nameOfVertex = algorithmStepTrace[indexOfAlgorithmStep];
            logMessage("Переход в вершину " + nameOfVertex + ".\n");
        }
        logMessage("- - - - - - - - - - - - - - -\n");
    }

    private void logMessage(String message) {
        String nowLogText = messageArea.getText() + message;
        messageArea.setText(nowLogText);
    }

    private String createStrongConnectivityComponentsReport() {
        StringBuilder report = new StringBuilder("{");
        for (String elem : strongConnectivityComponents) {
            if (elem == null) {
                String modifiedReport = report.subSequence(0, report.length() -
2).toString();
                report = new StringBuilder(modifiedReport + "}\n{");
```

```
            continue;
        }
        report.append(elem);
        report.append(", ");
    }
    String modifiedReport = report.subSequence(0, report.length() - 2).toString();
    report = new StringBuilder(modifiedReport + "}\n");
    return report.toString();
}

private class openHelpMenuAction extends AbstractAction {
    private static final long serialVersionUID = 1L;

    openHelpMenuAction() {
        putValue(NAME, "Помощь");
    }

    public void actionPerformed(ActionEvent e) {
        new HelpMenu();
    }
}

private class openAuthorsMenuAction extends AbstractAction {
    private static final long serialVersionUID = 1L;

    openAuthorsMenuAction() {
        putValue(NAME, "Контакты");
    }

    public void actionPerformed(ActionEvent e) {
```

```
            new AuthorsMenu();
        }
    }

    private class openGraphJSONFile extends AbstractAction {
        private static final long serialVersionUID = 1L;

        openGraphJSONFile() {
            putValue(NAME, "Открыть файл с графом");
        }

        public void actionPerformed(ActionEvent e) {
            if (isEditMode) {
                JFileChooser fileChooser = new JFileChooser();
                int returnCode = fileChooser.showDialog(null, "Открыть файл с графом");
                if (returnCode == JFileChooser.APPROVE_OPTION) {
                    File file = fileChooser.getSelectedFile();
                    if (!file.getName().endsWith(".json")) {
                        JOptionPane.showMessageDialog(null, "Файл не формата .json");
                        return;
                    }
                    try {
                        String fileContent = new
String(Files.readAllBytes(Paths.get(file.getAbsolutePath())));
                        deleteGraph();
                        parseJSONFile(fileContent);
                    } catch (IOException ex) {
                        JOptionPane.showMessageDialog(null, "Возникли проблемы с
чтением файла");
                    } catch (ParseException ex) {
```

```
                    JOptionPane.showMessageDialog(null, "Некорректный формат json
файла");
                } catch (IncorrectJSONFileContentException ex) {
                    JOptionPane.showMessageDialog(null, "Некорректное содержимое
json файла");
                }
            }
        } else {
            JOptionPane.showMessageDialog(null, "Нельзя открывать файл во
время виртуализации");
        }
    }

    private void parseJSONFile(String JSONString) throws ParseException,
IncorrectJSONFileContentException {
        JSONParser parser = new JSONParser();
        JSONObject mainJSON = (JSONObject) parser.parse(JSONString);
        JSONArray vertexes = (JSONArray)
Optional.ofNullable(mainJSON.get("Vertexes")).orElse(null);
        if (vertexes == null) {
            throw new IncorrectJSONFileContentException();
        }
        createVertexes(vertexes);
        JSONObject edges = (JSONObject)
Optional.ofNullable(mainJSON.get("Edges")).orElse(null);
        if (edges == null) {
            throw new IncorrectJSONFileContentException();
        }
        createEdges(edges);
    }
```

```
    private void createVertexes(JSONArray vertexes) {
        int horizontalCoordinate = 100;
        Iterator<String> iterator = vertexes.iterator();
        for (int verticalCoordinateCoefficient = 0; iterator.hasNext();
verticalCoordinateCoefficient++) {
            String vertex = iterator.next();
            int verticalCoordinate = 80 + (verticalCoordinateCoefficient / 3) * 100;
            addVertex(vertex, horizontalCoordinate, verticalCoordinate);
            horizontalCoordinate += 200;
            horizontalCoordinate %= 600;
        }
    }

    private void createEdges(JSONObject edges) throws
IncorrectJSONFileContentException{
        String[] vertexes = VertexHashMap.keySet().toArray(new String[0]);
        for (String source : vertexes) {
            JSONArray neighbors = (JSONArray)
Optional.ofNullable(edges.get(source)).orElse(null);
            if (neighbors != null) {
                for (String target : (Iterable<String>) neighbors) {
                    if (VertexHashMap.containsKey(target)) {
                        addEdge(source, target);
                    } else {
                        throw new IncorrectJSONFileContentException();
                    }
                }
            }
        }
```

```
        }
    }
}
```

- VisualKosarajuApplication.java

```
package VisualKosarajuGUI;

public class VisualKosarajuApplication {
    private VisualKosarajuApplication() {
        VisualKosarajuWindow window = new VisualKosarajuWindow();
        window.setVisible(true);
    }

    public static void main(String[] args) {
        new VisualKosarajuApplication();
    }
}
```