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

ОТЧЁТ
по учебной практике
Тема: Мосты графа

Студент гр. 9303 _____ Ахримов А.М.

Студент гр. 9303 _____ Эйсвальд М.И.

Студент гр. 9303 _____ Максимов Е.А.

Руководитель _____ Фирсов М.А.

Санкт-Петербург
2021

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Ахримов А.М. группы 9303

Студент Эйсвальд М.И. группы 9303

Студент Максимов Е.А. группы 9303

Тема практики: Мосты графа

Задание на практику: Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом. Алгоритм: нахождение мостов графа модификацией поиска в глубину.

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчёта: 12.07.2021

Дата защиты отчёта: 12.07.2021

Студент	_____	Ахримов А.М.
Студент	_____	Эйсвальд М.И.
Студент	_____	Максимов Е.А.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

Задачей учебной практики является получение опыта командной работы над проектом. Практика заключается в разработке приложения — визуализатора указанного в задании алгоритма — на языке Java. Разработка ведётся итеративно: студентами и их руководителем согласовывается спецификация, чётко определяющая предполагаемое поведение программы, после чего создаются несколько версий проекта, каждая из которых расширяет функционал предыдущей. В отчёте приведена информация о ходе выполнения практической работы.

SUMMARY

The aim of the practice is to gain group work experience. The practice work consists of Java application development; the application must visualize an algorithm specified in the given task. Students and their mentor discuss and form project specification, which distinctly defines expected program behavior. Development should be iterative: several consecutive project versions are created; each of these extends the functionality of the previous one. This report contains information on the working process.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является разработка приложения, визуализирующего конкретный алгоритм, на языке программирования Java. Для достижения цели необходимо решить следующие задачи:

1. Сформулировать и согласовать с руководителем спецификацию, описывающую поведение программы;
2. Распределить роли в разработке приложения между членами бригады и составить план разработки;
3. Изучить алгоритм, который требуется визуализировать;
4. Написать и отладить модули проекта;
5. Скомпоновать написанные программные модули в общий проект;
6. Отладить приложение;
7. Представить выполненную работу руководителю.

В данной работе визуализируется алгоритм Тарьяна поиска мостов в графе. Алгоритм Тарьяна имеет линейную временну́ю сложность относительно количества вершин и рёбер графа и основывается на поиске в глубину. Алгоритм Тарьяна, как и другие алгоритмы поиска мостов, используется для выявления уязвимых мест (точек отказа) распределённой системы или сети.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе.

1.1.1. Требования ко входным данным.

Граф, мосты которого требуется найти, может быть предоставлен программе тремя способами:

1. Задан пользователем при помощи графического интерфейса: нажатие кнопки `Draw Node` (см. рисунок 1) и последующий клик мыши по рабочей области создают в указанном месте вершину графа; нажатие кнопки `Draw Edge` и последующие клики по двум различным вершинам создают ребро между этими вершинами графа. Попытка создать уже существующее ребро повторно не имеет эффекта. Нажатие кнопки `Erase` и клик мыши по компоненте (ребру или вершине графа) удаляют эту компоненту. Вершина удаляется со всеми инцидентными ей рёбрами.

2. Задан пользователем в виде текстового файла, содержащего список рёбер графа. Каждая строка файла должна иметь вид `<ключ 1> <ключ 2>`, где `<ключ 1>` и `<ключ 2>` — различные символы-ключи вершин графа. Попытка описать уже существующее ребро повторно не имеет эффекта. При задании графа таким способом расположение узлов в рамках рабочей области определяется автоматически.

3. Загружен из файла сохранения. Файл сохранения — файл, содержащий заголовок известного программе формата, после которого следуют: число — количество строк, описывающих вершины графа; указанное количество строчек, каждая из которых содержит ключ вершины (см. предыдущий пункт) и два числа — координаты вершины на рабочей области окна программы; список рёбер в том же формате, что и в предыдущем пункте.

В случае некорректных входных данных — например, нарушении формата файла или попытке описать петлю (ребро, соединяющее вершину графа саму с собой), — программа выдаёт информативное сообщение об ошибке, а не завершает работу.

1.1.2. Требования к визуализации процесса выполнения алгоритма.

Визуализируемый алгоритм — алгоритм Тарьяна поиска мостов в графе. Режим визуализации включается нажатием кнопки `Start` (см. рисунок 1). При этом кнопки добавления вершины и ребра в граф становятся неактивными, что делает невозможным изменение графа во время прогонки алгоритма. Кнопка `Stop` прекращает прогонку алгоритма и возвращает приложение в обычный режим работы. Визуализация алгоритма производится пошагово. Переход к визуализации следующего шага алгоритма осуществляется нажатием кнопки `Next`. Алгоритм содержит несколько этапов, каждый из которых будет визуализирован:

1. Первый этап — поиск в глубину и нумерация вершин. Вершина, из которой на данном шаге производится поиск, покрашена в яркий цвет; уже посещённые вершины покрашены в цвет, отличающийся от цвета ещё не посещённых вершин. Номера посещённых вершин отображаются рядом с вершинами. Пройденные рёбра отображаются как ориентированные. Количество потомков вершины отображается рядом с вершиной, когда у неё не остаётся непосещённых соседей. Текстовое описание текущего шага (какое ребро выбрано для продолжения поиска, полученные на этом шаге численные значения) печатается в поле текстового вывода.

2. Второй этап — вычисление дополнительных коэффициентов (они обозначены в статье как $L(v)$ и $H(v)$). На каждом шаге этого этапа обрабатываемая вершина выделяется цветом, и рядом с ней отображаются вычисленные значения. В поле текстового вывода печатается краткое обоснование того, как был получен очередной коэффициент.

3. Третий этап — проверка рёбер. На каждом шаге данного этапа проверяемое ребро подсвечивается одним из двух цветов в зависимости от того, является ли оно мостом. В поле текстового вывода печатается, на основании чего алгоритмом было принято такое решение.

1.1.3. Требования к пользовательскому интерфейсу.

1. Приложение должно иметь графический оконный интерфейс.

2. Способы взаимодействия пользователя с графом посредством графического интерфейса описаны в разделе «Требования ко входным данным».

3. Загрузка графа из файла сохранения осуществляется путём нажатия кнопки `Open File` (см. рисунок 1) и выбора нужного файла в возникшем диалоге.

4. Сохранение графа в файл осуществляется путём нажатия кнопки `Save File` и ввода имени файла в возникшем диалоге.

Эскиз пользовательского интерфейса представлен на рисунке 1.

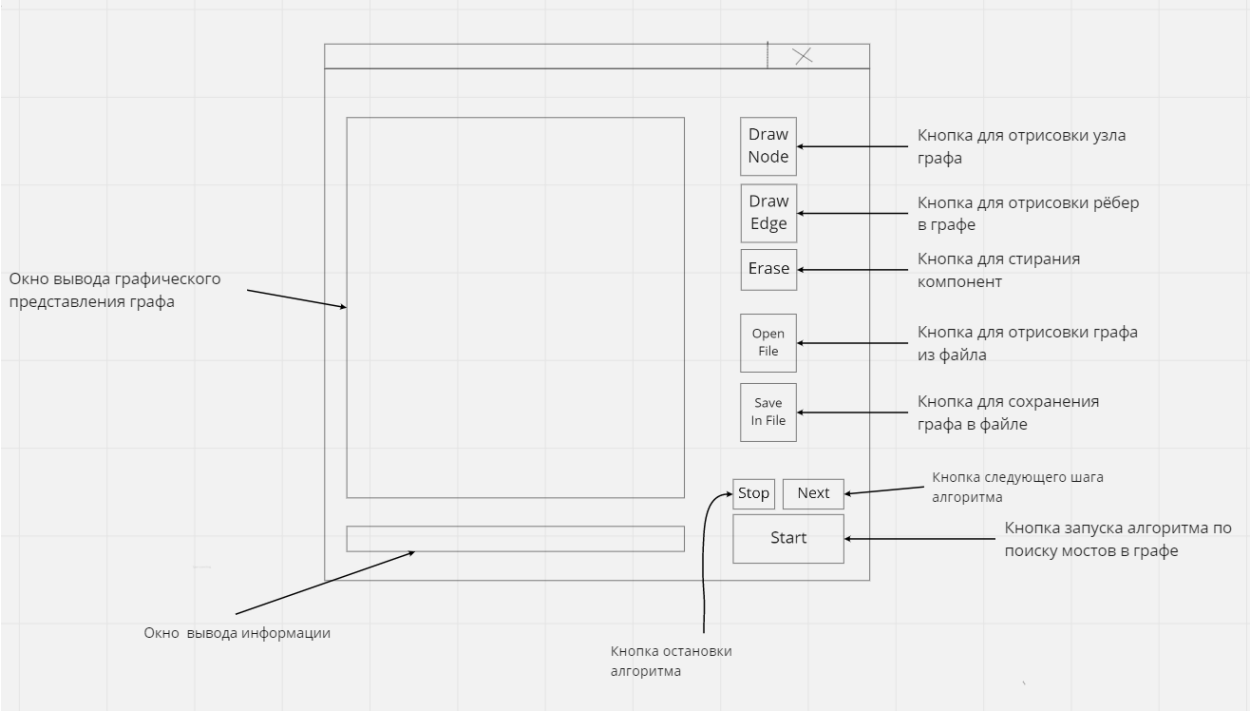

Рисунок 1 – Эскиз графического интерфейса программы

1.2. Уточнения требований после сдачи прототипа.

1. Поле вывода текстовой информации должно быть увеличено, чтобы на нём было видно несколько строк; текст в нём должен быть выделяемым и копируемым.

2. Кнопка `Next` должна быть справа от `Stop`, как в эскизе интерфейса.

3. В заголовок окна должно быть добавлено название алгоритма.

1.3. Уточнения требований после сдачи первой версии.

1. Реализовать функциональность удаления рёбер;
2. Заменить кнопки Draw Node, Draw Edge и Erase на радиокнопки, чтобы не нужно было нажимать кнопку, пока выполняется одно и то же действие редактирования графа;
3. Имена вершин выводить внутри вершин более крупным шрифтом.

1.4. Уточнения требований после сдачи второй версии.

1. Немного уменьшить диаметр вершин;
2. Помимо выделения мостов, нужно перечислить их в текстовом виде;
3. Добавить возможность редактировать текст в окне вывода, если алгоритм ещё не начат или уже завершён.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки.

1. К 5 июля — прототип приложения, демонстрирующий внешний вид интерфейса, а также графическое отображение вершин и рёбер графа.

2. К 7 июля — первая версия. Ожидаемые изменения по сравнению с прототипом: считывание графа из файла, возможность запуска алгоритма и получения результата в текстовом (не графическом) виде.

3. К 9 июля — вторая версия. Ожидаемые изменения по сравнению с первой версией: визуализация результатов работы (подсветка мостов), подсветка компоненты графа, с которой происходит работа, вывод текстовых пояснений.

4. К 12 июля — отчёт по результатам практики и третья версия приложения. Ожидаемые изменения по сравнению со второй версией: печать числовых характеристик вершин в графическом поле и отрисовка ориентированных ребёр на первом этапе работы алгоритма (см. требования к визуализации алгоритма), вывод информативных сообщений об ошибках. Дополнение от 7 июля: в финальной версии также должна быть реализована работа с файлами сохранения специального формата (см. требования ко входным данным): сохранение и загрузка.

2.2. Распределение ролей в бригаде.

- Ахримов А.М. — разработка графического интерфейса пользователя и графического представления графа;

- Эйсвальд М.И. — связь графических компонент и внутренней логики приложения; управление визуализацией алгоритма.

- Максимов Е.А. — разработка внутренней логики приложения и реализация алгоритма Тарьяна поиска мостов в графе;

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных.

Используемые в проекте структуры данных можно условно разделить на две группы: структуры, содержащие реализацию алгоритма, и структуры, отвечающие за визуализацию и общение программы со внешней средой. Классы, реализующие алгоритм Тарьяна, описаны ниже.

- `Node` — класс вершины графа. Содержит приватные поля для хранения имени, вычисленных в рамках алгоритма значений, инцидентных рёбер (в том числе ребра, ведущего в родительскую вершину остовного дерева) и координат на графическом поле, а также публичные методы для взаимодействия с содержимым полей.

- `Edge` — класс ребра графа. Хранит ссылки на инцидентные вершины в качестве приватных полей и флаги, сигнализирующие об ориентации ребра и о том, является ли ребро мостом — в качестве публичных. Класс также содержит публичные методы для взаимодействия с содержимым.

- `Graph` — класс, представляющий граф. Данные о графе представлены в виде списков вершин и рёбер. Для упрощения визуализации отдельно хранятся список мостов и список объектов — вершин и рёбер — в порядке их обхода. В последнем каждый объект встречается дважды: первое вхождение соответствует моменту входа в объект, второе — моменту выхода. Публичные методы графа позволяют получать и модифицировать его содержимое, в том числе запускать алгоритм Тарьяна и получать результат.

Классы, реализующие визуализацию алгоритма и взаимодействие со внешней средой, описаны в списке ниже.

- Классы `NodeImage` и `EdgeImage` представляют собой изображения компонент графа. Для отрисовки был переопределен метод рисования компоненты `paintComponent` класса `JComponent`, от которого унаследованы классы изображений.

- Класс `PaintArea` представляет собой «холст» для отрисовки графа, наследуется от `JLayeredPane`. Содержит функционал обработки кликов мышью (нажатия на «холст» приводят к различным изменениям графа) и отрисовки графа.

- Класс `MainWindow` представляет собой главное окно. Этот класс унаследован от JFrame. `MainWindow` — главный класс приложения, содержащий метод `main()`. Все элементы графического интерфейса, кроме изображений компонент графа, находятся в главном окне. Помимо элементов интерфейса, класс содержит обработчики нажатия на эти элементы.

- `FileReaderWrapper` и `FileWriterWrapper` — классы, обеспечивающие сохранение и загрузку графов из файлов специального формата. Экземпляры этих классов создаются в обработчиках нажатия на кнопки сохранения в файл и открытия файла. Классы скрывают процесс выбора, открытия и чтения (записи) файла от главного класса, который занимается только интерпретацией результатов.

О соотношениях между классами программы можно узнать из UML-диаграммы. UML-диаграмма классов проекта представлена в приложении А.

3.2. Основные методы.

Ниже описаны важнейшие методы разных классов проекта.

- Методы `addNode`, `removeNode`, `addEdge`, `removeEdge` класса `Graph` производят базовые манипуляции с графом, необходимые для удобной работы;

- Метод `void runAlgorithm()` класса `Graph` запускает работу алгоритма Тарьяна, предварительно сбрасывая все имеющиеся значения в объектах, которые образуют граф. Для каждой из компонент связности графа вызывается приватный метод, в основе которого лежит поиск в глубину. После обхода всех вершин в соответствующее свойство класса записываются все рёбра, которые являются мостами;

- `void drawNode(Node nd)`,`void drawNode(Node nd, Color color)`,`void drawNode(Node nd, Color color, String[] text)`,`void drawEdge(Edge edge)`,`void drawEdge(Edge edge, Color color)`,`void drawArrow(Edge edge)`,`void drawGraph()`,`void clear()` — методы управления графическим отображением графа на «холсте», рисующие отдельные элементы или граф целиком. Последний метод очищает «холст».

- Методы `ArrayList<ArrayList<String>> read()` и `boolean write(Graph graph)` соответственно классов `FileReaderWrapper` и `FileWriterWrapper` обеспечивают удобное чтение и запись файлов сохранения.

- Обработчики кликов мыши по элементам GUI обеспечивают взаимодействие между программой и пользователем.

Полный код программы представлен в приложении Б.

4. ТЕСТИРОВАНИЕ

Результаты тестирования представлены на скриншотах ниже.

4.1. Редактирование графа с помощью графического интерфейса.

На рисунках 3–8 в приложении В представлена демонстрация тестирования функционала модификации графа с помощью графического интерфейса. Были проверены все описанные в спецификации действия по модификации графа с помощью графического интерфейса. Для удобства восприятия большая часть рисунков выполнена в форме коллажей из скриншотов, отображающих последовательные состояния программы.

4.2. Визуализация алгоритма.

На рисунках 9–11 в приложении В представлена демонстрация тестирования визуализации алгоритма на различных графах.

Текстовый вывод программы после запуска на примитивном графе (см. рис. 10) представлен в листинге ниже.

```
Starting algorithm... Click 'Next' to explore steps.
====
Reached Node A
Choosing edge A -> B to move on.
Reached Node B
No more ways from Node B
Now coefficients cat be calculated:
Node number (entry time) N: 2
Number of nodes in subtree, including this node D: 1
L = min(N - D + 1; {L(μ) | B -> μ}; {N(μ) | B <- μ}): 2
H = max(N; {H(μ) | B -> μ}; {N(μ) | B <- μ}): 2
Returning to A
No more ways from Node A
Now coefficients cat be calculated:
Node number (entry time) N: 1
Number of nodes in subtree, including this node D: 2
L = min(N - D + 1; {L(μ) | A -> μ}; {N(μ) | A <- μ}): 1
H = max(N; {H(μ) | A -> μ}; {N(μ) | A <- μ}): 2
Depth-first search has finished.
Edge μ -> ν is a bridge if and only if
H(ν) ≤ N(ν) and L(ν) > N(ν) - D(ν)
Bridges are highlighted:
A -> B
```

4.3. Загрузка и сохранение графа.

На рисунках 12–14 в приложении В представлена демонстрация тестирования сохранения и загрузки графа. Загрузка производилась в том числе из повреждённых файлов. Не любое повреждение файла приводит к фатальному сбою создания графа: в некоторых случаях граф создаётся, но в текстовое поле выводится информация о произошедших во время чтения ошибках.

ЗАКЛЮЧЕНИЕ

В ходе учебной практики был изучен строго типизированный объектно-ориентированный язык программирования Java.

Перед выполнением основной задачи учебной практики был изучен алгоритм Тарьяна для поиска мостов в графе. Главным преимуществом алгоритма Тарьяна является линейная временная сложность. Недостатком алгоритма является обход графа в глубину и наличие рекурсивных вызовов методов.

В процессе выполнения учебной практики была рассмотрена библиотека Swing языка программирования Java для реализации графического интерфейса.

Результатом практической работы бригады является приложение, реализованное на языке программирования Java, которое визуализирует рассматриваемый алгоритм для поиска мостов. Приложение позволяет пользователю ввести граф как через интерфейс приложения, так и в виде файла с расширением .bfsv. Кнопки интерфейса обеспечивают пошаговую визуализацию рассматриваемого алгоритма. В результате тестирования были выявлены неточности в работе приложения, которые были успешно исправлены.

Полученный результат соответствует поставленным целям учебной практики. Функционал приложения соответствует согласованной спецификации с учётом замечаний преподавателя.

СПИСОК ЛИТЕРАТУРЫ

[1] R. Endre Tarjan. ”A note on finding the bridges of graph”. // Information Processing Letters. 1974, вып. №2. С. 160–161.

[2] Рыбин С. В. Дискретная математика и информатика: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2021. 260-263с.

[3] Всё о Java и SQL // Java Online. URL: http://java-online.ru (дата обращения: 04.07.2021).

[4] Java™ Platform, Standard Edition 7 API Specification. // Oracle Docs URL: https://docs.oracle.com/javase/7/docs/api/ (дата обращения: 04.07.2021).

[5] Java. Базовый курс // Образовательная платформа Stepik. URL: https://stepik.org/course/187/ (дата обращения: 02.07.2021)

ПРИЛОЖЕНИЕ А
UML-ДИАГРАММА КЛАССОВ

UML-диаграмма классов проекта представлена на рисунке ниже.

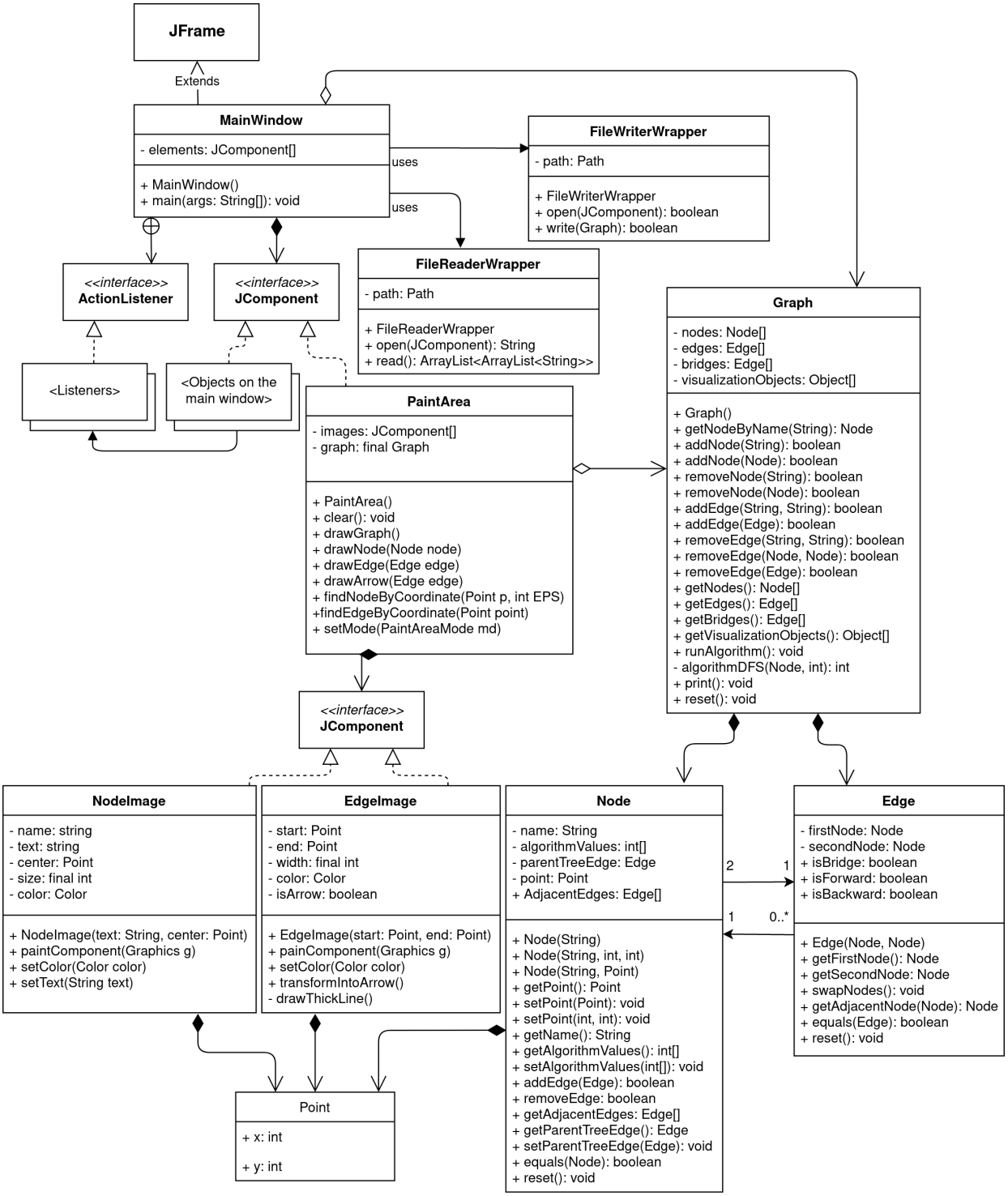

Рисунок 2 – UML-диаграмма классов

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Node.java

```
package algorithm;

import java.util.ArrayList;

public class Node {
private String name = "";
private int[] algorithmValues = new int[4];
private Edge parentTreeEdge = null;
private Point point;
private ArrayList<Edge> adjacentEdges = new ArrayList<>();

public Node(String name){
this.name = name;
this.point = new Point(0, 0);
}

public Node(String name, int x, int y){
this.name = name;
this.point = new Point(x, y);
}

public Node(String name, Point newPoint){
this.name = name;
this.point = newPoint;
}

public Point getPoint(){ return point; }

public void setPoint(Point newPoint){ setPoint(newPoint.x, newPoint.y); }

public void setPoint(int x, int y){
point.x = x;
point.y = y;
}

public String getName(){ return name; }

public int[] getAlgorithmValues(){ return algorithmValues; }

public void setAlgorithmValues(int[] newValues){
for(int i = 0; i < algorithmValues.length; i++ )
algorithmValues[i] = newValues[i];
}

public boolean addEdge(Edge newEdge){
for(int i = 0; i < adjacentEdges.size(); i++)
if(adjacentEdges.get(i).equals(newEdge))
return false;
return adjacentEdges.add(newEdge);
}

public boolean removeEdge(Edge removedEdge){
int tempIndex = adjacentEdges.indexOf(removedEdge);
if(tempIndex == -1)
```

```
return false;
else {
adjacentEdges.remove(tempIndex);
return true;
}
}

public ArrayList<Edge> getAdjacentEdges(){ return adjacentEdges; }

public Edge getParentTreeEdge(){ return parentTreeEdge; }

public void setParentTreeEdge(Edge parentTreeEdge){
this.parentTreeEdge = parentTreeEdge;
}

public boolean equals(Node anotherNode){
return this.getName().equals(anotherNode.getName());
}

public void reset(){
algorithmValues = new int[4];
parentTreeEdge = null;
}

@Override
public String toString() {
return this.name;
}
}
```

Название файла: Edge.java

```
package algorithm;

public class Edge {
private Node firstNode = null, secondNode = null;
public boolean isBridge = false;
public boolean isForward = false;
public boolean isBackward = false;

public Edge(Node firstNode, Node secondNode){
this.firstNode = firstNode;
this.secondNode = secondNode;
}

public Node getFirstNode(){ return firstNode; }

public Node getSecondNode(){ return secondNode; }

public void swapNodes(){
Node tempNode = firstNode;
firstNode = secondNode;
secondNode = tempNode;
}

public Node getAdjacentNode(Node node){
return (firstNode == node) ? secondNode : firstNode;
}

public boolean equals(Edge anotherEdge){
```

```
boolean isEqualForward =
((this.firstNode == anotherEdge.firstNode) &&
(this.secondNode == anotherEdge.secondNode));
boolean isEqualBackward =
((this.firstNode == anotherEdge.secondNode) &&
(this.secondNode == anotherEdge.firstNode));
return (isEqualForward || isEqualBackward);
}

public void reset(){
isBridge = false;
isForward = false;
isBackward = false;
}

@Override
public String toString() {
String delimiter;
delimiter = isForward ? ” -> ” : ” <- ”;
return this.getFirstNode().getName() + delimiter + this.getSecondNode().getName();
}
}
```

Название файла: Point.java

```
package algorithm;

public class Point {
public Point(int x, int y){
this.x = x;
this.y = y;
}

public int x;
public int y;
}
```

Название файла: Graph.java

```
package algorithm;

import java.util.ArrayList;

public class Graph {
private ArrayList<Node> nodes = new ArrayList<>();
private ArrayList<Edge> edges = new ArrayList<>();
private ArrayList<Edge> bridges = new ArrayList<>();
private ArrayList<Object> visualizationObjects = new ArrayList<>();

public Graph(){}

public Node getNodeByName(String searchNodeName){
for(int i = 0; i < nodes.size(); i++)
if(nodes.get(i).getName().equals(searchNodeName))
return nodes.get(i);
return null;
}

public boolean addNode(String newNodeName){
```

```
return addNode(new Node(newNodeName));
}

public boolean addNode(Node newNode){
for(int i = 0; i < nodes.size(); i++)
if(nodes.get(i).equals(newNode))
return false;
return nodes.add(newNode);
}

public boolean removeNode(String nodeName){
return removeNode(getNodeByName(nodeName));
}

public boolean removeNode(Node removedNode){
if(removedNode == null)
return false;
Edge tempEdge;
Node tempNode;
ArrayList<Edge> tempAdjacentEdges;
int tempIndex = nodes.indexOf(removedNode);
if(tempIndex == -1)
return false;
nodes.remove(tempIndex);
for(int i = 0; i < edges.size(); i++){
tempEdge = edges.get(i);
if((tempEdge.getFirstNode() == removedNode)||
(tempEdge.getSecondNode() == removedNode)){
edges.remove(i);
tempNode = (tempEdge.getFirstNode() == removedNode) ?
tempEdge.getSecondNode() : tempEdge.getFirstNode();
tempAdjacentEdges = tempNode.getAdjacentEdges();
tempAdjacentEdges.remove(tempAdjacentEdges.indexOf(tempEdge));
i--;
}
}
return true;
}

public boolean addEdge(String newFirstNodeName, String newSecondNodeName){
Node firstNode = getNodeByName(newFirstNodeName),
secondNode = getNodeByName(newSecondNodeName);
return ((firstNode == null)||(secondNode == null)) ? false :
addEdge(new Edge(getNodeByName(newFirstNodeName),
getNodeByName(newSecondNodeName)));
}

public boolean addEdge(Edge newEdge){
if(newEdge.getFirstNode() == newEdge.getSecondNode())
return false;
for(int i = 0; i < edges.size(); i++)
if(edges.get(i).equals(newEdge))
return false;
edges.add(newEdge);
newEdge.getFirstNode().addEdge(newEdge);
newEdge.getSecondNode().addEdge(newEdge);
return true;
}

public boolean removeEdge(String firstNodeName, String secondNodeName){
```

```
return removeEdge(getNodeByName(firstNodeName), getNodeByName(secondNodeName));
}

public boolean removeEdge(Node firstNode, Node secondNode){
if((firstNode == null)||(secondNode == null))
return false;
Edge currentEdge = new Edge(firstNode, secondNode);
for(int i = 0; i < edges.size(); i++)
if(edges.get(i).equals(currentEdge))
return removeEdge(edges.get(i));
return false;
}

public boolean removeEdge(Edge removedEdge){
int tempIndex = edges.indexOf(removedEdge);
if(tempIndex == -1)
return false;
edges.remove(tempIndex);
ArrayList<Edge> tempAdjacentEdges;
tempAdjacentEdges = removedEdge.getFirstNode().getAdjacentEdges();
tempIndex = tempAdjacentEdges.indexOf(removedEdge);
if(tempIndex != -1)
tempAdjacentEdges.remove(tempIndex);
tempAdjacentEdges = removedEdge.getSecondNode().getAdjacentEdges();
tempIndex = tempAdjacentEdges.indexOf(removedEdge);
if(tempIndex != -1)
tempAdjacentEdges.remove(tempIndex);
return true;
}

public ArrayList<Node> getNodes(){ return nodes; }

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

public ArrayList<Edge> getBridges(){ return bridges; }

public ArrayList<Object> getVisualizationObjects(){
return visualizationObjects;
}

public void runAlgorithm(){
bridges.clear();
visualizationObjects.clear();
for(int i = 0; i < edges.size(); i++){
edges.get(i).reset();
}
for(int i = 0; i < nodes.size(); i++){
nodes.get(i).reset();
}
int timeDFS = 0;
for(int i = 0; i < nodes.size(); i++)
if(nodes.get(i).getAlgorithmValues()[0] == 0)
timeDFS = algorithmDFS(nodes.get(i), timeDFS);
for(int i = 0; i < edges.size(); i++)
if(edges.get(i).isBridge)
bridges.add(edges.get(i));
}

private int algorithmDFS(Node nodeComponent, int timeDFS){
timeDFS++;
```

```
Node nextNode;
Edge adjacentEdge;
ArrayList<Edge> adjacentEdges = nodeComponent.getAdjacentEdges();
int[] newAlgorithmValues = new int[4];
int[] nextNodeAlgorithmValues;
newAlgorithmValues[0] = timeDFS;
newAlgorithmValues[1] = 1;
newAlgorithmValues[2] = timeDFS;
newAlgorithmValues[3] = timeDFS;
nodeComponent.setAlgorithmValues(newAlgorithmValues);
visualizationObjects.add(nodeComponent);

for(int i = 0; i < adjacentEdges.size(); i++){
adjacentEdge = adjacentEdges.get(i);
if(nodeComponent.getParentTreeEdge() == adjacentEdge)
continue;
nextNode = adjacentEdge.getAdjacentNode(nodeComponent);
nextNodeAlgorithmValues = nextNode.getAlgorithmValues();
if(nextNodeAlgorithmValues[1] == 0){
if(adjacentEdge.getFirstNode() != nodeComponent)
adjacentEdge.swapNodes();
adjacentEdge.isForward = true;
nextNode.setParentTreeEdge(adjacentEdge);
visualizationObjects.add(adjacentEdge);
timeDFS = algorithmDFS(nextNode, timeDFS);
newAlgorithmValues[1] += nextNodeAlgorithmValues[1];
if(nextNodeAlgorithmValues[2] < newAlgorithmValues[2])
newAlgorithmValues[2] = nextNodeAlgorithmValues[2];
if(nextNodeAlgorithmValues[3] > newAlgorithmValues[3])
newAlgorithmValues[3] = nextNodeAlgorithmValues[3];
} else {
if(adjacentEdge.getFirstNode() != nodeComponent)
adjacentEdge.swapNodes();
if(!adjacentEdge.isBackward){
visualizationObjects.add(adjacentEdge);
visualizationObjects.add(adjacentEdge);
}
adjacentEdge.isBackward = true;
if(nextNodeAlgorithmValues[0] < newAlgorithmValues[2])
newAlgorithmValues[2] = nextNodeAlgorithmValues[0];
if(nextNodeAlgorithmValues[0] > newAlgorithmValues[3])
newAlgorithmValues[3] = nextNodeAlgorithmValues[0];
}
}
nodeComponent.setAlgorithmValues(newAlgorithmValues);
visualizationObjects.add(nodeComponent);
if(nodeComponent.getParentTreeEdge() != null){
visualizationObjects.add(nodeComponent.getParentTreeEdge());
if((newAlgorithmValues[0] == newAlgorithmValues[2])
&&(newAlgorithmValues[3] < newAlgorithmValues[0] + newAlgorithmValues[1]))
nodeComponent.getParentTreeEdge().isBridge = true;
}
return timeDFS;
}

public void print(){
for(int i = 0; i < nodes.size(); i++){
int[] nodeAlgorithmValues = nodes.get(i).getAlgorithmValues();
Point nodePoint = nodes.get(i).getPoint();
System.out.printf(”Node: %s | AlgValues: %d,%d,%d,%d | Point: %d %d\n”,
```

```
nodes.get(i).getName(), nodeAlgorithmValues[0],
nodeAlgorithmValues[1], nodeAlgorithmValues[2],
nodeAlgorithmValues[3], nodePoint.x, nodePoint.y);
}
System.out.printf("\n");
for(int i = 0; i < edges.size(); i++)
System.out.printf("Edge: %s-%s | Forward/Backward: %b/%b | isBridge: %b\n",
edges.get(i).getFirstNode().getName(),
edges.get(i).getSecondNode().getName(),
edges.get(i).isForward, edges.get(i).isBackward,
edges.get(i).isBridge);
System.out.printf("\n");
}

public void reset(){
nodes.clear();
edges.clear();
bridges.clear();
visualizationObjects.clear();
}
}
```

Название файла: EdgeImage.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.AffineTransform;

import algorithm.Point;

public class EdgeImage extends JComponent{
private static final long serialVersionUID = 1L;
private Point start;
private Point end;
private final int ARR_SIZE = 10;

private boolean isArrow = false;

private Color color = Color.BLACK;


EdgeImage(Point start, Point end){
this.start = new Point(start.x, start.y);
this.end = new Point(end.x, end.y);
setBounds(0, 0, 10000, 10000);
};

public void transformIntoArrow(){
int x1 = start.x - end.x;
int y1 = start.y - end.y;
int x2 = 0;
int y2 = 0;

double a = y2 - y1;
double b = x1 - x2;
double c = y1 * (x2 - x1) - x1 * (y2 - y1);
int r = 35;

double x0 = -a * c / (a * a + b * b), y0 = -b * c / (a * a + b * b);
```

```
double d = r * r - c * c / (a * a + b * b);
double mult = Math.sqrt(Math.abs(d) / (a * a + b * b));
double ax, ay, bx, by;
ax = x0 + b * mult;
bx = x0 - b * mult;
ay = y0 - a * mult;
by = y0 + a * mult;
if (Math.abs(ax - start.x) + Math.abs(ay - start.y) <
Math.abs(bx - start.x) + Math.abs(by - start.y)) {
end.x =  (int) ax + end.x;
end.y =  (int) ay + end.y;
} else {
end.x = - (int) bx + end.x;
end.y = - (int) by + end.y;
}

isArrow = true;

}

public void drawThickLine(
Graphics g, int x1, int y1, int x2, int y2, int thickness, Color c) {

g.setColor(c);
int dX = x2 - x1;
int dY = y2 - y1;

double lineLength = Math.sqrt(dX * dX + dY * dY);

double scale = (double)(thickness) / (2 * lineLength);

double ddx = -scale * (double)dY;
double ddy = scale * (double)dX;
ddx += (ddx > 0) ? 0.5 : -0.5;
ddy += (ddy > 0) ? 0.5 : -0.5;
int dx = (int)ddx;
int dy = (int)ddy;

int xPoints[] = new int[4];
int yPoints[] = new int[4];

xPoints[0] = x1 + dx; yPoints[0] = y1 + dy;
xPoints[1] = x1 - dx; yPoints[1] = y1 - dy;
xPoints[2] = x2 - dx; yPoints[2] = y2 - dy;
xPoints[3] = x2 + dx; yPoints[3] = y2 + dy;

g.fillPolygon(xPoints, yPoints, 4);
}


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


public void paintComponent(Graphics g1){
g1.setColor(color);
if(isArrow) {
double dx = end.x - start.x, dy = end.y - start.y;
double angle = Math.atan2(dy, dx);
int len = (int) Math.sqrt(dx*dx + dy*dy);
AffineTransform at = AffineTransform.getTranslateInstance(start.x, start.y);
```

```
at.concatenate(AffineTransform.getRotateInstance(angle));
Graphics2D g = (Graphics2D) g1.create();
g.transform(at);
len += 5;
drawThickLine(g, 0, 0, len, 0, 3, color);
g.fillPolygon(new int[] {len, len-ARR_SIZE, len-ARR_SIZE, len},
new int[] {0, -ARR_SIZE, ARR_SIZE, 0}, 3);
}
else
g1.drawLine(start.x, start.y, end.x, end.y);
}
}
```

Название файла: NodeImage.java

```
import javax.swing.*;
import java.awt.*;
import java.sql.Array;

import algorithm.Point;

public class NodeImage extends JComponent{
private static final long serialVersionUID = 1L;
public static final int SIZE = 60;
private Point center;
private String name;
private String[] text;
private Color color = Color.CYAN;
NodeImage(String text, Point center){
this.name = text;
this.center = center;
setBounds(0, 0, 10000, 10000);
};

public void setText(String[] text){
this.text = text;
}


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

public void paintComponent(Graphics g){
g.setColor(color);
g.fillOval(center.x-SIZE/2,center.y-SIZE/2, SIZE, SIZE);
g.setColor(Color.black);
Font f = new Font(”Times New Roman”, Font.PLAIN, 20);
g.setFont(f);
g.drawString(name,  center.x-7, center.y+5);
if(text != null) {
g.drawString(”N = ” + text[0], center.x + SIZE / 2 + 5, center.y - SIZE / 2);
g.drawString(”D = ” + text[1], center.x + SIZE / 2 + 5, center.y - SIZE / 2 + 20);
g.drawString(”L = ” + text[2], center.x + SIZE / 2 + 5, center.y - SIZE / 2 + 40);
g.drawString(”H = ” + text[3], center.x + SIZE / 2 + 5, center.y - SIZE / 2 + 60);
}
}
}
```

Название файла: PaintAreaMode.java

```
public enum PaintAreaMode {
Node,
Edge1,
Edge2,
Erase,
None
}
```

Название файла: PaintArea.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;

import algorithm.Point;
import algorithm.Edge;
import algorithm.Node;
import algorithm.Graph;

public class PaintArea extends JLayeredPane {

private final Graph graph;
private int x1, y1, x2, y2;
PaintAreaMode currentMode;

private class MyMouseHandler extends MouseAdapter {
public void mousePressed(MouseEvent e) {
x1 = e.getX();
y1 = e.getY();

switch(currentMode) {
case Node:
if (findNodeByCoordinate(new Point(x1, y1), 2500) != null) {
JOptionPane.showMessageDialog(null, ”Do not put nodes too close.”,
”Illegal node placement”,
JOptionPane.WARNING_MESSAGE);
return;
}
String s = (String) JOptionPane.showInputDialog(null,
”Enter a Node name (single character):\n”);
if(s != null && s.length() == 1) {
if(graph.addNode(s)) {
Node node = graph.getNodeByName(s);
node.setPoint(x1, y1);
drawNode(node);
}
else{
JOptionPane.showMessageDialog(null, ”Nodes cannot be named with the same name.”,
”Illegal node name”, JOptionPane.WARNING_MESSAGE);
}
}
else {
if(s != null) {
JOptionPane.showMessageDialog(null, ”Node name must contain a single symbol.”,
”Illegal node name”, JOptionPane.WARNING_MESSAGE);
}
}
break;
```

```
case Edge1:
Node nd = findNodeByCoordinate(new Point(x1, y1), 100);
if(nd != null) {
x2 = nd.getPoint().x;
y2 = nd.getPoint().y;
currentMode = PaintAreaMode.Edge2;
}
break;
case Edge2:
Node nd1 = findNodeByCoordinate(new Point(x1, y1), 100);
if(nd1 != null) {
Edge edge = new Edge(findNodeByCoordinate(new Point(x2, y2), 100), nd1);
if(graph.addEdge(edge)) {
drawEdge(edge);
}
else {
JOptionPane.showMessageDialog(null, ”Parallel edges and loops are not supported.”,
”Illegal edge”, JOptionPane.WARNING_MESSAGE);
}
}
currentMode = PaintAreaMode.Edge1;
break;
case Erase:
Node deletedNode = findNodeByCoordinate(new Point(x1, y1), 0);
if(deletedNode != null) {
graph.removeNode(deletedNode);
clear();
drawGraph();
}
else{
Edge deletedEdge = findEdgeByCoordinate(new Point(x1, y1));
if(deletedEdge != null){
graph.removeEdge(deletedEdge);
clear();
drawGraph();
}
}
break;
case None:
break;
}
}

public void mouseDragged(MouseEvent e) {
}
}

public PaintArea(Graph graph) {
this.graph = graph;

setOpaque(true);
setBackground(Color.WHITE);
setBorder(BorderFactory.createLineBorder(Color.black));


currentMode = PaintAreaMode.None;

MyMouseHandler handler = new MyMouseHandler();
this.addMouseListener(handler);
this.addMouseMotionListener(handler);
```

```
}

public void setMode(PaintAreaMode md){
currentMode = md;
}

public void drawNode(Node node) {
NodeImage img = new NodeImage(node.getName(), node.getPoint());
this.add(img);
this.moveToFront(img);
}

public void drawEdge(Edge edge) {
this.add(new EdgeImage(edge.getFirstNode().getPoint(),
edge.getSecondNode().getPoint()));
}

public void drawArrow(Edge edge) {
EdgeImage arrow = new EdgeImage(edge.getFirstNode().getPoint(),
edge.getSecondNode().getPoint());
arrow.transformIntoArrow();
add(arrow);
}

public void drawArrow(Edge edge, Color color) {
EdgeImage arrow = new EdgeImage(edge.getFirstNode().getPoint(),
edge.getSecondNode().getPoint());
arrow.transformIntoArrow();
arrow.setColor(color);
add(arrow);
}

public void drawNode(Node node, Color color) {
NodeImage img = new NodeImage(node.getName(), node.getPoint());
img.setColor(color);
this.add(img);
this.moveToFront(img);
}

public void drawNode(Node node, Color color, String[] text) {
NodeImage img = new NodeImage(node.getName(), node.getPoint());
img.setText(text);
img.setColor(color);
this.add(img);
this.moveToFront(img);
}

public void drawEdge(Edge edge, Color color) {
EdgeImage img = new EdgeImage(edge.getFirstNode().getPoint(),
edge.getSecondNode().getPoint());
img.setColor(color);
this.add(img);
this.moveToFront(img);
}

public void drawGraph() {
for(Node node: this.graph.getNodes()) {
this.drawNode(node);
}
for(Edge edge: this.graph.getEdges()) {
```

```
this.drawEdge(edge);
}
this.revalidate();
this.repaint();
}

public void clear() {
Component[] components = this.getComponents();
while(components.length > 0) {
this.remove(components[0]);
components = this.getComponents();
}
}

private Node findNodeByCoordinate(Point point, int EPS){
for(Node nd: graph.getNodes()){
if(Math.pow((point.x - nd.getPoint().x), 2) +
Math.pow((point.y - nd.getPoint().y), 2) <
Math.pow(NodeImage.SIZE/2, 2) + EPS )
return nd;
}
return null;
}

private Edge findEdgeByCoordinate(Point point){
for(Edge edge: graph.getEdges()){
int EPS = 10;
int x1 = edge.getFirstNode().getPoint().x;
int y1 = edge.getFirstNode().getPoint().y;
int x2 = edge.getSecondNode().getPoint().x;
int y2 = edge.getSecondNode().getPoint().y;
int minx = Math.min(x1, x2);
int miny = Math.min(y1, y2);
int maxx = Math.max(x1, x2);
int maxy = Math.max(y1, y2);
if(point.x >= minx && point.y >= miny && point.x <= maxx && point.y <= maxy &&
point.x * (y2 - y1) - point.y * (x2 - x1) - x1 * (y2 - y1) +
y1 * (x2 - x1) <= EPS )
return edge;
}
return null;
}
}
```

Название файла: MainWindow.java

```
import java.util.*;

import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import algorithm.Node;
import algorithm.Edge;
import algorithm.Graph;

public class MainWindow extends JFrame {
private final Graph graph;
```

```
private VisualStatus context;

private PaintArea area;

private JTextArea textArea;

private JScrollPane scroll;

private ButtonGroup buttonGroup;

private JRadioButton buttonDrawNode;
private JRadioButton buttonDrawEdge;
private JRadioButton buttonErase;

private JButton buttonOpenFile;
private JButton buttonSaveInFile;
private JButton buttonStop;
private JButton buttonNext;
private JButton buttonStart;

private DrawNodeActionListener buttonDrawNodeActionListener;
private DrawEdgeActionListener buttonDrawEdgeActionListener;
private EraseActionListener buttonEraseActionListener;
private StartActionListener buttonStartActionListener;
private NextActionListener buttonNextActionListener;
private OpenFileActionListener buttonOpenFileActionListener;
private StopActionListener buttonStopActionListener;
private SaveInFileActionListener buttonSaveInFileActionListener;

private PaintAreaMode stashedMode;

public MainWindow(){
graph = new Graph();

setTitle(”Tarjan’s bridge-finding algorithm”);

setDefaultCloseOperation(EXIT_ON_CLOSE);

setMinimumSize(new Dimension(900, 800));

area = new PaintArea(this.graph);

textArea = new JTextArea(10, 1);
textArea.setOpaque(true);
textArea.setBackground(Color.WHITE);
textArea.setEditable(true);
scroll = new JScrollPane(textArea);
scroll.setBorder(BorderFactory.createLineBorder(Color.black));

buttonGroup = new ButtonGroup();

buttonDrawNode = new JRadioButton(”Draw Node”, false);
buttonDrawEdge = new JRadioButton(”Draw Edge”, false);
buttonErase = new JRadioButton(”Erase”, false);

buttonGroup.add(buttonDrawNode);
buttonGroup.add(buttonDrawEdge);
buttonGroup.add(buttonErase);
```

```
buttonOpenFile = new JButton(”Open File”);
buttonSaveInFile = new JButton(”Save In File”);
buttonStop = new JButton(”Stop”);
buttonStop.setEnabled(false);
buttonNext = new JButton(”Next”);
buttonNext.setEnabled(false);
buttonStart = new JButton(”Start”);

buttonDrawNodeActionListener = new DrawNodeActionListener();
buttonDrawNode.addActionListener(buttonDrawNodeActionListener);

buttonDrawEdgeActionListener = new DrawEdgeActionListener();
buttonDrawEdge.addActionListener(buttonDrawEdgeActionListener);

buttonEraseActionListener = new EraseActionListener();
buttonErase.addActionListener(buttonEraseActionListener);

buttonStartActionListener = new StartActionListener();
buttonStart.addActionListener(buttonStartActionListener);

buttonNextActionListener = new NextActionListener();
buttonNext.addActionListener(buttonNextActionListener);

buttonOpenFileActionListener = new OpenFileActionListener();
buttonOpenFile.addActionListener(buttonOpenFileActionListener);

buttonStopActionListener = new StopActionListener();
buttonStop.addActionListener(buttonStopActionListener);

buttonSaveInFileActionListener = new SaveInFileActionListener();
buttonSaveInFile.addActionListener(buttonSaveInFileActionListener);

Container container = getContentPane();
container.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);

container.setLayout(new GridBagLayout());
GridBagConstraints constraints = new GridBagConstraints();

constraints.fill = GridBagConstraints.BOTH;
constraints.insets = new Insets(10, 10, 10, 10);

constraints.weighty = 0.9;
constraints.weightx = 0.7;
constraints.gridheight = 6;
constraints.gridwidth = 4;
constraints.gridy = 0;
constraints.gridx = 0;
container.add(area, constraints);

constraints.weighty = 0;
constraints.weightx = 0;

constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.gridheight = 1;
constraints.gridy = 6;
constraints.gridx = 0;
container.add(scroll, constraints);


constraints.anchor = GridBagConstraints.NORTH;
```

```
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.gridwidth = 2;
constraints.gridx = 5;
constraints.gridy = 0;
container.add(buttonDrawNode, constraints);

constraints.gridx = 5;
constraints.gridy = 1;
container.add(buttonDrawEdge, constraints);

constraints.gridx = 5;
constraints.gridy = 2;
container.add(buttonErase, constraints);

constraints.gridx = 5;
constraints.gridy = 4;
container.add(buttonOpenFile, constraints);

constraints.gridx = 5;
constraints.gridy = 5;
container.add(buttonSaveInFile, constraints);

constraints.anchor = GridBagConstraints.SOUTH;
constraints.gridwidth = 1;
constraints.gridx = 5;
constraints.gridy = 5;
container.add(buttonStop, constraints);

constraints.gridx = 6;
constraints.gridy = 5;
container.add(buttonNext, constraints);

constraints.anchor = GridBagConstraints.CENTER;
constraints.gridwidth = 2;
constraints.gridx = 5;
constraints.gridy = 6;
container.add(buttonStart, constraints);

pack();
setVisible(true);
}

public static void main(String[] args) throws UnsupportedLookAndFeelException,
ClassNotFoundException, InstantiationException,
IllegalAccessException {
new MainWindow();
}

public class DrawNodeActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
area.setMode(PaintAreaMode.Node);
}
}

public class DrawEdgeActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
area.setMode(PaintAreaMode.Edge1);
}
}
```

```
public class EraseActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
area.setMode(PaintAreaMode.Erase);
}
}

public class OpenFileActionListener implements ActionListener {
private final String SAVEFILE_SIGNATURE = ”BFSV\u0002”;
private final String ERROR_PREFIX = ”ERROR: ”;
private final String INFO_PREFIX = ”INFO: ”;

public void actionPerformed(ActionEvent e) {
textArea.setText(””);
FileReaderWrapper wrapper = new FileReaderWrapper();
if(wrapper.open(MainWindow.this) == null) {
textArea.append(INFO_PREFIX + ”File has not been chosen.\n”);
return;
}

ArrayList<ArrayList<String>> content = wrapper.read();
if(content == null) {
textArea.setText(ERROR_PREFIX + ”Failed to parse file.\n”);
JOptionPane.showMessageDialog(null, ”Failed to parse file.”,
”Input error”, JOptionPane.ERROR_MESSAGE);
return;
}

graph.reset();

boolean nonFatalErrorFlag = false;
if(content.size() > 0 &&
content.get(0).size() > 0 && content.get(0).get(0).equals(SAVEFILE_SIGNATURE)) {
content.remove(0);
try {
nonFatalErrorFlag = createGraphFromSaveData(content);
}
catch(Exception ex) {
area.clear();
JOptionPane.showMessageDialog(null,
”File structure does not correspond with a save file structure.”,
”Fatal file interpretation error”, JOptionPane.ERROR_MESSAGE);
}
}
else {
nonFatalErrorFlag = createGraphFromInput(content);
}

area.clear();
area.drawGraph();

if(nonFatalErrorFlag) {
JOptionPane.showMessageDialog(null,
”Errors occurred while creating a graph. Check log for information.”,
”File interpretation error”, JOptionPane.WARNING_MESSAGE);
}
}

private boolean createGraphFromInput(ArrayList<ArrayList<String>> content) {
boolean errorFlag = false;
for(ArrayList<String> line: content) {
```

```
if(line.size() != 2) {
errorFlag = true;
textArea.append(ERROR_PREFIX+
”Input line doesn’t describe an edge: ”+line.toString()+”. Ignored.\n”);
continue;
}
if(line.get(0).length() == 1) graph.addNode(line.get(0));
if(line.get(1).length() == 1) graph.addNode(line.get(1));
if(!(graph.addEdge(line.get(0), line.get(1)))) {
errorFlag = true;
textArea.append(ERROR_PREFIX +
”Edge ” + line.get(0) + ” - ” + line.get(1) + ” is not allowed.\n”);
textArea.append(”\t” +
”Either invalid node name has been encountered, or this edge already ” +
”exists, or this edge is a loop.\n”);
}
}

int radius = Math.min(area.getBounds().height,
area.getBounds().width) / 2 - NodeImage.SIZE/2;
int xCenter = area.getBounds().width/2;
int yCenter = area.getBounds().height/2;

ArrayList<Node> lst = graph.getNodes();
for(int i = 0; i < lst.size(); ++i) {
double angle = 2*i*Math.PI/lst.size();
lst.get(i).setPoint(new algorithm.Point((int)(xCenter + radius*Math.cos(angle)),
(int)(yCenter + radius*Math.sin(angle))));
}
return errorFlag;
}

private boolean createGraphFromSaveData(ArrayList<ArrayList<String>> content) {
boolean errorFlag = false;
int expectedNodesCount;
textArea.append(INFO_PREFIX + ”Save file signature recognized.\n”);
try {
expectedNodesCount = Integer.parseInt(content.get(0).get(0));
content.remove(0);
}
catch(Exception ex) {
throw ex;
}

int nodeCounter = 0;
ArrayList<String> names = new ArrayList<String>();
ArrayList<Integer> xValues = new ArrayList<Integer>();
ArrayList<Integer> yValues = new ArrayList<Integer>();

while(content.size() > 0 && content.get(0).size() >= 3) {
ArrayList<String> line = content.get(0);
try {
names.add(line.get(0));
xValues.add(Integer.valueOf(Integer.parseInt(line.get(1))));
yValues.add(Integer.valueOf(Integer.parseInt(line.get(2))));
}
catch(Exception ex) {
textArea.append(ERROR_PREFIX + ”Node ” + line.get(0) +
” coordinates don’t seem to be valid.\n”);
throw ex;
```

```
}
if(line.size() > 3) {
errorFlag = true;
textArea.append(ERROR_PREFIX +
”Extra characters in a line describing node.\n”);
}
if(xValues.get(xValues.size() - 1) > area.getWidth() ||
yValues.get(yValues.size() - 1) > area.getHeight()) {
textArea.append(INFO_PREFIX +
”Coordinates larger than the canvas size may cause problems.\n”);
}
++nodeCounter;
content.remove(0);
}
if(nodeCounter != expectedNodesCount) {
errorFlag = true;
textArea.append(ERROR_PREFIX +
”Declared number of nodes does not match number of nodes present in a file.\n”);
}

for(int i = 0; i < names.size(); ++i) {
if(names.get(i).length() == 1) {
if(graph.addNode(names.get(i))) {
if(xValues.get(i) < NodeImage.SIZE/2) {
xValues.set(i, NodeImage.SIZE/2);
errorFlag = true;
textArea.append(ERROR_PREFIX+”Node is too close to the upper-left border or ” +
”has negative coordinates.
Changed node placement to the very edge of the working area.\n”);
}
if(yValues.get(i) < NodeImage.SIZE/2) {
yValues.set(i, NodeImage.SIZE/2);
errorFlag = true;
textArea.append(ERROR_PREFIX+”Node is too close to the upper-left border or ” +
”has negative coordinates.
Changed node placement to the very edge of the working area.\n”);
}
graph.getNodeByName(names.get(i)).setPoint(xValues.get(i), yValues.get(i));
}
else {
errorFlag = true;
textArea.append(ERROR_PREFIX + ”Node ” + names.get(i) + ” already exists.\n”);
}
}
else {
errorFlag = true;
textArea.append(ERROR_PREFIX + ”Illegal node name: ’” + names.get(i) + ”’\n”);
}
}

for(ArrayList<String> line: content) {
if(line.size() == 2) {
if(!graph.addEdge(line.get(0), line.get(1))) {
errorFlag = true;
textArea.append(ERROR_PREFIX + ”Edge ” + line.get(0) + ” - ” + line.get(1) +
” could not be added. Either nodes don’t exist or edge
already exists or edge is a loop.\n”);
}
}
else {
```

```
errorFlag = true;
textArea.append(ERROR_PREFIX + ”Line does not describe an edge: ” +
line + ” Ignored.\n”);
}
}

return errorFlag;
}
}

public class SaveInFileActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
FileWriterWrapper wrapper = new FileWriterWrapper();
if(!wrapper.open(MainWindow.this)) {
textArea.append(”INFO: File was not chosen.\n”);
return;
}
if(!wrapper.write(graph)) {
textArea.append(”ERROR: Failed to save graph in file.\n”);
}
}
}

public class StartActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
stashedMode = area.currentMode;
area.currentMode = PaintAreaMode.None;
for(Enumeration<AbstractButton> enumeration = buttonGroup.getElements();
enumeration.hasMoreElements();) {
AbstractButton processedButton = enumeration.nextElement();
processedButton.setEnabled(false);
}
buttonOpenFile.setEnabled(false);
buttonSaveInFile.setEnabled(false);
buttonStart.setEnabled(false);
buttonStop.setEnabled(true);
buttonNext.setEnabled(true);
textArea.setEditable(false);

ArrayList<Object> visObjects = graph.getVisualizationObjects();
textArea.append(”\nStarting algorithm... Click ’Next’ to explore steps.\n====\n”);
graph.runAlgorithm();

context = new VisualStatus(visObjects, graph.getNodes(), graph.getBridges());
}
}

public class NextActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
context.step();
}
}

public class StopActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
for(Enumeration<AbstractButton> enumeration = buttonGroup.getElements();
enumeration.hasMoreElements();) {
enumeration.nextElement().setEnabled(true);
}
buttonOpenFile.setEnabled(true);
```

```
buttonSaveInFile.setEnabled(true);
buttonStart.setEnabled(true);
buttonStop.setEnabled(false);
buttonNext.setEnabled(false);
textArea.setEditable(true);
area.currentMode = stashedMode;
area.clear();
area.drawGraph();
}
}

public static enum Stage {
SEARCH,
DECISION,
FINISH
}

public class VisualStatus {
private Stage stage;
private ArrayList<Object> objectsInSearch;
private ArrayList<Edge> bridges;
private Deque<Edge> edgeStack;
private Deque<Node> nodeStack;

private final Color usedNodeColor = Color.LIGHT_GRAY;
private final Color activeNodeColor = Color.YELLOW;
private final Color backwardEdgeColor = Color.BLUE;
private final Color bridgeColor = Color.RED;


public VisualStatus(ArrayList<Object> objectsInSearch,
ArrayList<Node> nodes, ArrayList<Edge> bridges) {
this.objectsInSearch = objectsInSearch;
this.stage = Stage.SEARCH;
this.bridges = bridges;
this.nodeStack = new ArrayDeque<Node>();
this.edgeStack = new ArrayDeque<Edge>();
}

public void step() {
switch(stage) {
case SEARCH:
if(objectsInSearch.isEmpty()) {
textArea.append("Depth-first search has finished.\n");
this.stage = Stage.DECISION;
break;
}

Object current = objectsInSearch.get(0);

if(current instanceof Node) {
Node currentNode = (Node)current;
if(nodeStack.size() > 0 && nodeStack.peekFirst().equals(currentNode)) {
nodeStack.removeFirst();
textArea.append("No more ways from Node " + currentNode + "\n");
textArea.append("Now coefficients cat be calculated:\n" +
"\tNode number (entry time) N: " + currentNode.getAlgorithmValues()[0] + "\n" +
"\tNumber of nodes in subtree, including this node D: " +
currentNode.getAlgorithmValues()[1] + "\n" +
"\tL = min(N - D + 1; {L(\u03bc) | " + currentNode + " -> \u03bc}; " +
```

```
”{N(\u03bc) | ”+currentNode+” <- \u03bc}): ”+
currentNode.getAlgorithmValues()[2] +
”\n\tH = max(N; {H(\u03bc) | ” + currentNode + ” -> \u03bc}; {N(\u03bc) | ” +
currentNode +
” <- \u03bc}): ” + currentNode.getAlgorithmValues()[3] + ”\n”);
area.drawNode(currentNode, usedNodeColor, new String[]
{String.valueOf(currentNode.getAlgorithmValues()[0]),
String.valueOf(currentNode.getAlgorithmValues()[1]),
String.valueOf(currentNode.getAlgorithmValues()[2]),
String.valueOf(currentNode.getAlgorithmValues()[3])});
}
else {
nodeStack.addFirst(currentNode);
textArea.append(”Reached Node ” + currentNode + ”\n”);
area.drawNode(currentNode, activeNodeColor);
}
}
else {
Edge currentEdge = (Edge)current;
if(edgeStack.size() > 0 && edgeStack.peekFirst().equals(currentEdge)) {
edgeStack.removeFirst();
if(currentEdge.isForward) {
textArea.append(”Returning to ” + currentEdge.getFirstNode() + ”\n”);
}
objectsInSearch.remove(0);
this.step();
return;
}
else {
edgeStack.addFirst(currentEdge);
if(currentEdge.isBackward) {
textArea.append(”Edge ” + currentEdge + ” is not used because ” +
currentEdge.getFirstNode() +
” was visited earlier. This edge becomes backward-oriented.\n”);
area.drawArrow(currentEdge, backwardEdgeColor);
}
else {
textArea.append(”Choosing edge ” + currentEdge + ” to move on.\n”);
area.drawArrow(currentEdge);
}
}
}

objectsInSearch.remove(0);
break;
case DECISION:
area.clear();
textArea.append(
”Edge \u03bc -> \u03bd is a bridge if and only if
H(\u03bd) \u2264 N(\u03bd) and ” +
”L(\u03bd) > N(\u03bd) - D(\u03bd)\nBridges are highlighted:\n”);
for(Edge bridge: this.bridges) {
area.drawEdge(bridge, bridgeColor);
textArea.append(bridge + ”\n”);
}
area.drawGraph();
stage = Stage.FINISH;
break;
case FINISH:
area.clear();
```

```
area.drawGraph();
textArea.append(”Algorithm has finished.\n====\n”);
}

}
}


}
```

Название файла: FileReaderWrapper.java

```
import java.util.*;
import java.awt.Component;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;

class FileReaderWrapper {
private Path path = null;
//private String filename = null;

public FileReaderWrapper() {}

public String open(Component dialogParent) {
int dialogResult;
String filename;

JFileChooser chooser = new JFileChooser(”.”);
chooser.setFileFilter(new FileNameExtensionFilter(
”Input files and save files”, ”txt”, ”bfsv”));
chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
chooser.setAcceptAllFileFilterUsed(false);

try {
dialogResult = chooser.showOpenDialog(dialogParent);
}
catch(Exception e) {
return null;
}

if(dialogResult != JFileChooser.APPROVE_OPTION) return null;

filename = chooser.getSelectedFile().getAbsolutePath();
if(filename.length() == 0) return null;

try {
path = Paths.get(filename);
}
catch(Exception e) {
return null;
}

//this.filename = filename;
return filename;
}

public ArrayList<ArrayList<String>> read() {
```

```
ArrayList<String> lines = new ArrayList<String>();
try {
lines = (ArrayList<String>)Files.readAllLines(path, StandardCharsets.UTF_8);
}
catch(IOException e) {
return null;
}

ArrayList<ArrayList<String>> result = new ArrayList<ArrayList<String>>();
for(String line: lines) {
ArrayList<String> tmp = new ArrayList<String>();
for(String word: line.split("\\s+")) {
tmp.add(word);
}
//result.add((ArrayList<String>)Arrays.asList(line.split("\\s+")));
result.add(tmp);
}

return result;
}
}
```

Название файла: FileWriterWrapper.java

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import algorithm.Edge;
import algorithm.Node;
import algorithm.Graph;

public class FileWriterWrapper {
private Path path = null;
private static final String SAVEFILE_SIGNATURE = "BFSV\u0002";

public boolean open(Component dialogParent) {
int dialogResult;
String filename;

JFileChooser chooser = new JFileChooser(".");
chooser.setFileFilter(new FileNameExtensionFilter(
"Bridge-finder format save files", "bfsv"));
chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
chooser.setAcceptAllFileFilterUsed(false);

try {
dialogResult = chooser.showSaveDialog(dialogParent);
}
catch(Exception e) {
return false;
}

if(dialogResult != JFileChooser.APPROVE_OPTION) return false;

filename = chooser.getSelectedFile().getAbsolutePath();
if(filename.length() == 0) return false;
if(chooser.getSelectedFile().getName().matches("\\..*")) {
```

```
JOptionPane.showMessageDialog(null,
”File name cannot begin with ’.’”, ”Saving process aborted”,
JOptionPane.ERROR_MESSAGE);
return false;
}
if(!(filename.endsWith(”.bfsv”)) ||
(chooser.getSelectedFile().getName().matches(”.+\\..*\\.bfsv”))) {
String dirName = chooser.getSelectedFile().getParent();
String lastName = chooser.getSelectedFile().getName();
filename = dirName + File.separator +
lastName.split(”\\.”)[0] + ”.bfsv”;
JOptionPane.showMessageDialog(null,
”Forcefully changed file extension to ’.bfsv’”, ”Saving issue”,
JOptionPane.INFORMATION_MESSAGE);
}

try {
path = Paths.get(filename);
}
catch(Exception e) {
return false;
}

return true;
}

public boolean write(Graph graph) {
try(BufferedWriter writer =
Files.newBufferedWriter(path, StandardCharsets.UTF_8)){
writer.write(SAVEFILE_SIGNATURE);
writer.newLine();
writer.write(String.valueOf(graph.getNodes().size()));
writer.newLine();
for(Node node: graph.getNodes()) {
writer.write(node.toString() + ” ” +
String.valueOf(node.getPoint().x) + ” ” +
String.valueOf(node.getPoint().y));
writer.newLine();
}
for(Edge edge: graph.getEdges()) {
writer.write(edge.getFirstNode().toString() +
” ” + edge.getSecondNode().toString());
writer.newLine();
}
}
catch(IOException e) {
return false;
}

return true;
}
}
```

ПРИЛОЖЕНИЕ В
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

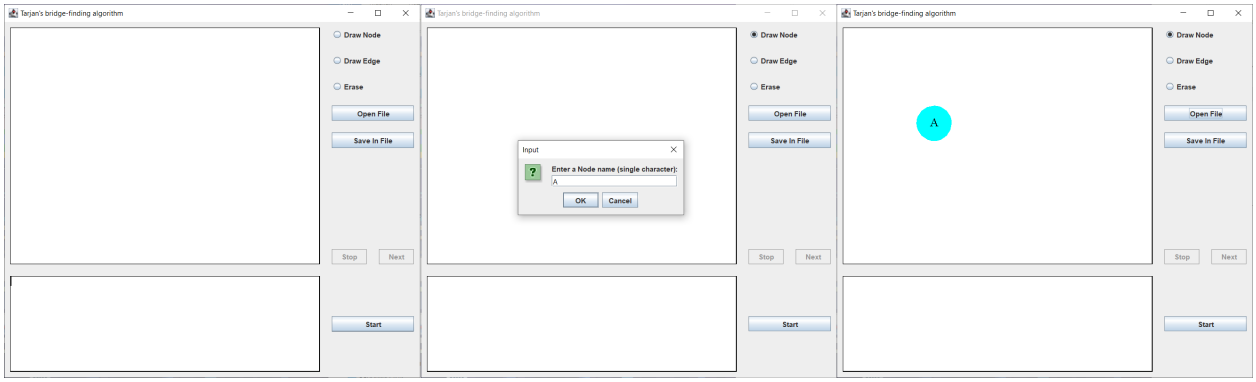

Рисунок 3 – Добавление вершины в граф

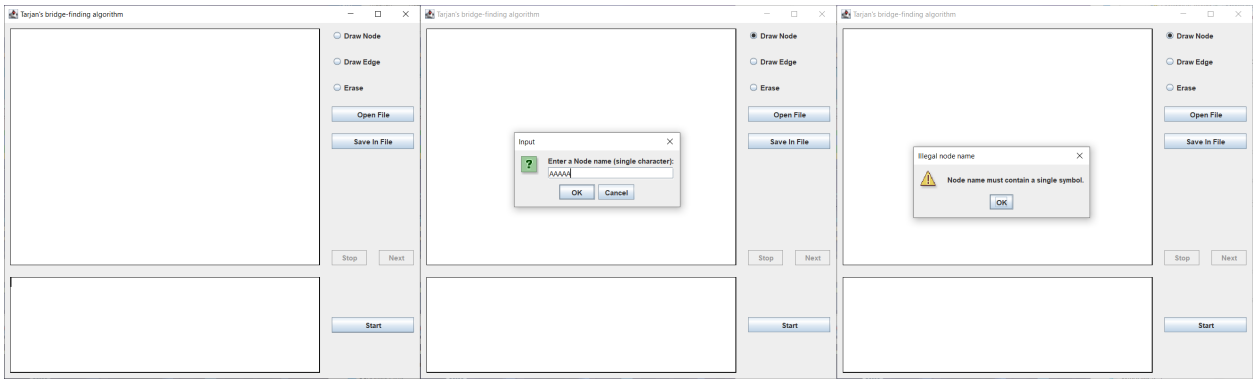

Рисунок 4 – Создание вершины с некорректным названием

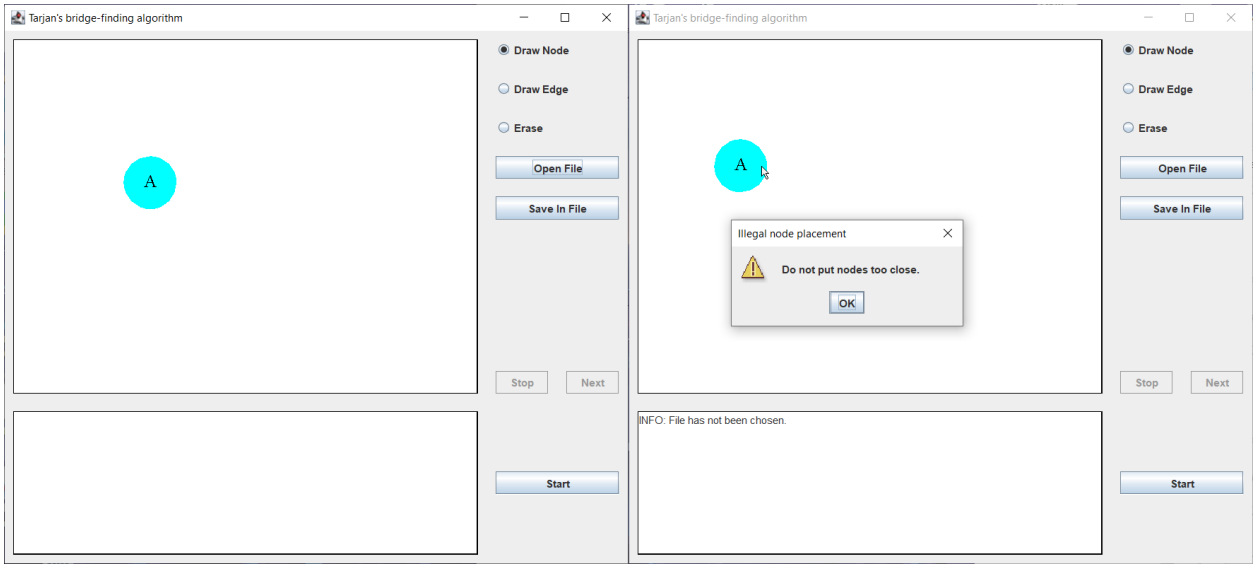

Рисунок 5 – Попытка создать вершину слишком близко к существующей

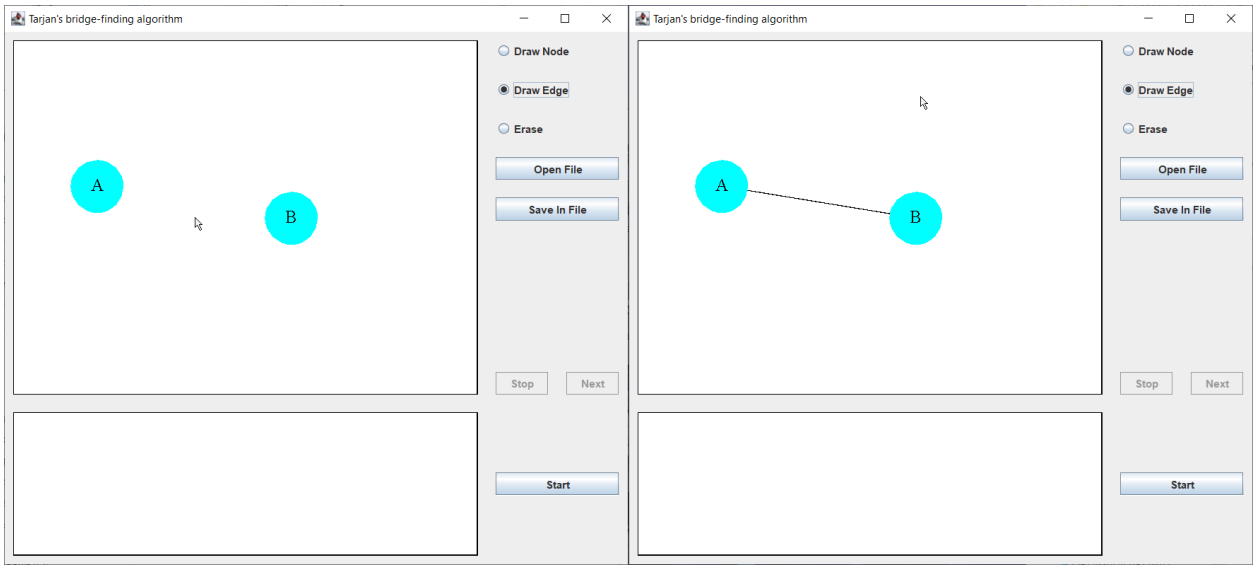

Рисунок 6 – Создание ребра

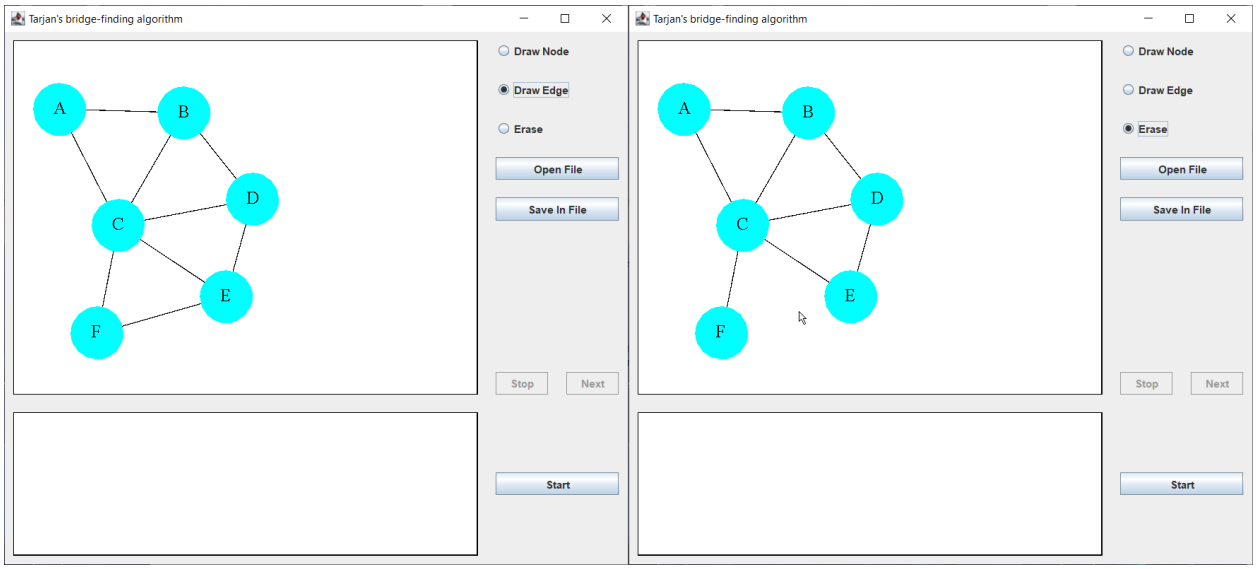

Рисунок 7 – Удаление ребра

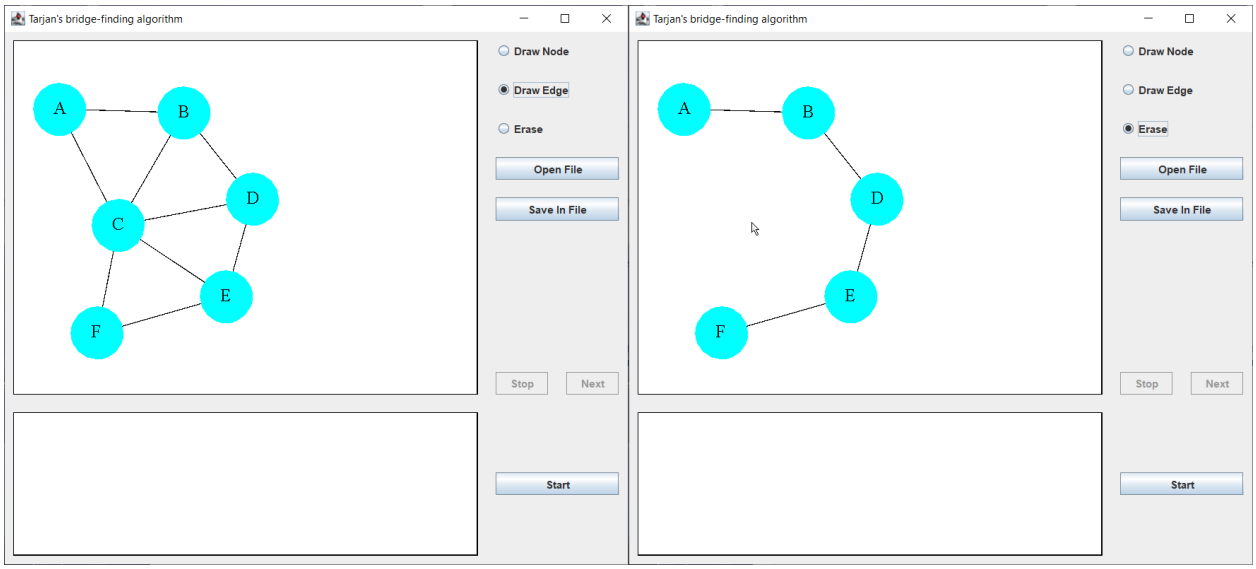

Рисунок 8 – Удаление вершины

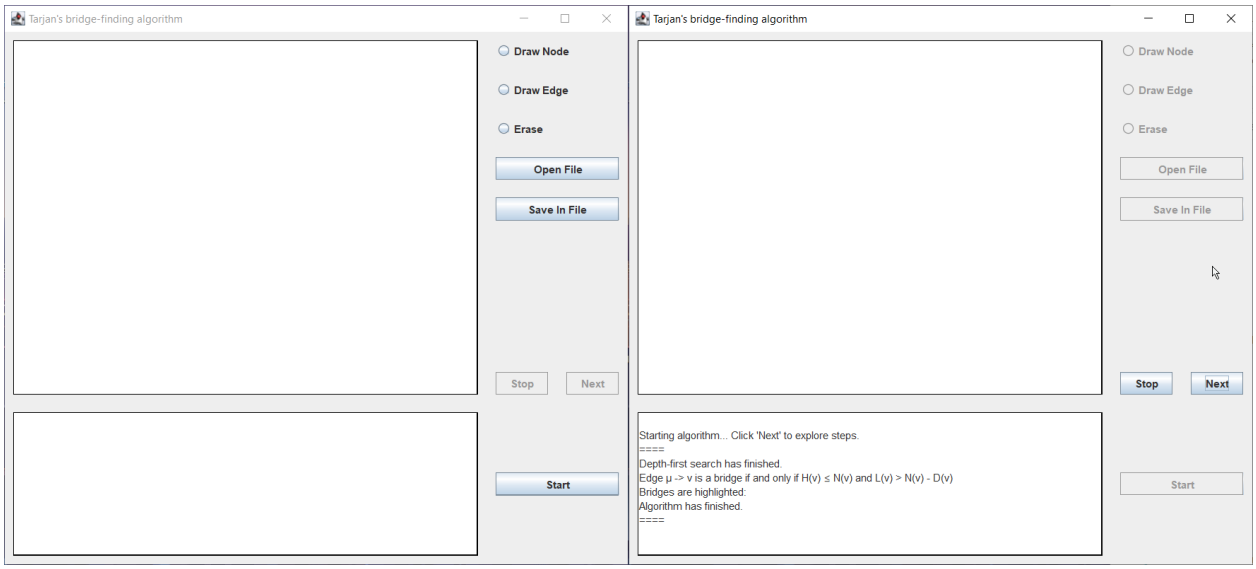

Рисунок 9 – Запуск алгоритма в пустом графе

Рисунок 10 – Запуск алгоритма в примитивном графе

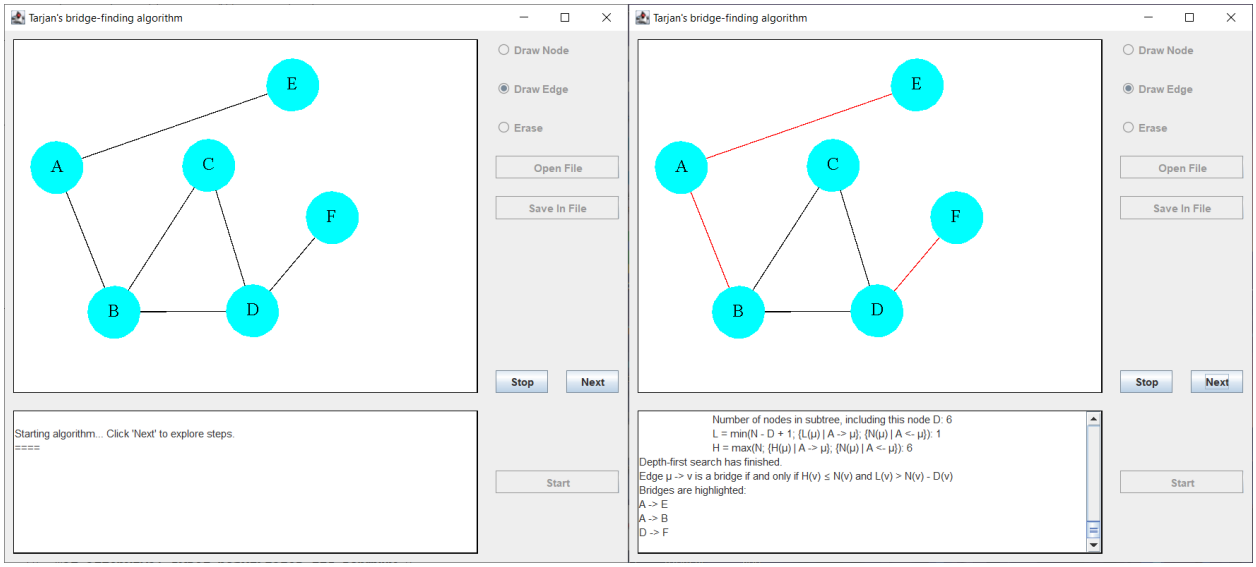

Рисунок 11 – Исходный граф и результат на произвольном графе

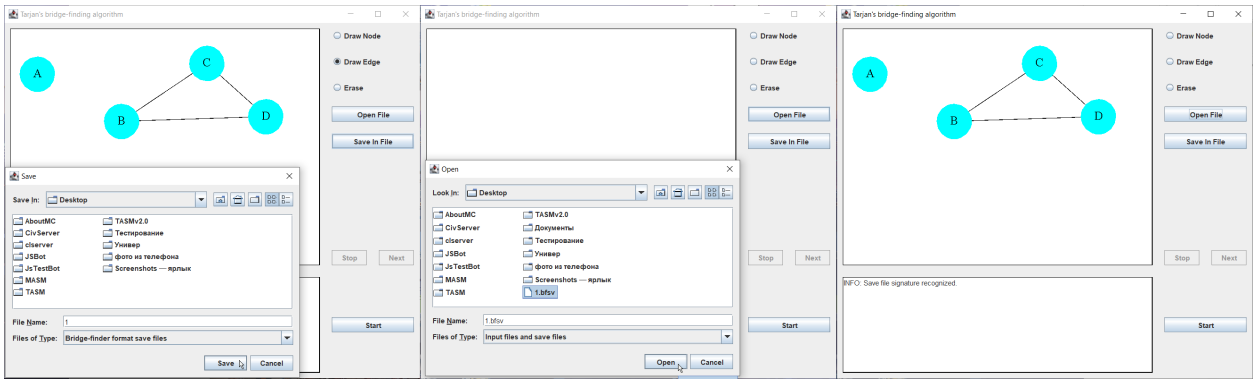

Рисунок 12 – Сохранение и последующая загрузка графа

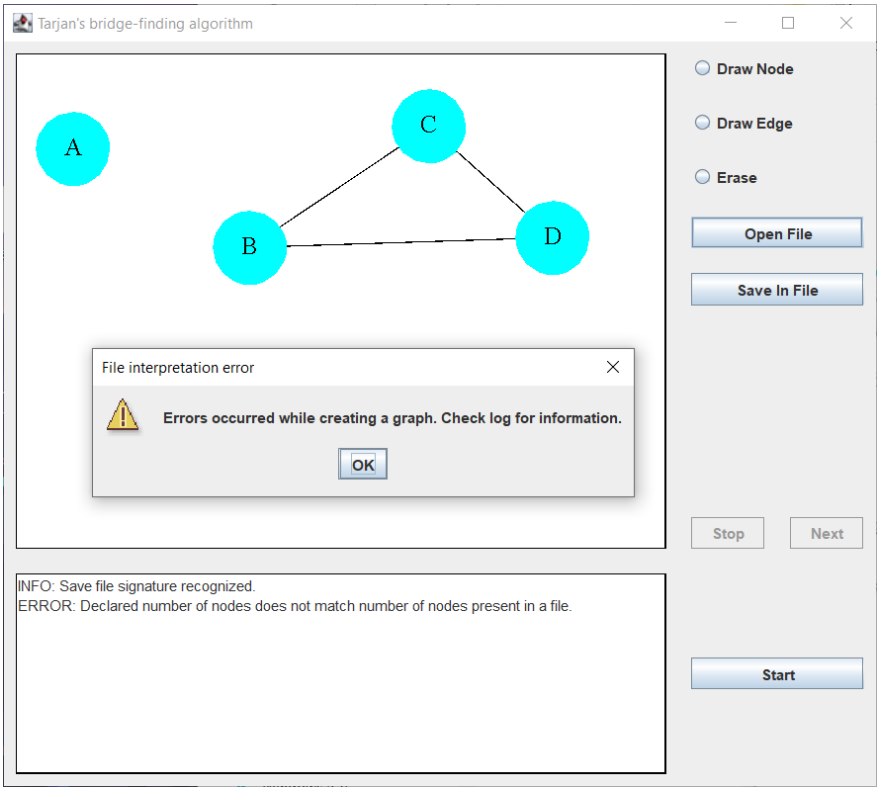

Рисунок 13 – Попытка загрузки графа из файла с неверно указанным количеством вершин

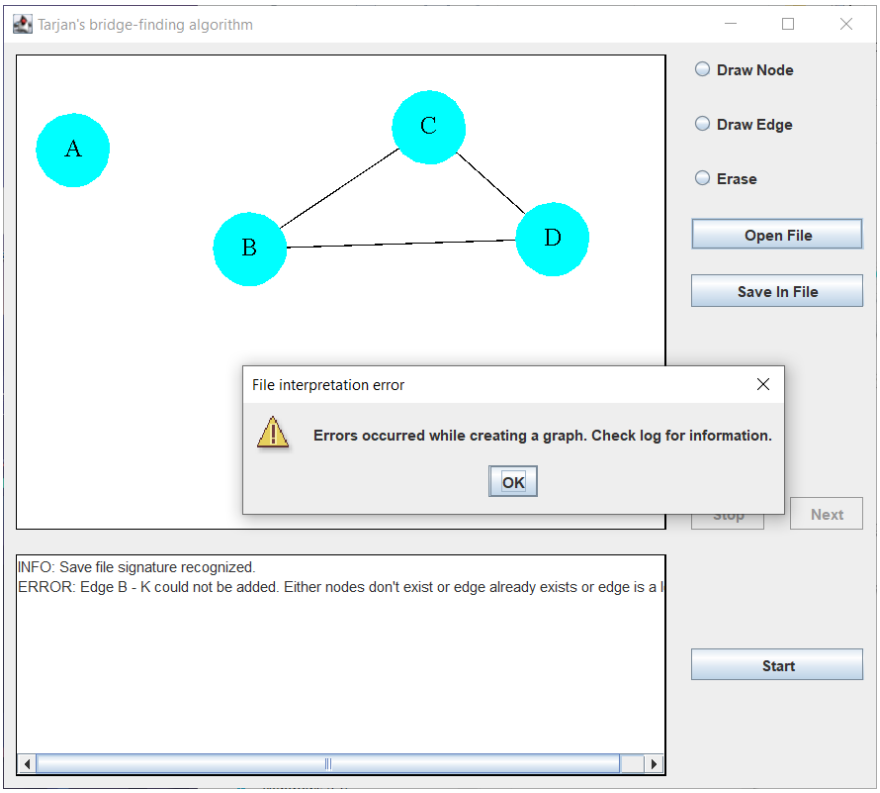

Рисунок 14 – Попытка загрузки графа из файла с указанным «лишним» ребром