Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВОЙ ПРОЕКТ**

**Дисциплина:** Алгоритмы и структуры данных

**Тема:** Разработка GUI приложения для алгоритма обхода всех рёбер графа

Выполнил

студент гр. 3530903/80003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Васильев Р.В.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Глухих М.И.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Санкт-Петербург   
2019

**Оглавление**

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc25151727)

[**МЕТОД РЕШЕНИЯ** 4](#_Toc25151728)

[**ЛИСТИНГ ПРОГРАММЫ** 7](#_Toc25151729)

[**Main.java** 7](#_Toc25151730)

[**MainMenuController.java** 8](#_Toc25151731)

[**ActionController.java** 13](#_Toc25151732)

[**HelpAboutController.java** 18](#_Toc25151733)

[**ExceptionController.java** 20](#_Toc25151734)

[**Parsing.java** 21](#_Toc25151735)

[**NodeCircle.java** 23](#_Toc25151736)

[**AlgorithmFindRoute.java** 24](#_Toc25151737)

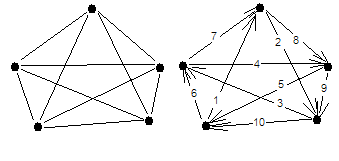
[**ОШИБКИ И ПРЕДУПРЕЖДЕНИЯ** 29](#_Toc25151738)

[**ТЕСТЫ** 30](#_Toc25151739)

[**СКРИНШОТЫ ПРОГРАММЫ** 32](#_Toc25151740)

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Задан неориентированный граф G=(V,E). Найти маршрут (если он существует), который проходит по каждому ребру графа один раз (фактически – обвести граф не отрывая руки)



GitHub репозиторий проекта: <https://github.com/killawetz/GraphSolverApp>

# **МЕТОД РЕШЕНИЯ**

В проекте использована концепция MVC (Model-View-Controller) для отделения бизнес-логики от визуализации. Код разделен на три пакета классов: model, controller, view

Содержимое пакета view:

* MainMenu.fxml – отвечает за графическое представление главного меню приложения. Содержит в себе тулбар, кнопки запуска приложения и выбора входного файла.
* Action.fxml – отвечает за графическое представление окна с отрисовкой графа и решения к нему. Содержит текстовое поле “Current progress:” и кнопку запускающую алгоритм решателя.
* HelpAbout.fxml – отвечает за графическое представление окна, которое появляется при переходе в Help -> About на тулбаре.

Cодержимое пакета controller:

* MainMenuController – контроллер, отвечающий за обработку событий происходящих в главном меню (нажатие на кнопки, открытие проводника).
* ActionController – контроллер, отвечающий за обработку событий происходящих в окне с решателем и отрисовкой. Помимо этого, этот класс отвечает за визуализацию входных данных и алгоритма решателя.
* HelpAboutController – контроллер, отвечающий за вставку текста на окне Help -> About.
* ExceptionController – контроллер, отвечающий за отображение предупреждений и ошибок.

Содержимое пакета model:

* Main – запуск приложения.
* Parsing – класс, для считывания данных из входного файла.
* NodeCircle – класс, который наследует Circle из javafx.
* AlgorithmFindRoute – алгоритм решателя.

Более подробная работа программы:

В главном меню пользователь с помощью тулбара может открыть проводник и с его помощью создать, либо редактировать уже готовый текстовый файл.

За это отвечает метод clickFileChooser().

В текстовом файле он задает входные данные согласно инструкции изложенной в Help -> About.

Для выбора рабочего текстового файла пользователю необходимо нажать на кнопку “Choose File”. В случае, если в текстовом файле находятся некорректные входные данные пользователю вылетит предупреждение об этом.

За это отвечает метод clickChoose(), который в свою очередь вызывает метод scanText() из класса Parsing, который проверяет входную строку на корректность входных данных, после чего записывает количество узлов графа и сами узлы в матрицу смежности.

После выбора необходимого текстового файла, пользователю необходимо нажать на кнопку “Start”, которая открывает следующую сцену. Так же в случае если пользователь предварительно не выбрал необходимый файл, вылезает предупреждение об этом.

За это отвечает метод clickStart().

На открытой новой сцене сразу инициализирована отрисовка графа, кнопка “Solve it” и надпись “Current progress”.

За отрисовку графа отвечает метод draw() класса ActionController, который рисует по следующему алгоритму:

Узлы располагаются как точки по окружности с заданным центром и радиусом.

Радиус константен. Центр окружности - центр сцены.

Шаг между двумя узлами равен углу.

360° делим на количество узлов - получаем шаг.

Узлы располагаются по формуле:

узел(центр сцены по X + синус(шаг) \* радиус, центр сцены по Y + косинус(шаг) \* радиус)

Далее, для запуска решателя, пользователю необходимо нажать на кнопку “Solve it”. Перед запуском решателя совершаются две проверки.

1. Проверка на связность. Если у графа есть хоть один узел не имеющий связей, то вылетает предупреждение о том, что маршрут у данного графа построить невозможно.
2. Так как алгоритм обхода графа, проходя по каждому ребру только по разу, давно известен как Эйлеров алгоритм, то проверка звучит как “проверить, является ли граф Эйлеровым”, т.е проверить не содержит ли граф больше двух нечетных узлов.

За это отвечают два метода allConnections() и isEuler() из класса AlgorithmFindRoute.

В случае, если все два условия удовлетворены, то запускается решатель.

Алгоритм построения Эйлерова цикла:

Чтобы построить Эйлеров путь, нужно запустить алгоритм из вершины с нечетной степенью.  
Алгоритм напоминает поиск в ширину. Главное отличие состоит в том, что пройденными помечаются не вершины, а ребра графа. Начиная со стартовой вершины v строим путь, добавляя на каждом шаге не пройденное еще ребро, смежное с текущей вершиной. Вершины пути накапливаются. Когда наступает такой момент, что для текущей вершины все инцидентные ей ребра уже пройдены, записываем накопленные вершины в ответ, пока не встретим вершину, которой инцидентны не пройденные еще ребра. Далее продолжаем обход по не посещенным ребрам.

За это отвечает метод printEulerTour() из класса AlgorithmFindRoute.

За отрисовку решателя отвечает метод printPassedRibs() из класса ActionController.

# **ЛИСТИНГ ПРОГРАММЫ**

## **Main.java**

package model;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application {

@Override

public void start(Stage primaryStage) throws Exception{

Parent root = FXMLLoader.load(getClass().getResource("/view/MainMenu.fxml"));

primaryStage.setTitle("Graph Solver");

primaryStage.setScene(new Scene(root, 600, 400));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

## **MainMenuController.java**

package controller;

import java.awt.\*;

import java.io.File;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Node;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.MenuBar;

import javafx.scene.control.MenuItem;

import javafx.stage.FileChooser;

import javafx.stage.Stage;

import model.Parsing;

public class MainMenuController {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button startButton;

@FXML

private MenuBar menuBar;

@FXML

private MenuItem menuBarOpenButton;

@FXML

private MenuItem menuBarQuitButton;

@FXML

private MenuItem helpAboutButton;

@FXML

private Button chooseFileButton;

private Desktop desktop = Desktop.getDesktop();

private File selectedFile;

@FXML

void initialize() {

}

public void openNewScene(String sceneName, Node node) { // метод для открытия следующего окна

node.getScene().getWindow().hide();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource(sceneName));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene(root));

stage.show();

}

public void closeCurrentScene(String sceneName, Node node) { // метод для закрытия

node.getScene().getWindow().hide();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource(sceneName));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene(root));

stage.close();

}

@FXML

private void clickStart(ActionEvent actionEvent) throws IOException {

if(Parsing.countOfNodes == 0) {

ExceptionController.ShowAlertCantStart();

return;

}

openNewScene("/view/Action.fxml", startButton);

}

@FXML

public void clickChoose(ActionEvent actionEvent) throws IOException {

fileExplorer();

try {

Parsing.scanText(selectedFile);

} catch (NullPointerException e) {

}

}

@FXML

public void closeScene(ActionEvent actionEvent) {

closeCurrentScene("MainMenu.fxml", startButton);

}

public void clickFileChooser(ActionEvent actionEvent) throws IOException {

fileExplorer();

if (selectedFile != null) {

openFile(selectedFile);

}

}

private void fileExplorer() throws IOException {

FileChooser fileChooser = new FileChooser();

fileChooser.getExtensionFilters().addAll(

new FileChooser.ExtensionFilter("Text Files", "\*.txt"));

selectedFile = fileChooser.showOpenDialog(menuBar.getScene().getWindow());

}

private void openFile(File file) {

try {

this.desktop.open(file);

} catch (IOException e) {

e.printStackTrace();

}

}

public void clickHelpAbout(ActionEvent actionEvent) throws IOException { // открытие вью при нажатии Help -> About

Parent root;

root = FXMLLoader.load(getClass().getResource("/view/HelpAbout.fxml"));

Stage stage = new Stage();

stage.setTitle("About");

stage.setScene(new Scene(root, 450, 450));

stage.show();

}

}

## **ActionController.java**

package controller;

import java.net.URL;

import java.util.ArrayList;

import java.util.ResourceBundle;

import javafx.animation.KeyFrame;

import javafx.animation.Timeline;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.layout.AnchorPane;

import javafx.scene.layout.StackPane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Line;

import javafx.scene.text.Text;

import javafx.scene.text.TextFlow;

import javafx.util.Duration;

import model.AlgorithmFindRoute;

import model.NodeCircle;

import model.Parsing;

import static java.lang.Math.PI;

import static java.lang.StrictMath.cos;

import static java.lang.StrictMath.sin;

public class ActionController {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

public AnchorPane anchoPane;

@FXML

public static Label currentProgressLabel;

@FXML

private Button solveItButton;

@FXML

private TextFlow textFlow;

@FXML

void initialize() {

draw(Parsing.countOfNodes);

createLine();

}

public void clickSolve(ActionEvent actionEvent) throws InterruptedException { // обработка нажатия кнопки Solve it для запуска решателя

AlgorithmFindRoute algo = new AlgorithmFindRoute(Parsing.countOfNodes, Parsing.matrix);

if(!algo.isEuler() || !algo.allConnections()) {

ExceptionController.showAlertCantFindRoute();

}

else {

algo.printEulerTour();

printPassedRibs();

}

}

Timeline timeline = new Timeline();

Duration timepoint = Duration.ZERO ;

Duration pause = Duration.seconds(1);

public void printPassedRibs() throws InterruptedException { // отрисовка по ребру с задержкой в секунду

for (int i = 0; i < listOfPassedRibs.size() ; i++) {

String bufferString = AlgorithmFindRoute.sourceDestination.get(i); // строка, которую будем выводить в "Current progress"

Line bufferLine = listOfPassedRibs.get(i); // пройденно ребро, которое будем отрисовывать на сцене

timepoint = timepoint.add(pause);

KeyFrame keyFrame = new KeyFrame(timepoint, e -> { // определяем набор объектов, которые будут рисоваться вдоль временной шкалы

anchoPane.getChildren().add(bufferLine);

textFlow.getChildren().add(new Text(bufferString + "\n "));

});

timeline.getKeyFrames().add(keyFrame);

}

timeline.play();

}

private static ArrayList<NodeCircle> listOfNodes = new ArrayList<>(); // список в котором упорядоченно хранятся все узлы.

public void setNodeCircle(double distanceX, double distanceY, String textOnNodes, int numberOfNode) { // метод задающий значения для узлов на сцене

Text text = new Text(textOnNodes);

text.setFill(Color.WHITE);

final StackPane stack = new StackPane();

NodeCircle nodeCircle = new NodeCircle(distanceX, distanceY, 15, numberOfNode);

listOfNodes.add(new NodeCircle(distanceX, distanceY, 15, numberOfNode));

stack.getChildren().addAll(nodeCircle, text);

anchoPane.getChildren().add(stack);

stack.setLayoutX(distanceX);

stack.setLayoutY(distanceY);

}

/\*метод отрисовывающий узлы по сцене

суть алгоритма:

узлы располагаются как точки по окружности с заданным центром и радиусом

Радиус константен. Центр окружности - центр сцены.

Шаг между двумя узлами равен углу.

360° делим на количество узлов - получаем шаг.

Узлы располгаются по такой формуле

узел(цетр сцены по X + синус(шаг) \* радиус, центр сцены по Y + косинус(шаг) \* радиус)

\*/

public void draw(int countOfNodes) {

int bufferCountOfNodes = 0;

int bufferNumberOfNode = 0;

System.out.println(bufferCountOfNodes);

int step = 360 / countOfNodes; // считаем, на какой угл должны отступать узлы друг от друга

int radius = 90;

double sceneCenterX = anchoPane.getPrefWidth() / 2;

double sceneCenterY = anchoPane.getPrefHeight() / 2;

for (double angle = 0; angle < 2 \* PI; angle += angleInRadian(step)) {

if(bufferCountOfNodes < Parsing.countOfNodes) {

setNodeCircle(sceneCenterX + sin(angle) \* radius, sceneCenterY + cos(angle) \* radius, String.valueOf(bufferCountOfNodes),bufferNumberOfNode); // расположение

bufferCountOfNodes++;

bufferNumberOfNode++;

}

}

}

static ArrayList<Line> listOfPassedRibs = new ArrayList<>(); // список пройденных ребер

public void createPassedRib(int vertex, int destination) { // метод для создания пройденного ребра

Line rib = new Line(listOfNodes.get(vertex).getCenterX() + 10,

listOfNodes.get(vertex).getCenterY() + 10,

listOfNodes.get(destination).getCenterX() + 10,

listOfNodes.get(destination).getCenterY() + 10);

rib.setStroke(Color.RED);

listOfPassedRibs.add(rib);

}

public void createLine() { // метод для создания ребра между узлами

for (int i = 0; i < Parsing.countOfNodes; i++) {

for (int j = 0; j < Parsing.countOfNodes; j++) {

if(Parsing.matrix[i][j]) {

Line line = new Line(listOfNodes.get(i).getCenterX() + 10,

listOfNodes.get(i).getCenterY() + 10,

listOfNodes.get(j).getCenterX() + 10,

listOfNodes.get(j).getCenterY() + 10);

anchoPane.getChildren().add(line);

}

}

}

}

private double angleInRadian(double grad) {

return PI\*grad/180;

}

}

## **HelpAboutController.java**

package controller;

import java.net.URL;

import java.util.ResourceBundle;

import javafx.fxml.FXML;

import javafx.scene.control.Label;

import javafx.scene.layout.VBox;

public class HelpAboutController {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private VBox vboxHelpAbout;

@FXML

void initialize() {

Label text = new Label();

text.setText("Инструкция по работе с приложением. \n" +

"\n" +

"File -> Open - открыть проводник с возможностью открытия окна нужного txt файла.\n" +

"File -> Quit - выход из приложения.\n" +

"\n" +

"Help -> About - пояснение к приложению \n" +

"\n" +

"Последовательность работы Graph Solver:\n" +

"\n" +

"Создаем документ формата .txt, заполяем его входными данными и сохраняем.\n" +

"File Chooser -> выбираем нужный txt файл\n" +

"Start - переход на новое окно с отрисовкой введеного графа\n" +

"\n" +

"Шаблон входных данных:\n" +

"n - кол-во узлов графа\n" +

"a b - ребра графа от узла a до b\n" +

"a c - ребра графа от узла a до с\n" +

"c b - ребра графа от узла с до b\n" +

"\n" +

"Пробелов после последней, перед передней цифрами строки быть не должно.\n" +

"Пустые строки, пустой файл, кириллица и латиница не считываются.\n" +

"Узлы задаются цифрами от 0 до n - 1, где n - количество узлов.\n" +

"Два узла указывающие ребро разделяются пробелом \" \".\n" +

"\n" +

"Пример \n" +

"4\n" +

"0 1\n" +

"1 2\n" +

"1 3\n" +

"2 0\n" +

"3 0\n");

vboxHelpAbout.getChildren().add(text);

}

}

## **ExceptionController.java**

package controller;

import javafx.scene.control.Alert;

public class ExceptionController {

public static void showAlertWithHeaderText() {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setHeaderText("Неверный формат ввода! Попробуйте еще раз!");

alert.setContentText("Файл ввода не может быть пустым" +

"и не может содержать ничего кроме цифр." +

" Ознакомление с форматом ввода Help -> About.");

alert.showAndWait();

}

public static void showAlertCantFindRoute() {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setHeaderText("Ошибка!");

alert.setContentText("У данного графа нельзя построить маршрут.");

alert.showAndWait();

}

public static void ShowAlertCantStart() {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Error");

alert.setHeaderText("Не выбран входной файл!");

alert.setContentText("Файл ввода не был указан. Нажмите File Chooser и выберите необходимый файл в директории.");

alert.showAndWait();

}

}

## **Parsing.java**

package model;

import controller.ExceptionController;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

public class Parsing {

public ArrayList<String> inputStrings = new ArrayList<>();

public static int countOfNodes; // количество узлов в матрице

public static boolean[][] matrix; // считанная с текстового файла матрица смежности

public static void scanText(File input) throws IOException { // метод для получения текста

Scanner scan = new Scanner(input);

if(input.length() == 0) { // если входной файл пустой, вывести алерт

ExceptionController.showAlertWithHeaderText();

return;

}

String nodes = scan.nextLine();

if (nodes.matches("\\d+")) { // проверка первой строчки входного файла, т.е количество вершин у графа

countOfNodes = Integer.parseInt(nodes);

if (countOfNodes <= 15) { // нельзя построить граф, если в нем больше 15 вершин

boolean[][] bufferMatrix = new boolean[countOfNodes][countOfNodes];

while (scan.hasNextLine()) {

String[] s = scan.nextLine().split(" ");

if(s.length > 2) { // если во входном файле на строчке больше двух символов, то вывести алерт

ExceptionController.showAlertWithHeaderText();

return;

}

try { // назначаем ребра между вершинами

bufferMatrix[Integer.parseInt(s[0])][Integer.parseInt(s[1])] = true;

bufferMatrix[Integer.parseInt(s[1])][Integer.parseInt(s[0])] = true;

}catch (ArrayIndexOutOfBoundsException | NumberFormatException e) {

ExceptionController.showAlertWithHeaderText();

return;

}

}

matrix = bufferMatrix;

}

else {

ExceptionController.showAlertWithHeaderText();

return;

}

}

else {

ExceptionController.showAlertWithHeaderText();

}

}

}

## **NodeCircle.java**

package model;

import javafx.scene.shape.Circle;

public class NodeCircle extends Circle { // класс для реализации объекта узла графа

int number; // номер узла

public NodeCircle(double centerX, double centerY, double radius, int n) {

super(centerX,centerY,radius); // вызов конструктора суперкласса

number = n;

}

public int getNumber() {

return number;

}

public double getX() {

return getCenterX();

}

public double get() {

return getCenterY();

}

}

## **AlgorithmFindRoute.java**

package model;

import controller.ActionController;

import java.util.ArrayList;

public class AlgorithmFindRoute { // класс алгоритм для обхода графа, за основу взят Эйлеров Алгоритм

public boolean[][] adjacencyMatrix; // матрица смежности

private int numberOfNodes; // количество узлов

public AlgorithmFindRoute (int numberOfNodes, boolean[][] adjacencyMatrix) { // конструктор

this.numberOfNodes = numberOfNodes;

this.adjacencyMatrix = new boolean[numberOfNodes] [numberOfNodes];

for (int sourceVertex = 0; sourceVertex <= numberOfNodes - 1; sourceVertex++) {

System.arraycopy(adjacencyMatrix[sourceVertex], 0, this.adjacencyMatrix[sourceVertex], 0, numberOfNodes - 1 + 1);

}

}

public boolean allConnections() { // метод, который проверяет, можно ли построить маршрут

boolean result = true;

for (int i = 0; i < adjacencyMatrix.length; i++) {

for (int j = 0; j < adjacencyMatrix.length ; j++) {

if(adjacencyMatrix[i][j]) {

result = adjacencyMatrix[i][j];

break;

}

result = adjacencyMatrix[i][j];

}

}

return result;

}

public int degree (int vertex) { // метод считает количество ребер исходящих из вершины, т.е четность узла

int degree = 0;

for (int destinationvertex = 0; destinationvertex <= numberOfNodes - 1 ; destinationvertex++) {

if (adjacencyMatrix[vertex][destinationvertex]

|| adjacencyMatrix[destinationvertex][vertex]) {

degree++;

}

}

return degree;

}

public boolean isEuler() { // проверка, является ли граф Эйлеровым: если в нем не больше двух нечетных узлов, то граф эйлеров

int countOfOdd = 0;

for (int node = 0; node <= numberOfNodes - 1; node++) {

if ((degree(node) % 2) != 0) {

countOfOdd++;

}

}

return !(countOfOdd > 2);

}

private int oddDegreeVertex() { // метод проверяющий существует ли вообще нечетная вершина, для того, чтобы начать с нее ход

int vertex = -1;

for (int node = 0; node <= numberOfNodes - 1; node++) {

if ((degree(node) % 2) != 0) {

vertex = node;

break;

}

}

return vertex;

}

public static ArrayList<String> sourceDestination = new ArrayList<>();

private ActionController actController = new ActionController();

private void printEulerTourUtil(int vertex) {

for (int destination = 0; destination <= numberOfNodes-1; destination++)

{

if(adjacencyMatrix[vertex][destination] && isVaildNextEdge(vertex, destination)) // если существует ребро, и следом за ним существует путь

{

System.out.println(" source : " + vertex + " destination " + destination);

// добавляет текст в массив под тем же индексом, что и пройденное ребро

sourceDestination.add(" source : " + String.valueOf(vertex) + " destination " + String.valueOf(destination));

actController.createPassedRib(vertex,destination); // делает ребро пройденым

removeEdge(vertex, destination);

printEulerTourUtil(destination);

}

}

}

public void printEulerTour () { // главный метод алгоритма, запускающий решатель

int vertex = 0;

if (oddDegreeVertex() != -1) {

vertex = oddDegreeVertex();

}

printEulerTourUtil(vertex);

}

private boolean isVaildNextEdge(int source, int destination) { // существует ли следующее ребро

int count = 0;

for (int vertex = 0; vertex <= numberOfNodes - 1; vertex++) { // считаем количество ребер у узла

if (adjacencyMatrix[source][vertex])

{

count++;

}

}

if (count == 1 ) {

return true;

}

int[] visited = new int[numberOfNodes]; // пройденные ребра

int count1 = DFSCount(source, visited);

removeEdge(source, destination);

for (int vertex = 0; vertex <= numberOfNodes - 1; vertex++)

{

visited[vertex] = 0;

}

int count2 = DFSCount(source, visited);

addEdge(source, destination);

return count1 <= count2;

}

private int DFSCount(int source, int[] visited) { // алгоритм поиска в глубину

visited[source] = 1;

int count = 1;

int destination = 0;

while (destination < numberOfNodes) {

if(adjacencyMatrix[source][destination] && visited[destination] == 0) {

count += DFSCount(destination, visited);

}

destination++;

}

return count;

}

public void removeEdge (int source, int destination) { // метод: убрать ребро

adjacencyMatrix[source][destination] = false;

adjacencyMatrix[destination][source] = false;

}

public void addEdge (int source, int destination) { // метод: добавить ребро

adjacencyMatrix[source][destination] = true;

adjacencyMatrix[destination][source] = true;

}

}

# **ОШИБКИ И ПРЕДУПРЕЖДЕНИЯ**

Все “ошибки” являются алертами из класса ExceptionController

|  |  |
| --- | --- |
| Код ошибки | Описание |
| 1 | showAlertWithHeaderText; Неверный формат ввода. |
| 2 | showAlertCantFindRoute; Нельзя построить маршрут. |
| 3 | ShowAlertCantStart; Не выбран входной файл. |

# **ТЕСТЫ**

Все функции из класса GraphSolverTest тестируют методы класса AlgorithmFindRoute.

import model.AlgorithmFindRoute;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class GraphSolverTest {

private boolean[][] matrixBad = {{false, true, true, false}, // несвязный граф

{true, false, false, false},

{true, false, false, false},

{false, false, false, false}};

private boolean[][] matrixGood = {{false, true, true, false}, // связный граф

{true, false, false, true},

{true, false, false, true},

{false, true, true, false}};

AlgorithmFindRoute algoBad = new AlgorithmFindRoute(matrixBad.length, matrixBad);

AlgorithmFindRoute algoGood = new AlgorithmFindRoute(matrixGood.length, matrixGood);

@Test

void testPossibilityOfPassing() { // тест возможно ли вообще построить маршрут

assertEquals(algoBad.allConnections(), false);

assertEquals(algoGood.allConnections(), true);

}

@Test

void testIsEuler() { // проверка графа на Эйлеровость

assertEquals(algoGood.isEuler(), true);

} // проверка графа на эйлеровость

@Test

void testDegree() { // тест метода подсчета ребер исходящих из узла

assertEquals(algoGood.degree(1), 2);

assertEquals(algoGood.degree(0), 2);

}

@Test

void testAddEdge() { // добавить ребро в граф который нельзя обойти

algoBad.addEdge(3,1);

assertEquals(algoBad.adjacencyMatrix[3][1], true);

}

@Test

void testRemoveEdge() { // удалить раннее добавленное ребро

algoBad.removeEdge(3,1);

assertEquals(algoBad.adjacencyMatrix[3][1], false);

}

}

# **СКРИНШОТЫ ПРОГРАММЫ**

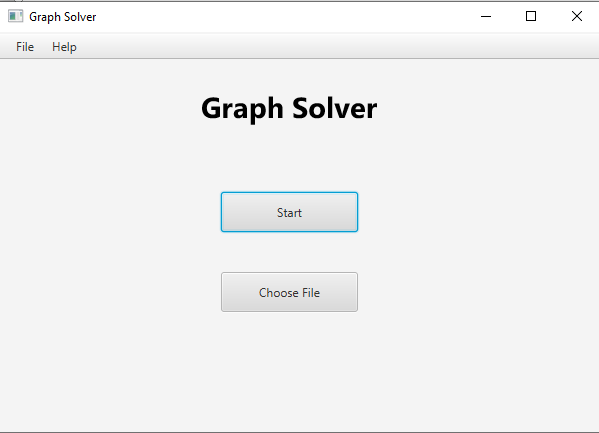


Рис.1 Главное меню

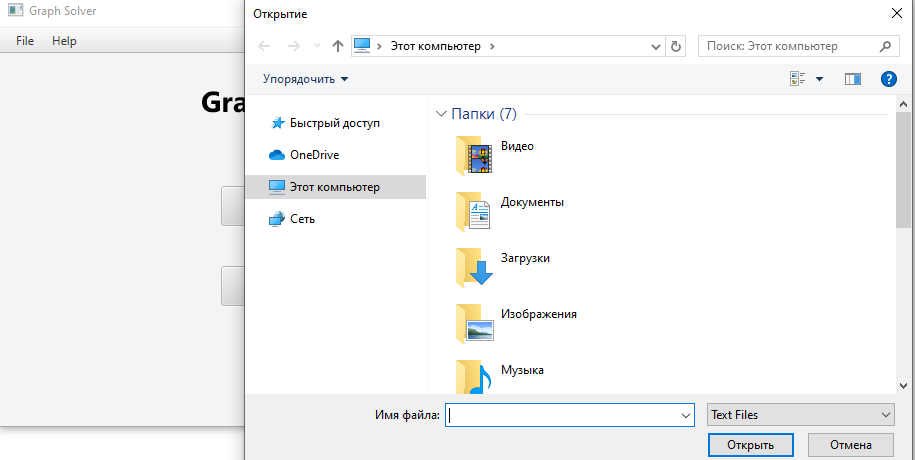


Рис. 2 Проводник

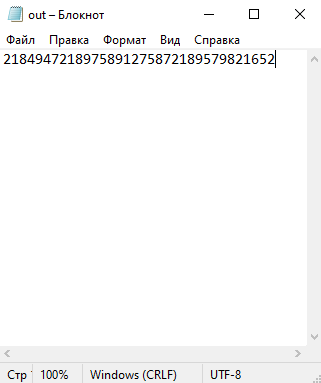


Рис. 3.1 Случайный набор чисел во входном файле

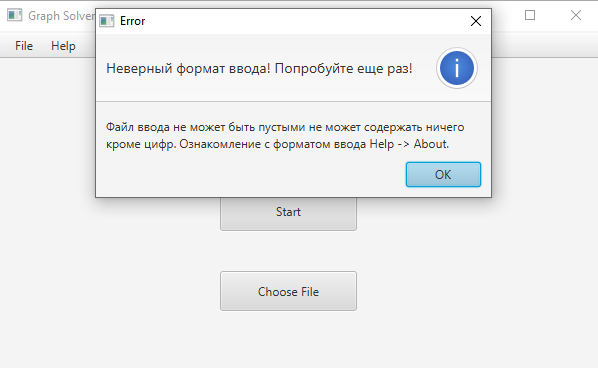


Рис. 3.2 Ошибка из-за неверного формата ввода

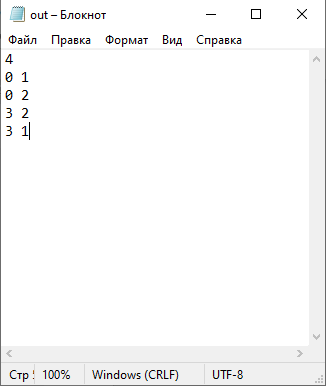


Рис. 4.1 Допустимые входные данные

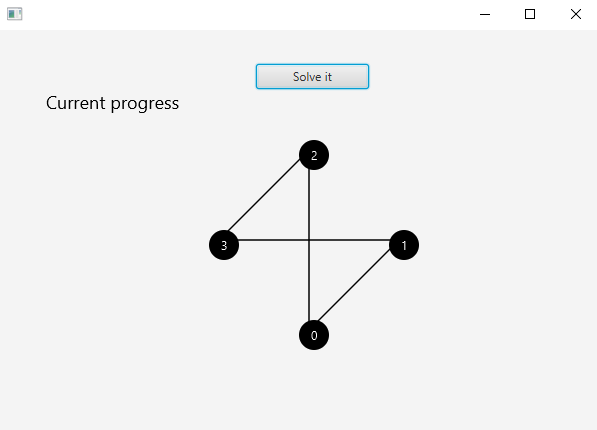


Рис. 4.2 Сцена с отрисованным графом по заданным в 4.1 входным данным

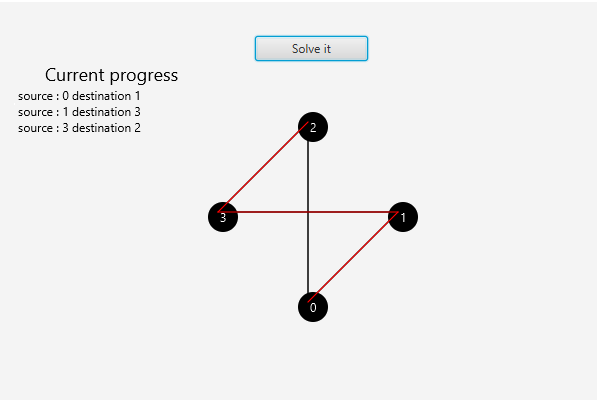


Рис. 5 Работа решателя

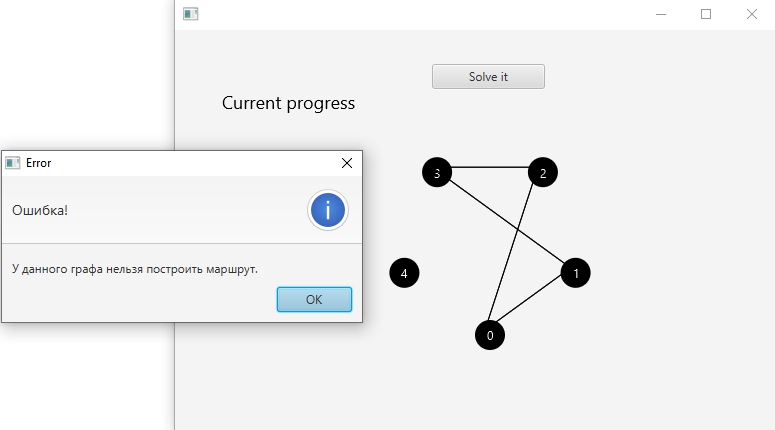


Рис. 6 Ошибка о невозможности построения маршрута.