Государственное образовательное учреждение

высшего профессионального образования

«Уфимский Государственный Авиационный Технический Университет»

Кафедра «Вычислительной математики и кибернетики»

Пояснительная записка

к курсовому проекту

по дисциплине: Дискретная математика

тема: Диаметр, радиус и центр графа

Проверил

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнил

студент гр. ПРО-101в \_\_\_\_\_\_\_\_\_\_Доронин С.Г.

Уфа 2012

# Содержание

Введение

1. Основные понятия теории графов
2. Область применения алгоритма
3. Теория по алгоритму
4. Задача с решением
5. Написание алгоритма на Java
6. Текст программы

Заключение

Список литературы

СОДЕРЖАНИЕ

[Содержание 2](#_Toc327969521)

[Введение 4](#_Toc327969522)

[1 Основные понятия теории графов 5](#_Toc327969523)

[2 Область применения алгоритма 6](#_Toc327969524)

[3 Теория по алгоритму 7](#_Toc327969525)

[4 Задача с решением 8](#_Toc327969526)

[5 Написание алгоритма на Java 9](#_Toc327969527)

[6 Текст программы 11](#_Toc327969528)

[Заключение 20](#_Toc327969529)

[Список литературы 21](#_Toc327969530)

# Введение

Объектом исследования является диаметр, радиус и центр графа (алгоритм нахождения диаметра, радиуса и центра в графе).

Цель работы – написание программы на языке Java, которая бы при заданных пользователем параметрах графа, находила бы его (графа) диаметр, радиус и центр.

# 1 Основные понятия теории графов

Теория графов — раздел дискретной математики, изучающий свойства графов. В общем смысле граф представляется как множество вершин (узлов), соединённых рёбрами. В строгом определении графом называется такая пара множеств G=(V,E), где V есть подмножество любого счётного множества, а E — подмножество V×V.

Теория графов находит применение, например, в геоинформационных системах (ГИС). Существующие или вновь проектируемые дома, сооружения, кварталы и т. п. рассматриваются как вершины, а соединяющие их дороги, инженерные сети, линии электропередачи и т. п. — как рёбра. Применение различных вычислений, производимых на таком графе, позволяет, например, найти кратчайший объездной путь или ближайший продуктовый магазин, спланировать оптимальный маршрут.

Теория графов содержит большое количество нерешённых проблем и пока не доказанных гипотез.

При изображении графов чаще всего используется следующая система обозначений: каждой вершине сопоставляется точка на плоскости, и если между вершинами существует ребро, то соответствующие точки соединяются отрезком. В случае ориентированного графа отрезки заменяют стрелками.

Не следует путать изображение графа с собственно графом (абстрактной структурой), поскольку одному графу можно сопоставить не одно графическое представление. Изображение призвано лишь показать, какие пары вершин соединены рёбрами, а какие — нет. Часто на практике бывает трудно ответить на вопрос, являются ли два изображения моделями одного и того же графа или нет. В зависимости от задачи, одни изображения могут давать более наглядную картину, чем другие.

# 2 Область применения алгоритма

Теория графов находит свое применение во многих отраслях таких как:

* в химии (для описания структур, путей сложных реакций, правило фаз также может быть интерпретировано как задача теории графов); компьютерная химия — сравнительно молодая область химии, основанная на применении теории графов. Теория графов представляет собой математическую основу хемоинформатики. Теория графов позволяет точно определить число теоретически возможных изомеров у углеводородов и других органических соединений;
* в информатике и программировании (граф-схема алгоритма);
* в коммуникационных и транспортных системах. В частности, для маршрутизации данных в Интернете;
* в экономике;
* в логистике;
* в схемотехнике (топология межсоединений элементов на печатной плате или микросхеме представляет собой граф или гиперграф).

# 3 Теория по алгоритму

Диаметром связного графа называется максимально возможное расстояние между двумя его вершинами.

Центром графа называется такая вершина, что максимальное расстояние между ней и любой другой вершиной является наименьшим из всех возможных; это расстояние называется радиусом графа.

Чтобы определить центры, радиус, диаметр графа G, найдем матрицу D(G) расстояний между вершинами графа, элементами dij которой будут расстояния между вершинами vi и vj. Для этого воспользуемся графическим представлением графа.

# 4 Задача с решением

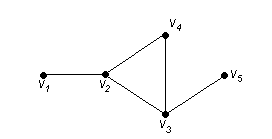


Рис.

Для графа G, изображенного на рРис. 1, найти радиус, диаметр и центры.

Решение.

Чтобы определить центры, радиус, диаметр графа G, найдем матрицу D(G) расстояний между вершинами графа, элементами dij которой будут расстояния между вершинами vi и vj. Для этого воспользуемся графическим представлением графа. Заметим, что матрица D(G) симметрична относительно главной диагонали.

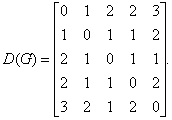


Рис.

С помощью полученной матрицы для каждой вершины графа G определим наибольшее удаление из выражения: для i, j = 1, 2, …, 5. В результате получаем: r(v1) = 3, r(v2) = 2, r(v3) = 2, r(v4) = 2, r(v5) = 3. Минимальное из полученных чисел является радиусом графа G, максимальное – диаметром графа G. Значит, R(G) = 2 и D(G) = 3, центрами являются вершины v2, v3, v4.

# 5 Написание алгоритма на Java

Функция «task()» получает все вершины графа, у каждой вершины получает всего ее грани. Из всех граней, каждой вершины, находит грань с максимальным размером с помощью функции «getMaxEdge(mxCell cell)» и добавляет найденную грань в массив.

Далее в массиве находится грани с максимальным и минимальным размером. Максимальный размер – это диаметр, он показывается на изображении графа другим цветом с помощью функции «showDiametr(mxCell diametr)».

Минимальный размер – это радиус графа, вершины содержащие грань с минимальным размером, являются центрами графа. Они выводятся в цикле перебором всех вершины и сравнением размера и показываются на изображении графа с помощью функции «showRadius(mxCell radius)».

/\*\*

\* Находит расстояния в графе: диаметр, центр, радиус графа

\*/

private void task() {

mxCell diametr, radius;

Set<mxCell> maxEdgeList = new HashSet<mxCell>();

final Object[] vertices = graph.getChildVertices(parent);

if (vertices.length > 0) {

System.out.println("\n\n========= Vertices =========");

for (Object vertice : vertices) {

mxCell cell = (mxCell) vertice;

mxCell maxEdge = getMaxEdge(cell);

if (maxEdge != null) {

maxEdgeList.add(maxEdge);

}

}

}

if (maxEdgeList.size() > 0) {

diametr = maxEdgeList.iterator().next();

radius = diametr;

for (mxCell edge : maxEdgeList) {

if (getDist(edge).compareTo(getDist(diametr)) > 0) {

diametr = edge;

}

if (getDist(edge).compareTo(getDist(radius)) < 0) {

radius = edge;

}

}

System.out.println("=============================");

showDiametr(diametr);

System.out.println("Diametr: " + getDist(diametr) + "; " + diametr);

for (mxCell edge : maxEdgeList) {

if (getDist(edge).compareTo(getDist(radius)) == 0) {

showRadius(edge);

System.out.println("Radius: " + getDist(edge) + "; " + edge);

}

}

System.out.println("=============================");

}

}

/\*\*

\* Показывает диаметр на изображении графа

\* @param diametr диаметр

\*/

private void showDiametr(mxCell diametr) {

graph.setCellStyles(mxConstants.STYLE\_STROKECOLOR, mxUtils.hexString(Const.STROKECOLOR\_DIAMETR), new Object[]{diametr});

}

/\*\*

\* Показывает радиус на изображении графа

\* @param radius радиус

\*/

private void showRadius(mxCell radius) {

graph.setCellStyles(mxConstants.STYLE\_STROKECOLOR, mxUtils.hexString(Const.STROKECOLOR\_RADIUS), new Object[]{radius});

if (radius.getSource() != null) {

graph.setCellStyles(mxConstants.STYLE\_STROKECOLOR, mxUtils.hexString(Const.STROKECOLOR\_RADIUS), new Object[]{radius.getSource()});

graph.setCellStyles(mxConstants.STYLE\_FILLCOLOR, mxUtils.hexString(Const.FILLCOLOR\_RADIUS), new Object[]{radius.getSource()});

}

}

/\*\*

\* Находит максимальную по длине грань в передаваемой вершине

\* @param cell Вершина

\* @return Максимальная по длине грань

\*/

private mxCell getMaxEdge(mxCell cell) {

if (cell.getEdgeCount() > 0) {

mxCell maxEdge = (mxCell) cell.getEdgeAt(0);

if (getDist(maxEdge) != null) {

for (int i=0; i<cell.getEdgeCount(); i++) {

final mxCell edge = (mxCell) cell.getEdgeAt(i);

final Double distEdge = getDist(edge);

final Double distMaxEdge = getDist(maxEdge);

if (distEdge!=null && distEdge.compareTo(distMaxEdge) > 0) {

maxEdge = edge;

}

}

System.out.println(cell.getValue() + ": maxEdge=" + getDist(maxEdge, 7));

return maxEdge;

}

}

return null;

}

# 6 Текст программы

Запускающий класс

package org.dyndns.phpusr.graph;

import com.mxgraph.util.mxResources;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import javax.swing.\*;

import java.awt.\*;

/\*\*

\* @author phpusr

\* Date: 27.05.12

\* Time: 16:04

\*/

public class Main {

public static void main(String[] args) {

Logger logger = LoggerFactory.getLogger(Main.class);

logger.debug("Start application.");

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

}

catch (Exception e1) {

e1.printStackTrace();

}

GraphEditor frame = new GraphEditor();

frame.showFrame();

}

}

class GraphEditor extends JFrame {

/\*\*

\* Adds required resources for i18n

\*/

static {

try {

mxResources.add("org/dyndns/phpusr/graph/editor");

}

catch (Exception e) {

// ignore

}

}

public GraphEditor() throws HeadlessException {

super("JGraph");

setContentPane(GraphForm.getInstance(new GraphUtil(this)));

setSize(1000, 600);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public void showFrame() {

setVisible(true);

}

}

Класс формы, для отображения графа

package org.dyndns.phpusr.graph;

import com.mxgraph.examples.swing.editor.DefaultFileFilter;

import com.mxgraph.util.mxResources;

import javax.swing.\*;

import javax.swing.filechooser.FileFilter;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

/\*\*

\* @author phpusr

\* Date: 27.05.12

\* Time: 14:42

\*/

public class GraphForm {

private static GraphForm INSTANCE;

private JButton btnAdd;

private JPanel pnlMain;

private JPanel pnlGraph;

private JButton btnEncode;

private JButton btnExit;

private JButton btnSave;

private JButton btnOpen;

private JButton btnNew;

private JButton btnDelete;

private GraphUtil util;

public GraphForm(GraphUtil graphUtil) {

this.util = graphUtil;

localizeForm();

btnAdd.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

util.addVertex();

}

});

btnEncode.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

util.getEncodeGraph();

}

});

btnExit.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

util.exit();

}

});

btnSave.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

JFileChooser fc = new JFileChooser();

fc.addChoosableFileFilter(new DefaultFileFilter(Const.EXT\_DEF,

Const.CHOOSE\_FILE\_FILTER\_DESCRIP));

int rc = fc.showDialog(null, mxResources.get("saveFile"));

if (rc == JFileChooser.APPROVE\_OPTION) {

String filename = fc.getSelectedFile().getAbsolutePath();

FileFilter selectedFilter = fc.getFileFilter();

//Добавление расширения файлу, если нет

if (selectedFilter instanceof DefaultFileFilter) {

String ext = ((DefaultFileFilter) selectedFilter)

.getExtension();

if (!filename.toLowerCase().endsWith(ext)) {

filename += ext;

}

}

//Проверка на существование файла

if (new File(filename).exists()

&& JOptionPane.showConfirmDialog(util.getGraphComponent(),

mxResources.get("overwriteExistingFile")) != JOptionPane.YES\_OPTION) {

return;

}

util.saveToFile(filename);

}

} catch (Throwable ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(util.getGraphComponent(),

ex.toString(), mxResources.get("error"),

JOptionPane.ERROR\_MESSAGE);

}

}

});

btnOpen.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JFileChooser fc = new JFileChooser();

fc.addChoosableFileFilter(new DefaultFileFilter(Const.EXT\_DEF,

Const.CHOOSE\_FILE\_FILTER\_DESCRIP));

int rc = fc.showDialog(null, mxResources.get("openFile"));

if (rc == JFileChooser.APPROVE\_OPTION) {

try {

File file = fc.getSelectedFile();

util.openFile(file);

} catch (IOException ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(

util.getGraphComponent(),

ex.toString(),

mxResources.get("error"),

JOptionPane.ERROR\_MESSAGE);

}

}

}

});

btnNew.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (JOptionPane.showConfirmDialog(util.getGraphComponent(),

mxResources.get("loseChanges")) == JOptionPane.YES\_OPTION) {

util.clear();

}

}

});

btnDelete.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

util.deleteCell();

}

});

}

/\*\*

\* Локализация компонентов формы

\*/

private void localizeForm() {

btnNew.setText(mxResources.get("new"));

btnOpen.setText(mxResources.get("openFile"));

btnSave.setText(mxResources.get("saveFile"));

btnAdd.setText(mxResources.get("add"));

btnDelete.setText(mxResources.get("delete"));

btnEncode.setText(mxResources.get("encode"));

btnExit.setText(mxResources.get("exit"));

}

/\*\*

\* Создание или получение экземпляра данного класса

\* @param util Утилита работа с библиотекой графа

\* @return Экземпляр данного класса

\*/

public synchronized static JPanel getInstance(GraphUtil util){

if ( INSTANCE == null ) {

INSTANCE = new GraphForm(util);

}

return INSTANCE.pnlMain;

}

/\*\*

\* Ручное создание компонентов формы

\*/

private void createUIComponents() {

pnlGraph = new JPanel(new BorderLayout());

pnlGraph.setSize(Const.FRAME\_WIDTH, Const.FRAME\_HEIGHT);

pnlGraph.add(util.getGraphComponent());

}

}

Класс работающий с библиотекой графа (jgraph)

package org.dyndns.phpusr.graph;

import com.mxgraph.io.gd.mxGdDocument;

import com.mxgraph.io.mxGdCodec;

import com.mxgraph.model.mxCell;

import com.mxgraph.model.mxGeometry;

import com.mxgraph.model.mxGraphModel;

import com.mxgraph.swing.mxGraphComponent;

import com.mxgraph.swing.util.mxGraphActions;

import com.mxgraph.util.\*;

import com.mxgraph.view.mxGraph;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.io.File;

import java.io.IOException;

import java.util.HashSet;

import java.util.Set;

/\*\*

\*

\* @author phpusr

\* Date: 27.05.12

\* Time: 17:57

\*/

public class GraphUtil {

private final mxGraph graph;

private Object parent;

private final mxGraphComponent graphComponent;

private final JFrame frame;

private int countVertex = 0;

private final Logger logger;

public GraphUtil(GraphEditor frame) {

logger = LoggerFactory.getLogger(GraphUtil.class);

this.frame = frame;

graph = new mxGraph();

customGraph(graph);

parent = graph.getDefaultParent();

graphComponent = new mxGraphComponent(graph);

graph.getModel().addListener(mxEvent.CHANGE, new mxEventSource.mxIEventListener() {

public void invoke(Object sender, mxEventObject evt) {

logger.debug("CHANGE");

onChange();

}

});

graph.addListener(mxEvent.ADD\_CELLS, new mxEventSource.mxIEventListener() {

public void invoke(Object sender, mxEventObject evt) {

logger.debug("ADD\_CELLS");

changeEdgeTitles();

resetStyleCells((Object[]) evt.getProperty("cells"));

graph.refresh();

}

});

getGraphComponent().getGraphControl().addMouseListener(new MouseAdapter() {

@Override

public void mousePressed(MouseEvent e) {

mouseReleased(e);

}

@Override

public void mouseReleased(MouseEvent e) {

if (e.isPopupTrigger()) {

logger.debug("Delete object");

deleteCell();

}

}

});

}

/\*\*

\* Включение и отключение опций графа

\* @param graph Граф

\*/

private void customGraph(mxGraph graph) {

graph.setAllowDanglingEdges(false);

}

/\*\*

\* Запускается при изменении графа

\*/

private void onChange() {

changeEdgeTitles();

//Сброс стиля для веришн

resetStyleCells(graph.getChildVertices(parent));

//Сброс стиля для граней

resetStyleCells(graph.getChildEdges(parent));

task();

graph.refresh();

}

/\*\*

\* Находит расстояния в графе: диаметр, центр, радиус графа

\*/

private void task() {

mxCell diametr, radius;

Set<mxCell> maxEdgeList = new HashSet<mxCell>();

final Object[] vertices = graph.getChildVertices(parent);

if (vertices.length > 0) {

System.out.println("\n\n========= Vertices =========");

for (Object vertice : vertices) {

mxCell cell = (mxCell) vertice;

mxCell maxEdge = getMaxEdge(cell);

if (maxEdge != null) {

maxEdgeList.add(maxEdge);

}

}

}

if (maxEdgeList.size() > 0) {

diametr = maxEdgeList.iterator().next();

radius = diametr;

for (mxCell edge : maxEdgeList) {

if (getDist(edge).compareTo(getDist(diametr)) > 0) {

diametr = edge;

}

if (getDist(edge).compareTo(getDist(radius)) < 0) {

radius = edge;

}

}

System.out.println("=============================");

showDiametr(diametr);

System.out.println("Diametr: " + getDist(diametr) + "; " + diametr);

for (mxCell edge : maxEdgeList) {

if (getDist(edge).compareTo(getDist(radius)) == 0) {

showRadius(edge);

System.out.println("Radius: " + getDist(edge) + "; " + edge);

}

}

System.out.println("=============================");

}

}

/\*\*

\* Показывает диаметр на изображении графа

\* @param diametr Диаметр

\*/

private void showDiametr(mxCell diametr) {

graph.setCellStyles(mxConstants.STYLE\_STROKECOLOR, mxUtils.hexString(Const.STROKECOLOR\_DIAMETR), new Object[]{diametr});

}

/\*\*

\* Показывает радиус на изображении графа

\* @param radius Радиус

\*/

private void showRadius(mxCell radius) {

graph.setCellStyles(mxConstants.STYLE\_STROKECOLOR, mxUtils.hexString(Const.STROKECOLOR\_RADIUS), new Object[]{radius});

if (radius.getSource() != null) {

graph.setCellStyles(mxConstants.STYLE\_STROKECOLOR, mxUtils.hexString(Const.STROKECOLOR\_RADIUS), new Object[]{radius.getSource()});

graph.setCellStyles(mxConstants.STYLE\_FILLCOLOR, mxUtils.hexString(Const.FILLCOLOR\_RADIUS), new Object[]{radius.getSource()});

}

}

/\*\*

\* Находит максимальную по длине грань в передаваемой вершине

\* @param cell Вершина

\* @return Максимальная по длине грань

\*/

private mxCell getMaxEdge(mxCell cell) {

if (cell.getEdgeCount() > 0) {

mxCell maxEdge = (mxCell) cell.getEdgeAt(0);

if (getDist(maxEdge) != null) {

for (int i=0; i<cell.getEdgeCount(); i++) {

final mxCell edge = (mxCell) cell.getEdgeAt(i);

final Double distEdge = getDist(edge);

final Double distMaxEdge = getDist(maxEdge);

if (distEdge!=null && distEdge.compareTo(distMaxEdge) > 0) {

maxEdge = edge;

}

}

System.out.println(cell.getValue() + ": maxEdge=" + getDist(maxEdge, 7));

return maxEdge;

}

}

return null;

}

/\*\*

\* Сброс стиля ячеек на стандартный

\* @param objects Масив вершин

\*/

private void resetStyleCells(Object[] objects) {

graph.setCellStyles(mxConstants.STYLE\_FONTSIZE, Const.FONT\_SIZE\_DEF, objects);

graph.setCellStyles(mxConstants.STYLE\_STROKECOLOR, mxUtils.hexString(Const.STROKECOLOR\_DEF), objects);

graph.setCellStyles(mxConstants.STYLE\_FILLCOLOR, mxUtils.hexString(Const.FILLCOLOR\_DEF), objects);

graph.setCellStyles(mxConstants.STYLE\_FONTCOLOR, mxUtils.hexString(Const.FONTCOLOR\_DEF), objects);

graph.setCellStyles(mxConstants.STYLE\_STROKEWIDTH, Const.STROKEWIDTH\_DEF, objects);

}

/\*\*

\* Показывает длину грани на ее заголовке

\*/

private void changeEdgeTitles() {

final Object[] edges = graph.getChildEdges(parent);

for (Object edge : edges) {

mxCell cell = (mxCell) edge;

cell.setValue(getDist(cell, 7));

}

}

/\*\*

\* Добавляет вершину на форму

\*/

public void addVertex() {

graph.getModel().beginUpdate();

try {

int x = (int) (Math.random() \* (Const.FRAME\_WIDTH - 2 \* Const.VERTEX\_WIDTH));

int y = (int) (Math.random() \* (Const.FRAME\_HEIGHT - 2 \* Const.VERTEX\_HEIGHT));

String title = Const.VERTEX\_NAME\_STD + " " + Integer.toString(++countVertex);

graph.insertVertex(parent, null, title, x, y, Const.VERTEX\_WIDTH, Const.VERTEX\_HEIGHT);

}

finally {

graph.getModel().endUpdate();

}

}

/\*\*

\* Удаляет вершину или грань графа

\*/

public void deleteCell() {

mxGraphActions.getDeleteAction().actionPerformed(new ActionEvent(getGraphComponent(), 0, ""));

}

/\*\*

\* Возвращает длину грани

\* @param cell Грань

\* @return Длина грани

\*/

private Double getDist(mxCell cell) {

logger.debug("getDist()");

if (cell.getSource() != null && cell.getTarget() != null) {

final mxGeometry sourcePoint = cell.getSource().getGeometry();

final mxGeometry targetPoint = cell.getTarget().getGeometry();

final double width = Math.pow(sourcePoint.getX() - targetPoint.getX(), 2);

final double height = Math.pow(sourcePoint.getY() - targetPoint.getY(), 2);

return Math.pow(width + height, (double)1/2);

} else {

return null;

}

}

/\*\*

\* Возвращает длину грани в виде строки, с указанным размером

\* @param cell Грань

\* @param length Длина возвращаемой строки

\* @return Длина грани

\*/

private String getDist(mxCell cell, int length) {

final Double dist = getDist(cell);

return dist != null && dist.toString().length() >= length ? dist.toString().substring(0, length-1) : "";

}

public mxGraphComponent getGraphComponent() {

return graphComponent;

}

/\*\*

\* Показывает текстовое представление графа

\*/

public void getEncodeGraph() {

String content = mxGdCodec.encode(graph).getDocumentString();

System.out.println("Encode:\n" + content);

}

/\*\*

\* Сохраняет граф в файл

\* @param filename Имя файла с графом

\* @throws IOException

\*/

public void saveToFile(String filename) throws IOException {

String content = mxGdCodec.encode(graph).getDocumentString();

mxUtils.writeFile(content, filename);

}

/\*\*

\* Открывает граф из файла

\* @param file Файл с графом

\* @throws IOException

\*/

public void openFile(File file) throws IOException {

mxGdDocument document = new mxGdDocument();

document.parse(mxUtils.readFile(file.getAbsolutePath()));

openGD(file, document);

countVertex = 0;

}

/\*\*

\* @throws IOException Ошибка

\*

\*/

private void openGD(File file, mxGdDocument document) {

// Replaces file extension with .mxe

String filename = file.getName();

filename = filename.substring(0, filename.length() - 4) + ".mxe";

if (new File(filename).exists()

&& JOptionPane.showConfirmDialog(getGraphComponent(),

mxResources.get("overwriteExistingFile")) != JOptionPane.YES\_OPTION) {

return;

}

((mxGraphModel) graph.getModel()).clear();

mxGdCodec.decode(document, graph);

parent = graph.getDefaultParent();

getGraphComponent().zoomAndCenter();

onChange();

}

/\*\*

\* Очистка графа

\*/

public void clear() {

((mxGraphModel) graph.getModel()).clear();

parent = graph.getDefaultParent();

countVertex = 0;

}

/\*\*

\* Выход

\*/

public void exit() {

frame.dispose();

}

}

# Заключение

В ходе проделанной работы был написан исходный код, реализующий алгоритм нахождения диаметра, радиуса и центра графа и программа реализующая алгоритм на примере поставленной задачи.

# Список литературы

1. ru.wikipedia.org [Электронный ресурс] // Свободная энциклопедия - <http://ru.wikipedia.org/w/index.php?title=%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2&printable=yes> Режим доступа – свободный
2. www.lvf2004.com [Электронный ресурс] // Электронный учебник по дискретной математике - <http://www.lvf2004.com/dop_t4r9part1.html> Режим доступа – свободный
3. www.jgraph.com [Электронный ресурс] // Сайт библиотеки построения графов jgraph - http://www.jgraph.com Режим доступа – свободный