ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

СОГЛАСОВАНО

Профессор департамента программной инженерии факультета компьютерных наук

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. Л. Чернышев

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия», канд. техн. наук, профессор ДПИ ФКН

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Программа для построения полинома, соответствующего случайному блужданию на геометрическом графе**

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729. 504900-01 12 01-1

**Исполнитель**

Студент группы БПИ183

\_\_\_\_\_\_\_\_\_\_\_ / М.И. Сердюков /

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Москва 2020**

**УТВЕРЖДЕНО**

RU.17701729. 504900-01 12 01-1

**Программа для построения полинома, соответствующего случайному блужданию на геометрическом графе**

**Текст программы**

RU.17701729. 504900-01 12 01-1

Листов 46

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

**Москва 2020**

# Содержание

[Содержание 3](#_Toc40792116)

[1. Текст программы 4](#_Toc40792117)

[2. Models 5](#_Toc40792118)

[3. ui 32](#_Toc40792119)

[4. Controllers 38](#_Toc40792120)

[5. Python 43](#_Toc40792121)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 46](#_Toc40792122)

# Текст программы

Программа состоит из 34 классов, и четырех главных пакетов, это: models, ui, controllers и python. Рассмотрим их исходный код.

# Models

**EditMode.java**

package com.bulumutka.polyconstr.models;

public enum EditMode {

NONE, ADD\_VERTEX, ADD\_EDGE, REMOVE, START\_VERTEX

}

**GraphEditor.java**

package com.bulumutka.polyconstr.models;

import com.bulumutka.polyconstr.models.graphlib.GraphBuilder;

import com.bulumutka.polyconstr.models.graphlib.MetricGraph;

import com.bulumutka.polyconstr.ui.\*;

import javafx.collections.FXCollections;

import javafx.collections.ListChangeListener;

import javafx.collections.ObservableList;

import java.util.ArrayList;

import java.util.Stack;

public class GraphEditor {

private final GraphBuilder builder = new GraphBuilder();

private final ObservableList<Drawable> components =

FXCollections.observableList(new ArrayList<>());

private final Stack<Vertex2D<Integer>> stack = new Stack<>();

private EditMode mode = EditMode.NONE;

private int vertexNumber = 0;

private boolean isEmpty = true;

private boolean hasStartVertex;

public void setEditMode(EditMode mode) {

System.out.println("setEditMode: " + mode);

this.mode = mode;

}

public boolean hasStartVertex() {

return hasStartVertex;

}

public GraphEditor(GraphCanvas canvas) {

canvas.setData(components);

canvas.setOnMouseClicked(event -> {

System.out.println("canvas was clicked at: " + event.getX() + " " + event.getY());

switch (mode) {

case ADD\_EDGE:

addEdge(event.getX(), event.getY());

break;

case ADD\_VERTEX:

addVertex(event.getX(), event.getY());

break;

case START\_VERTEX:

hasStartVertex = true;

var vertex = getVertex(event.getX(), event.getY());

if (vertex != null) {

builder.setRoot(vertex.getVertex());

for (var c : components) {

if (c instanceof Vertex2D) {

((Vertex2D) c).setIsStart(false);

}

}

vertex.setIsStart(true);

canvas.draw();

}

break;

default:

System.out.println("NONE action.");

break;

}

});

components.addListener((ListChangeListener<? super Drawable>) changeEvent -> {

System.out.println("Data components changed.");

canvas.draw();

});

}

public void reset() {

hasStartVertex = false;

isEmpty = true;

builder.reset();

mode = EditMode.NONE;

components.clear();

}

public boolean hasGraph() {

return !isEmpty;

}

private void addVertex(double x, double y) {

isEmpty = false;

builder.addVertex();

components.add(new Vertex2D<>(vertexNumber++, x, y));

}

private void addEdge(double x, double y) {

isEmpty = false;

Vertex2D<Integer> vertex = getVertex(x, y);

if (vertex == null) {

return;

}

stack.push(vertex);

if (stack.size() == 2) {

var source = stack.pop();

var target = stack.pop();

DialogWindow.edgeWeightDialog().ifPresent(time -> {

components.add(new Edge2D<>(source, target, time));

builder.addEdge(source.getVertex(), target.getVertex(), time);

});

}

}

private Vertex2D<Integer> getVertex(double x, double y) {

var element = components.filtered(d -> d instanceof Vertex2D && d.contains(x, y));

if (element.isEmpty()) {

return null;

}

return (Vertex2D<Integer>) element.get(0);

}

public MetricGraph getGraph() {

return builder.build();

}

}

**MathHelper.java**

package com.bulumutka.polyconstr.models;

import java.lang.Math;

import static java.lang.Math.sqrt;

public class MathHelper {

private MathHelper() {

}

public static double dist(double x1, double y1, double x2, double y2) {

double shiftX = Math.abs(x1 - x2);

double shiftY = Math.abs(y1 - y2);

return sqrt(shiftX \* shiftX + shiftY \* shiftY);

}

public static double length(double[] a, double[] b) {

return sqrt((a[0] - b[0]) \* (a[0] - b[0]) + (a[1] - b[1]) \* (a[1] - b[1]));

}

}

**PythonScript.java**

package com.bulumutka.polyconstr.models;

import java.io.\*;

import java.nio.Buffer;

public class PythonScript {

public static String start(String script) {

var builder = new ProcessBuilder();

builder.command("python3", script);

try {

var p = builder.start();

var output = new StringBuilder();

var reader = new BufferedReader(new InputStreamReader(p.getInputStream()));

String line;

while ((line = reader.readLine()) != null) {

output.append(line);

}

p.waitFor();

return output.toString();

} catch (IOException | InterruptedException exception) {

throw new RuntimeException("Can't excecute python script: " + exception.getMessage());

}

}

public static String startWithArgs(String script, String arg) {

var builder = new ProcessBuilder();

builder.command("python3", script);

try {

var p = builder.start();

var output = new StringBuilder();

var reader = new BufferedReader(new InputStreamReader(p.getInputStream()));

var writer = new BufferedWriter(new OutputStreamWriter(p.getOutputStream()));

writer.write(arg);

String line;

while ((line = reader.readLine()) != null) {

output.append(line);

}

p.waitFor();

return output.toString();

} catch (IOException | InterruptedException exception) {

throw new RuntimeException("Can't excecute python script: " + exception.getMessage());

}

}

}

**Repository.java**

package com.bulumutka.polyconstr.models;

public class Repository {

public static String formulas = "";

}

**SafeWriter.java**

package com.bulumutka.polyconstr.models;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.util.List;

public class SafeWriter {

public static void writeVector(List<Double> vector, String filePath) {

try {

var writer = new BufferedWriter(new FileWriter(filePath));

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

writer.write(String.valueOf(vector.get(i)));

writer.newLine();

}

writer.close();

} catch (IOException exception) {

throw new RuntimeException("Error while writing vector: " + exception.getMessage());

}

}

}

**Algorithms.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.DfsVisitor;

import com.bulumutka.polyconstr.models.graphlib.base.Edge;

import com.bulumutka.polyconstr.models.graphlib.base.Graph;

import java.util.HashSet;

import java.util.Set;

public class Algorithms {

private Algorithms() {

}

public static <E extends Edge<V>, V> Set<E> findReverseEdges(Graph<E, V> g, V originVertex) {

ReverseEdgeVisitor<E, V> visitor = new ReverseEdgeVisitor<E, V>();

depthFirstSearch(g, originVertex, visitor);

return visitor.getReverseEdges();

}

public static <E extends Edge<V>, V> boolean isConnected(Graph<E, V> g, V originVertex) {

var visitor = new ConnectedVisitor<E, V>(g.getVertexNumber());

depthFirstSearch(g, originVertex, visitor);

return visitor.isConnected();

}

public static <E extends Edge<V>, V> Set<E> findBridges(Graph<E, V> g, V originVertex) {

var visitor = new BridgesVisitor<E, V>();

depthFirstSearch(g, originVertex, visitor);

return visitor.getBridges();

}

public static <E extends Edge<V>, V> Set<E> findEdges(Graph<E, V> g) {

var visitor = new EdgesListVisitor<E, V>();

depthFirstSearch(g, g.getRoot(), visitor);

return visitor.getEdges();

}

public static <E extends Edge<V>, V> Set<E> findMarks(Graph<E, V> g, V originVertex,

V targetVertex) {

var visitor = new MarksVisitor<E, V>(targetVertex);

depthFirstSearch(g, originVertex, visitor);

return visitor.getMarks();

}

public static <E extends Edge<V>, V> void depthFirstSearch(Graph<E, V> g, V v,

DfsVisitor<E, V> visitor) {

Set<V> set = new HashSet<>();

dfs(g, v, visitor, set);

}

private static <E extends Edge<V>, V> void dfs(Graph<E, V> g, V vertex,

DfsVisitor<E, V> visitor, Set<V> used) {

used.add(vertex);

visitor.discoverVertex(vertex);

for (var edge : g.outgoingEdges(vertex)) {

var target = edge.getTarget();

visitor.examineEdge(edge);

if (!used.contains(target)) {

visitor.goEdge(edge);

dfs(g, target, visitor, used);

visitor.returnEdge(edge);

} else {

visitor.backEdge(edge);

}

}

}

}

**BridgesVisitor.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.Edge;

import com.bulumutka.polyconstr.models.graphlib.base.DfsVisitor;

import java.util.\*;

public class BridgesVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V> {

private int time = 0;

private final Set<E> bridges = new HashSet<>();

private final Map<V, V> parent = new HashMap<>();

private final Map<V, Integer> timeIn = new HashMap<>();

private final Map<V, Integer> fup = new HashMap<>();

@Override

public void discoverVertex(V vertex) {

timeIn.put(vertex, time);

fup.put(vertex, time);

++time;

}

@Override

public void examineEdge(E edge) {

if (edge.getTarget().equals(parent.get(edge.getSource()))) {

return;

}

boolean targetUsed = timeIn.containsKey(edge.getTarget());

if (targetUsed) {

fup.put(edge.getSource(), minTime(edge.getSource(), edge.getTarget()));

}

}

@Override

public void returnEdge(E edge) {

fup.put(edge.getSource(), Math.min(fup.get(edge.getSource()), fup.get(edge.getTarget())));

if (fup.get(edge.getTarget()) > timeIn.get(edge.getSource())) {

bridges.add(edge);

}

}

@Override

public void goEdge(E e) {

parent.put(e.getTarget(), e.getSource());

}

public Set<E> getBridges() {

return bridges;

}

private Integer minTime(V f, V s) {

var first = fup.get(f);

var second = timeIn.get(s);

return Math.min(first, second);

}

}

**CompressedGraph.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.ForAllVertices;

import java.util.\*;

/\*\*

\* Class that mapping graph to vector of numbers,

\* encapsulates all data needed for calculation function

\* of number endpoints of finite graph.

\*/

public class CompressedGraph {

private final MetricGraph graph;

private List<Double> vector = null;

private final List<Double> firstTerm = new ArrayList<>();

private final List<Double> secondTerm = new ArrayList<>();

public CompressedGraph(MetricGraph g) {

graph = g;

}

public List<Double> getVector() {

if (vector == null) {

buildCompression();

}

vector = new ArrayList<>();

vector.add(Double.valueOf(graph.getVertexNumber()));

vector.add(Double.valueOf(graph.getEdgesNumber()));

vector.addAll(firstTerm);

vector.addAll(secondTerm);

firstTerm.clear();

secondTerm.clear();

System.out.println("Vector built");

return vector;

}

private void buildCompression() {

SubGraphBruteForce brute = new SubGraphBruteForce(graph);

var g = brute.next();

int graphNumber = 0;

while (g != null) {

if (Algorithms.isConnected(g, g.getRoot())) {

++graphNumber;

proceedGraph(g);

}

g = brute.next();

}

firstTerm.add((double) graphNumber);

secondTerm.add((double) graphNumber);

Collections.reverse(firstTerm);

Collections.reverse(secondTerm);

}

private int vertexNumberTerm2 = 0;

private void proceedGraph(SubGraph g) {

Map<Integer, GraphEdge> bridgesEnds = new HashMap<>();

for (var edge : Algorithms.findBridges(g, g.getRoot())) {

bridgesEnds.put(edge.getTarget(), edge);

}

var reverseEdges = List.copyOf(Algorithms.findReverseEdges(g, g.getRoot()));

vertexNumberTerm2 = 0;

Algorithms.depthFirstSearch(g, g.getRoot(), (ForAllVertices<GraphEdge, Integer>) vertex -> {

if (g.outgoingEdges(vertex).size() > 1 && bridgesEnds.containsKey(vertex)) {

proceedFirstAndSecondTerm(g, vertex, findAllMarksSet(g, reverseEdges, vertex),

bridgesEnds.get(vertex));

++vertexNumberTerm2;

} else {

proceedFirstTerm(g, vertex, findAllMarksSet(g, reverseEdges, vertex));

}

});

firstTerm.add(Double.valueOf(g.getVertexNumber()));

secondTerm.add((double) vertexNumberTerm2);

var allEdges = Algorithms.findEdges(g);

for (var edge : allEdges) {

firstTerm.add(edge.time);

secondTerm.add(edge.time);

}

firstTerm.add((double) allEdges.size());

secondTerm.add((double) allEdges.size());

}

private void proceedFirstTerm(SubGraph g, Integer vertex, List<Set<GraphEdge>> allMarks) {

// Only first term

var sub = graph.outgoingEdges(vertex).size() - g.outgoingEdges(vertex).size();

for (var set : allMarks) {

var marksSum = 0.0;

for (var markedEdge : set) {

marksSum += markedEdge.time;

}

firstTerm.add(marksSum);

}

firstTerm.add((double) allMarks.size());

firstTerm.add((double) sub);

}

private void proceedFirstAndSecondTerm(SubGraph g, Integer vertex,

List<Set<GraphEdge>> allMarks, GraphEdge edge) {

var sub = graph.outgoingEdges(vertex).size() - g.outgoingEdges(vertex).size();

// First and second term

for (var set : allMarks) {

var marksSum1 = 0.0;

var marksSum2 = 0.0;

for (var markedEdge : set) {

marksSum1 += markedEdge.time;

if (markedEdge.id != edge.id) {

marksSum2 += markedEdge.time - edge.time;

}

}

firstTerm.add(marksSum1);

secondTerm.add(marksSum2);

}

secondTerm.add((double) allMarks.size());

secondTerm.add(edge.time);

firstTerm.add((double) allMarks.size());

firstTerm.add((double) sub);

}

private List<Set<GraphEdge>> findAllMarksSet(SubGraph g, List<GraphEdge> edges,

Integer targetVertex) {

var list = new ArrayList<Set<GraphEdge>>();

var bruteForce = new EdgesBruteForce(g, edges);

var subGraph = bruteForce.next();

while (subGraph != null) {

list.add(Algorithms.findMarks(subGraph, subGraph.getRoot(), targetVertex));

subGraph = bruteForce.next();

}

return list;

}

}

**ConnectedVisitor.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.DfsVisitor;

public class ConnectedVisitor<E, V> implements DfsVisitor<E, V> {

private final int vertexNumber;

private int visitedVertices = 0;

public ConnectedVisitor(int vertexNumber) {

this.vertexNumber = vertexNumber;

}

@Override

public void discoverVertex(V vertex) {

visitedVertices++;

}

public boolean isConnected() {

return visitedVertices == vertexNumber;

}

}

**EdgesBruteForce.java**

package com.bulumutka.polyconstr.models.graphlib;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class EdgesBruteForce {

private static final int MAX\_EDGE\_NUMBER = 63;

private long mask = -1;

private final SubGraph graph;

private final long maxMask;

private final Map<Integer, Integer> permutation = new HashMap<>();

public EdgesBruteForce(SubGraph g, List<GraphEdge> edges) {

if (edges.size() > MAX\_EDGE\_NUMBER) {

throw new IllegalArgumentException("Graph should not has more than 63 edges.");

}

graph = g;

maxMask = 1L << edges.size();

for (var i = 0; i != edges.size(); ++i) {

permutation.put(edges.get(i).id / 2, i);

}

}

public SubGraph next() {

if (mask == maxMask - 1) {

return null;

}

++mask;

return graph.addFilter(edge -> {

if (permutation.containsKey(edge.id / 2)) {

return (mask & 1L << permutation.get(edge.id / 2)) != 0;

}

return true;

});

}

}

**EdgesListVisitor.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.DfsVisitor;

import com.bulumutka.polyconstr.models.graphlib.base.Edge;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

import java.util.Set;

public class EdgesListVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V> {

private final Map<V, V> parent = new HashMap<>();

private final Set<V> black = new HashSet<>();

private final Set<E> edges = new HashSet<>();

@Override

public void goEdge(E e) {

edges.add(e);

parent.put(e.getTarget(), e.getSource());

}

@Override

public void returnEdge(E e) {

black.add(e.getTarget());

}

@Override

public void backEdge(E e) {

if (black.contains(e.getTarget()) || parent.get(e.getSource()) == null ||

e.getTarget().equals(parent.get(e.getSource()))) {

return;

}

edges.add(e);

}

public Set<E> getEdges() {

return edges;

}

}

**FilteredIterable.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.DfsVisitor;

import com.bulumutka.polyconstr.models.graphlib.base.Edge;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

import java.util.Set;

public class EdgesListVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V> {

private final Map<V, V> parent = new HashMap<>();

private final Set<V> black = new HashSet<>();

private final Set<E> edges = new HashSet<>();

@Override

public void goEdge(E e) {

edges.add(e);

parent.put(e.getTarget(), e.getSource());

}

@Override

public void returnEdge(E e) {

black.add(e.getTarget());

}

@Override

public void backEdge(E e) {

if (black.contains(e.getTarget()) || parent.get(e.getSource()) == null ||

e.getTarget().equals(parent.get(e.getSource()))) {

return;

}

edges.add(e);

}

public Set<E> getEdges() {

return edges;

}

}

**GraphBuilder.java**

package com.bulumutka.polyconstr.models.graphlib;

import java.util.ArrayList;

import java.util.List;

public class GraphBuilder {

private int vertexNumber = 0;

private int currentEdgeId = 0;

private int root = 0;

private final List<GraphEdge> edges = new ArrayList<>();

public GraphBuilder() {

}

public GraphBuilder(int vertexNumber) {

addVertex(vertexNumber);

}

public void addEdge(int source, int target, double time) {

edges.add(new GraphEdge(currentEdgeId++, source, target, time));

edges.add(new GraphEdge(currentEdgeId++, target, source, time));

}

public void addVertex(int number) {

for (var i = 0; i != number; ++i) {

addVertex();

}

}

public void addVertex() {

vertexNumber++;

}

public void removeEdge(int source, int target) {

edges.removeIf(edge -> (edge.target == target && edge.source == source ||

edge.source == target && edge.target == source));

}

public void removeVertex(int vertex) {

edges.removeIf(edge -> (edge.target == vertex || edge.source == vertex));

for (var edge : edges) {

if (edge.source > vertex) {

--edge.source;

}

if (edge.target > vertex) {

--edge.target;

}

}

vertexNumber--;

}

public void setRoot(int vertex) {

root = vertex;

}

public MetricGraph build() {

for (int i = 0; i != edges.size(); ++i) {

edges.get(i).id = i;

}

ArrayList<ArrayList<GraphEdge>> adjacencyList = new ArrayList<>(vertexNumber);

for (var i = 0; i != vertexNumber; ++i) {

adjacencyList.add(new ArrayList<>());

}

for (var edge : edges) {

adjacencyList.get(edge.source).add(edges.get(edge.id));

}

return new MetricGraph(edges, adjacencyList, root);

}

public void reset() {

currentEdgeId = 0;

vertexNumber = 0;

root = 0;

edges.clear();

}

}

**GraphEdge.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.Edge;

public class GraphEdge implements Edge<Integer> {

public int id;

int source;

int target;

public double time;

public GraphEdge(int id, int source, int target, double time) {

this.id = id;

this.source = source;

this.target = target;

this.time = time;

}

@Override

public int hashCode() {

return id;

}

@Override

public boolean equals(Object obj) {

var other = (GraphEdge) obj;

return source == other.source && target == other.target;

}

@Override

public Integer getTarget() {

return target;

}

@Override

public Integer getSource() {

return source;

}

}

**MarksVisitor.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.Edge;

import com.bulumutka.polyconstr.models.graphlib.base.DfsVisitor;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

import java.util.Set;

public class MarksVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V> {

private final Map<V, E> parent = new HashMap<>();

private final Set<V> found = new HashSet<>();

private final Set<E> marks = new HashSet<>();

private final Set<V> black = new HashSet<>();

private final V target;

public MarksVisitor(V target) {

this.target = target;

}

public Set<E> getMarks() {

return marks;

}

@Override

public void discoverVertex(V v) {

if (v.equals(target)) {

found.add(v);

}

}

@Override

public void goEdge(E e) {

parent.put(e.getTarget(), e);

markEdge(e);

}

@Override

public void returnEdge(E e) {

black.add(e.getTarget());

if (found.contains(e.getTarget())) {

found.add(e.getSource());

} else {

markEdge(e);

}

}

@Override

public void backEdge(E e) {

if (black.contains(e.getTarget()) || parent.get(e.getSource()) == null ||

e.getTarget().equals(parent.get(e.getSource()).getSource())) {

return;

}

markEdge(e);

var current = parent.get(e.getSource());

while (current != null && !current.getTarget().equals(e.getTarget())) {

markEdge(current);

current = parent.get(current.getSource());

}

}

private void markEdge(E edge) {

if (!marks.remove(edge)) {

marks.add(edge);

}

}

}

**MetricGraph.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.AbstractGraph;

import java.util.ArrayList;

import java.util.List;

public class MetricGraph extends AbstractGraph<GraphEdge, Integer> {

private final List<GraphEdge> edges;

private final ArrayList<ArrayList<GraphEdge>> adjacencyList;

private final int rootVertex;

public MetricGraph(List<GraphEdge> edges, ArrayList<ArrayList<GraphEdge>> list, int root) {

this.edges = edges;

adjacencyList = list;

rootVertex = root;

}

@Override

public List<GraphEdge> outgoingEdges(Integer vertex) {

return adjacencyList.get(vertex);

}

@Override

public Integer getVertexNumber() {

return adjacencyList.size();

}

@Override

public Integer getEdgesNumber() {

return edges.size() / 2;

}

public Integer getRoot() {

return rootVertex;

}

public List<GraphEdge> getEdges() {

return edges;

}

}

**ReverseEdgeVisitor.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.DfsVisitor;

import com.bulumutka.polyconstr.models.graphlib.base.Edge;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

import java.util.Set;

public class ReverseEdgeVisitor<E extends Edge<V>, V> implements DfsVisitor<E, V> {

private final Map<V, V> parent = new HashMap<>();

private final Set<V> black = new HashSet<>();

private final Set<E> reverseEdges = new HashSet<>();

@Override

public void goEdge(E e) {

parent.put(e.getTarget(), e.getSource());

}

@Override

public void returnEdge(E e) {

black.add(e.getTarget());

}

@Override

public void backEdge(E e) {

if (black.contains(e.getTarget()) || parent.get(e.getSource()) == null ||

e.getTarget().equals(parent.get(e.getSource()))) {

return;

}

reverseEdges.add(e);

}

public Set<E> getReverseEdges() {

return reverseEdges;

}

}

**SubGraph.java**

package com.bulumutka.polyconstr.models.graphlib;

import com.bulumutka.polyconstr.models.graphlib.base.AbstractGraph;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.function.Predicate;

public class SubGraph extends AbstractGraph<GraphEdge, Integer> {

private final MetricGraph otherGraph;

private final Predicate<GraphEdge> predicate;

private int vertexNumber = -1;

private int edgesNumber = -1;

private List<GraphEdge> edges = null;

public SubGraph(MetricGraph otherGraph) {

this.otherGraph = otherGraph;

this.predicate = edge -> true;

}

public SubGraph(MetricGraph otherGraph, Predicate<GraphEdge> predicate) {

this.otherGraph = otherGraph;

this.predicate = predicate;

}

@Override

public Integer getRoot() {

return otherGraph.getRoot();

}

@Override

public FilteredIterable<GraphEdge> outgoingEdges(Integer vertex) {

return new FilteredIterable<>(otherGraph.outgoingEdges(vertex), predicate);

}

@Override

public Integer getVertexNumber() {

if (vertexNumber != -1) {

return vertexNumber;

}

if (getEdges().isEmpty()) {

vertexNumber = 1;

return vertexNumber;

}

Set<Integer> set = new HashSet<>();

for (var edge : getEdges()) {

set.add(edge.source);

set.add(edge.target);

}

vertexNumber = set.size();

return vertexNumber;

}

@Override

public Integer getEdgesNumber() {

if (edgesNumber != -1) {

return edgesNumber;

}

edgesNumber = getEdges().size() / 2;

return edgesNumber;

}

public SubGraph addFilter(Predicate<GraphEdge> filter) {

return new SubGraph(otherGraph, predicate.and(filter));

}

public List<GraphEdge> getEdges() {

if (edges != null) {

return edges;

}

edges = new ArrayList<>();

for (var edge : otherGraph.getEdges()) {

if (predicate.test(edge)) {

edges.add(edge);

}

}

return edges;

}

}

**SubGraphBruteForce.java**

package com.bulumutka.polyconstr.models.graphlib;

public class SubGraphBruteForce {

private static final int MAX\_EDGE\_NUMBER = 63;

private long mask = -1;

private final MetricGraph graph;

private final long maxMask;

public SubGraphBruteForce(MetricGraph g) {

if (g.getEdgesNumber() > MAX\_EDGE\_NUMBER) {

throw new IllegalArgumentException("Graph should not has more than 63 edges.");

}

this.graph = g;

maxMask = 1L << g.getEdgesNumber();

}

public SubGraph next() {

if (mask == maxMask - 1) {

return null;

}

++mask;

return new SubGraph(graph, edge -> {

int bit = edge.id / 2;

return (mask & 1L << bit) != 0;

});

}

}

**AbstractGraph.java**

package com.bulumutka.polyconstr.models.graphlib;

public class SubGraphBruteForce {

private static final int MAX\_EDGE\_NUMBER = 63;

private long mask = -1;

private final MetricGraph graph;

private final long maxMask;

public SubGraphBruteForce(MetricGraph g) {

if (g.getEdgesNumber() > MAX\_EDGE\_NUMBER) {

throw new IllegalArgumentException("Graph should not has more than 63 edges.");

}

this.graph = g;

maxMask = 1L << g.getEdgesNumber();

}

public SubGraph next() {

if (mask == maxMask - 1) {

return null;

}

++mask;

return new SubGraph(graph, edge -> {

int bit = edge.id / 2;

return (mask & 1L << bit) != 0;

});

}

}

**DfsVisitor.java**

package com.bulumutka.polyconstr.models.graphlib;

public class SubGraphBruteForce {

private static final int MAX\_EDGE\_NUMBER = 63;

private long mask = -1;

private final MetricGraph graph;

private final long maxMask;

public SubGraphBruteForce(MetricGraph g) {

if (g.getEdgesNumber() > MAX\_EDGE\_NUMBER) {

throw new IllegalArgumentException("Graph should not has more than 63 edges.");

}

this.graph = g;

maxMask = 1L << g.getEdgesNumber();

}

public SubGraph next() {

if (mask == maxMask - 1) {

return null;

}

++mask;

return new SubGraph(graph, edge -> {

int bit = edge.id / 2;

return (mask & 1L << bit) != 0;

});

}

}

**Edge.java**

package com.bulumutka.polyconstr.models.graphlib;

public class SubGraphBruteForce {

private static final int MAX\_EDGE\_NUMBER = 63;

private long mask = -1;

private final MetricGraph graph;

private final long maxMask;

public SubGraphBruteForce(MetricGraph g) {

if (g.getEdgesNumber() > MAX\_EDGE\_NUMBER) {

throw new IllegalArgumentException("Graph should not has more than 63 edges.");

}

this.graph = g;

maxMask = 1L << g.getEdgesNumber();

}

public SubGraph next() {

if (mask == maxMask - 1) {

return null;

}

++mask;

return new SubGraph(graph, edge -> {

int bit = edge.id / 2;

return (mask & 1L << bit) != 0;

});

}

}

**ForAllVertices.java**

package com.bulumutka.polyconstr.models.graphlib;

public class SubGraphBruteForce {

private static final int MAX\_EDGE\_NUMBER = 63;

private long mask = -1;

private final MetricGraph graph;

private final long maxMask;

public SubGraphBruteForce(MetricGraph g) {

if (g.getEdgesNumber() > MAX\_EDGE\_NUMBER) {

throw new IllegalArgumentException("Graph should not has more than 63 edges.");

}

this.graph = g;

maxMask = 1L << g.getEdgesNumber();

}

public SubGraph next() {

if (mask == maxMask - 1) {

return null;

}

++mask;

return new SubGraph(graph, edge -> {

int bit = edge.id / 2;

return (mask & 1L << bit) != 0;

});

}

}

**Graph.java**

package com.bulumutka.polyconstr.models.graphlib;

public class SubGraphBruteForce {

private static final int MAX\_EDGE\_NUMBER = 63;

private long mask = -1;

private final MetricGraph graph;

private final long maxMask;

public SubGraphBruteForce(MetricGraph g) {

if (g.getEdgesNumber() > MAX\_EDGE\_NUMBER) {

throw new IllegalArgumentException("Graph should not has more than 63 edges.");

}

this.graph = g;

maxMask = 1L << g.getEdgesNumber();

}

public SubGraph next() {

if (mask == maxMask - 1) {

return null;

}

++mask;

return new SubGraph(graph, edge -> {

int bit = edge.id / 2;

return (mask & 1L << bit) != 0;

});

}

}

# ui

**DialogWindow.java**

package com.bulumutka.polyconstr.ui;

import javafx.scene.control.Alert;

import javafx.scene.control.TextInputDialog;

import java.util.Optional;

public class DialogWindow {

static public Optional<Double> edgeWeightDialog() {

TextInputDialog dialog = new TextInputDialog("");

dialog.setTitle("Edge parameters");

dialog.setHeaderText("Enter edge weight:");

dialog.setContentText("Weight:");

Optional<String> result = dialog.showAndWait();

if (result.isEmpty()) {

return Optional.empty();

}

try {

Double weight = Double.parseDouble(result.get());

return Optional.of(weight);

} catch (NumberFormatException ex) {

errorDialog("Wrong edge weight parameter. It should be real number.");

}

return Optional.empty();

}

static public void errorDialog(String msg) {

var dialog = new Alert(Alert.AlertType.ERROR);

dialog.setTitle("Error");

dialog.setHeaderText(msg);

dialog.show();

}

}

**Drawable.java**

package com.bulumutka.polyconstr.ui;

import javafx.scene.canvas.Canvas;

public interface Drawable {

void draw(Canvas canvas);

boolean contains(double x, double y);

}

**Edge2D.java**

package com.bulumutka.polyconstr.ui;

import com.bulumutka.polyconstr.models.MathHelper;

import javafx.scene.canvas.Canvas;

import javafx.scene.paint.Color;

import java.util.Arrays;

import java.util.List;

public class Edge2D<Vertex extends Vertex2D> implements Drawable {

public Vertex source;

public Vertex target;

public double time;

public Edge2D(Vertex source, Vertex target, double time) {

this.source = source;

this.target = target;

this.time = time;

}

@Override

public void draw(Canvas canvas) {

double x1 = source.getX();

double y1 = source.getY();

double x2 = target.getX();

double y2 = target.getY();

var g = canvas.getGraphicsContext2D();

g.setFill(Color.BLACK);

g.setStroke(Color.BLACK);

g.strokeLine(x1, y1, x2, y2);

g.strokeLine(x1, y1, x2, y2);

g.fillText(Double.toString(time), (x1 + x2) / 2 + 10, (y1 + y2) / 2 + 10);

}

@Override

public boolean contains(double x, double y) {

return false;

}

private List<Double> cutLine(double x1, double y1, double x2, double y2, double value) {

var s1 = x2 - x1;

var s2 = y2 - y1;

var dist = MathHelper.dist(x1, y1, x2, y2);

var n1 = s1 / Math.sqrt(dist);

var n2 = s2 / Math.sqrt(dist);

var s3 = n1 \* (dist - value);

var s4 = n2 \* (dist - value);

return Arrays.asList(x1, y1, x1 + s3, y1 + s4);

}

}

**GraphCanvas.java**

package com.bulumutka.polyconstr.ui;

import java.util.List;

public class GraphCanvas extends ResizableCanvas {

private List<Drawable> components;

public void setData(List<Drawable> data) {

components = data;

}

@Override

public void draw() {

super.draw();

System.out.println("Draw graph canvas.");

for (var component : components) {

component.draw(this);

}

for (var component : components) {

if (component instanceof Vertex2D) {

component.draw(this);

}

}

}

}

**ResizableCanvas.java**

package com.bulumutka.polyconstr.ui;

import javafx.scene.canvas.Canvas;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.paint.Color;

public class ResizableCanvas extends Canvas {

@Override

public boolean isResizable() {

return true;

}

@Override

public double prefHeight(double width) {

return minHeight(width);

}

@Override

public double minHeight(double width) {

return 64;

}

@Override

public double maxHeight(double width) {

return 1080;

}

@Override

public double minWidth(double height) {

return 0;

}

@Override

public double maxWidth(double height) {

return 1920;

}

@Override

public void resize(double width, double height) {

super.setWidth(width);

super.setHeight(height);

draw();

}

public void draw() {

double width = getWidth();

double height = getHeight();

GraphicsContext gc = getGraphicsContext2D();

gc.setFill(Color.web("#fffff6"));

gc.fillRect(0, 0, width, height);

}

}

**Vertex2D.java**

package com.bulumutka.polyconstr.ui;

import com.bulumutka.polyconstr.models.MathHelper;

import javafx.scene.canvas.Canvas;

import javafx.scene.paint.Color;

public class Vertex2D<Vertex> implements Drawable {

public static final double VERTEX\_RADIUS = 10;

private final Vertex vertex;

private final double x;

private final double y;

private boolean start = false;

public Vertex2D(Vertex vertex, double x, double y) {

this.vertex = vertex;

this.x = x;

this.y = y;

}

public double getX() {

return x;

}

public double getY() {

return y;

}

public Vertex getVertex() {

return vertex;

}

public void setIsStart(boolean isStart) {

start = isStart;

}

@Override

public void draw(Canvas canvas) {

double x = getX();

double y = getY();

var g = canvas.getGraphicsContext2D();

g.setStroke(Color.BLACK);

g.setFill(Color.WHITE);

if (start) {

g.setStroke(Color.BLUE);

}

g.fillOval(centerX(), centerY(), 2 \* VERTEX\_RADIUS, 2 \* VERTEX\_RADIUS);

g.strokeOval(centerX(), centerY(), 2 \* VERTEX\_RADIUS, 2 \* VERTEX\_RADIUS);

}

@Override

public boolean contains(double x, double y) {

return MathHelper.dist(getX(), getY(), x, y) < VERTEX\_RADIUS;

}

@Override

public String toString() {

return "Vertex2D{" + "vertex=" + vertex + ", x=" + x + ", y=" + y + '}';

}

public double centerX() {

return getX() - VERTEX\_RADIUS;

}

public double centerY() {

return getY() - VERTEX\_RADIUS;

}

}

# Controllers

**MainController.java**

package com.bulumutka.polyconstr.controllers;

import com.bulumutka.polyconstr.models.\*;

import com.bulumutka.polyconstr.models.graphlib.CompressedGraph;

import com.bulumutka.polyconstr.ui.DialogWindow;

import com.bulumutka.polyconstr.ui.GraphCanvas;

import com.sun.javafx.fxml.builder.JavaFXSceneBuilder;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.StackPane;

import javafx.stage.Stage;

import java.awt.\*;

import java.io.IOException;

import java.net.ResponseCache;

import java.net.URL;

import java.util.Objects;

import java.util.ResourceBundle;

import java.util.Stack;

public class MainController implements Initializable {

public StackPane drawArea;

public GraphCanvas canvas;

public GridPane mainPane;

public Button vertexButton;

public Button edgeButton;

public Button startVertexButton;

public Button clearButton;

public Button countButton;

private GraphEditor editor;

@Override

public void initialize(URL location, ResourceBundle resources) {

editor = new GraphEditor(canvas);

vertexButton.setOnAction(event -> {

System.out.println("Vertex button clicked.");

editor.setEditMode(EditMode.ADD\_VERTEX);

});

edgeButton.setOnAction(event -> {

System.out.println("Edge button clicked.");

editor.setEditMode(EditMode.ADD\_EDGE);

});

clearButton.setOnAction(event -> editor.reset());

startVertexButton.setOnAction(event -> {

System.out.println("Start button clicked.");

editor.setEditMode(EditMode.START\_VERTEX);

});

countButton.setOnAction(event -> {

if (!editor.hasGraph()) {

DialogWindow.errorDialog("Draw your graph first");

} else if (!editor.hasStartVertex()) {

DialogWindow.errorDialog("Choose start vertex");

} else {

var graph = editor.getGraph();

var c = new CompressedGraph(graph);

var vector = c.getVector();

SafeWriter.writeVector(vector, "data.txt");

Repository.formulas = PythonScript

.start("out/production/PolynomialConstruction/com/bulumutka/polyconstr" +

"/get\_formulas.py");

showCountResult();

}

});

}

private void showCountResult() {

try {

Parent parent = FXMLLoader.load(Objects.requireNonNull(

getClass().getClassLoader().getResource("views" + "/ShowResultView.fxml")));

Stage stage = new Stage();

stage.setScene(new Scene(parent));

stage.centerOnScreen();

stage.setTitle("Function's formulas");

stage.show();

} catch (IOException ex) {

throw new RuntimeException("No resources found " + ex.getMessage());

}

}

}

**ShowResultController.java**

package com.bulumutka.polyconstr.controllers;

import com.bulumutka.polyconstr.models.Repository;

import javafx.fxml.Initializable;

import javafx.scene.control.TextField;

import java.net.URL;

import java.util.ResourceBundle;

public class ShowResultController implements Initializable {

public TextField formulasField;

@Override

public void initialize(URL location, ResourceBundle resources) {

formulasField.setText(Repository.formulas);

}

}

**MainView.fxml**

<?xml version="1.0" encoding="UTF-8"?>

<?import com.bulumutka.polyconstr.ui.GraphCanvas?>

<?import javafx.scene.control.\*?>

<?import javafx.scene.layout.\*?>

<GridPane fx:id="mainPane"

fx:controller="com.bulumutka.polyconstr.controllers.MainController"

stylesheets="@../styles/styles.css"

xmlns:fx="http://javafx.com/fxml">

<columnConstraints>

<ColumnConstraints hgrow="ALWAYS"/>

</columnConstraints>

<rowConstraints>

<RowConstraints prefHeight="50"/>

<RowConstraints vgrow="ALWAYS"/>

</rowConstraints>

<BorderPane GridPane.rowIndex="0"

GridPane.columnIndex="0"

style="-fx-background-color: #bdc192">

<center>

<HBox alignment="CENTER" spacing="10">

<Button fx:id="vertexButton"

styleClass="vertex-button"

text="Vertex"/>

<Button fx:id="edgeButton"

styleClass="edge-button"

text="Edge"/>

<Button fx:id="startVertexButton"

style="-fx-font-size: 16; -fx-padding: 5"

text="Start"/>

<Button fx:id="clearButton"

style="-fx-font-size: 16; -fx-padding: 5"

text="Clear"/>

</HBox>

</center>

<right>

<HBox alignment="CENTER" prefWidth="100">

<Button alignment="CENTER" fx:id="countButton"

style="-fx-padding: 5; -fx-font-size: 16"

text="Count"/>

</HBox>

</right>

</BorderPane>

<StackPane fx:id="drawArea"

GridPane.rowIndex="1"

GridPane.columnIndex="0"

style="-fx-background-color: #738C73">

<GraphCanvas fx:id="canvas"/>

</StackPane>

</GridPane>

**ShowResultView.fxml**

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.\*?>

<?import javafx.scene.layout.AnchorPane?>

<?import javafx.scene.layout.VBox?>

<AnchorPane xmlns="http://javafx.com/javafx"

xmlns:fx="http://javafx.com/fxml"

fx:controller="com.bulumutka.polyconstr.controllers.ShowResultController"

style="-fx-background-color: #fffff6"

prefHeight="400.0" prefWidth="600.0">

<VBox spacing="15" prefWidth="1000">

<Label text="Formulas" style="-fx-font-size: 18"/>

<TextField editable="false" fx:id="formulasField" style="-fx-font-size: 18"

maxWidth="1000"/>

</VBox>

</AnchorPane>

# Python

**FindExpression.py**

import numpy as np

import sympy as sp

def find\_polynomial(vector):

n = vector[0]

m = vector[1]

# First term R'(T)

sub\_graph\_number = int(vector[2])

index = 3

first\_term = 0

for num in range(sub\_graph\_number):

edges\_number = int(vector[index])

index += 1

edges\_weights = [vector[index + j] \* 2 for j in range(edges\_number)]

index += edges\_number

vertex\_number = int(vector[index])

index += 1

for \_ in range(vertex\_number):

sub = vector[index]

index += 1

args\_number = int(vector[index])

index += 1

args = [vector[index + j] for j in range(args\_number)]

index += args\_number

sum1 = sum([bernoulli\_barns(args[i], edges\_weights) for i in range(args\_number)])

first\_term += sum1 \* sub

# Second term R''(T)

sub\_graph\_number = int(vector[index])

index += 1

second\_term = 0

for \_ in range(sub\_graph\_number):

edges\_number = int(vector[index])

index += 1

edges\_weights = [vector[index + j] \* 2 for j in range(edges\_number)]

index += edges\_number

vertex\_number = int(vector[index])

index += 1

for \_ in range(vertex\_number):

j\_weight = vector[index] \* 2

index += 1

args\_number = int(vector[index])

index += 1

args = [vector[index + j] for j in range(args\_number)]

index += args\_number

edges\_weights.remove(j\_weight)

sum1 = sum([bernoulli\_barns(args[i], edges\_weights) for i in range(args\_number)])

edges\_weights.append(j\_weight)

second\_term += sum1

assert (index == len(vector))

return first\_term + second\_term

def find\_k\_graph(n, t):

m = int(n \* (n - 1) / 2)

n = int(n)

T = sp.symbols('T')

expr = T \*\* (m - 1) / (2 \*\* (n - 2) \* sp.factorial(m - 1)) \* sum(t) / np.prod(t)

expr += double\_sum(t) / np.prod(t) \* T \*\* (n - 2) / 2 \*\* (n - 2) / sp.factorial(n - 2)

return sp.expand(expr)

def bernoulli\_barns(S, t):

T = sp.symbols('T')

k = len(t)

expr = 1 / (np.prod(t))

expr \*= (T + S) \*\* k / sp.factorial(k) + 0.5 \* sum(t) \* (T + S) \*\* (k - 1) / sp.factorial(k - 1)

return sp.expand(expr)

def find\_cycle\_graph(n, t):

T = sp.symbols('T')

expr = T \*\* (n - 1) \* sum(t) / (2 \*\* (n - 2)) / sp.factorial(n - 1) / np.prod(t)

expr += double\_sum(t) \* T \*\* (n - 2) / 2 \*\* (n - 2) / sp.factorial(n - 2) / np.prod(t)

return sp.expand(expr)

def double\_sum(a):

sum1 = 0

for i in range(len(a)):

for j in range(i + 1, len(a)):

sum1 += a[i] \* a[j]

return sum1

def unpack\_vector(pathname):

return find\_polynomial(read\_vector(pathname))

def calculate\_polynomial(expression, x):

T = sp.sybols('T')

return expression.subs(T, x)

def read\_vector(pathname):

vector = []

with open(pathname, 'r') as reader:

vector = [float(line) for line in reader]

return vector

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |