Цель работы

В случайном графе вычислить вероятность существования пути между заданной парой вершин. Построить зависимость вероятности существования пути в случайном графе от вероятности существования ребра.

1. Задание

На рисунке 1 изображён случайный граф. Ищем вероятность существования пути из вершины 1 в вершину 4. P1=P2=…=P7

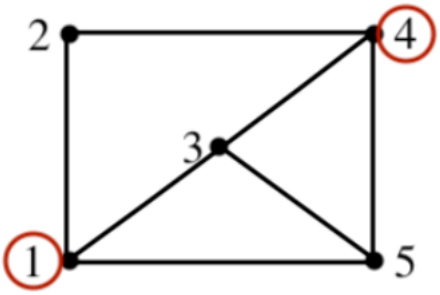

Рисунок 1. Случайный граф

2. Выполнение задания

Для нахождения вероятности существования пути 1-4 нужно провести декомпозицию случайного графа.

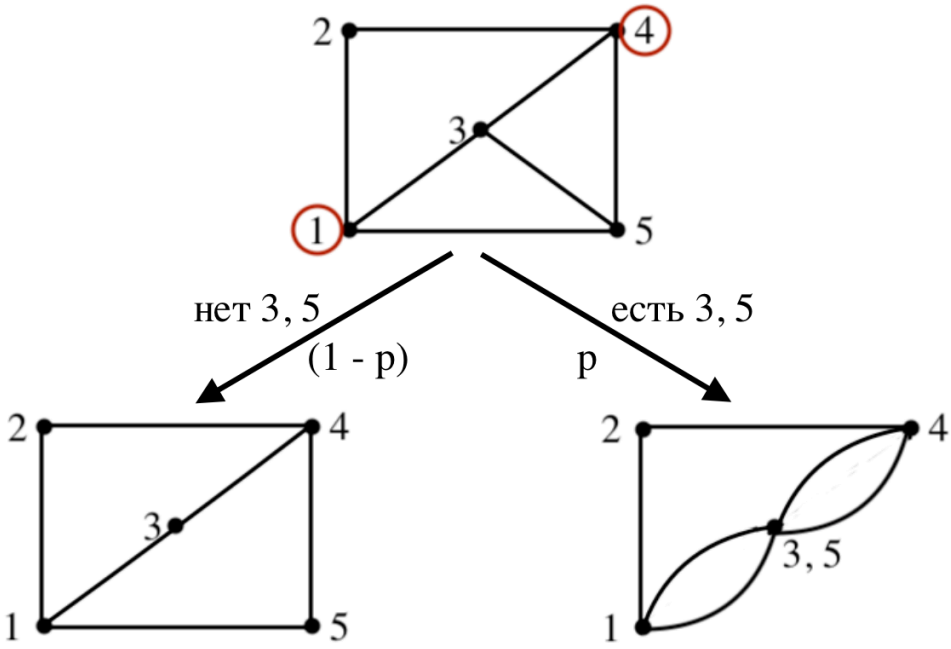

Рисунок 2. Декомпозиция графа

$\Pr\{\text{путь } 1, 4 \mid \text{нет } 3, 5\} = 3p^2 - 3p^4 + p^6$
$\Pr\{\text{путь } 1, 4 \mid \text{есть } 3, 5\} = p^2 + (2p - p^2) - p^2(2p - p^2)^2$

Итоговая формула вероятности пути из вершины 1 в вершину 4:

$$\Pr\{\text{путь } 1,4\} = (3p^2 - 3p^4 + p^6)(1-p) + p(p^2 + (2p - p^2) - p^2(2p - p^2)^2) = -2p^7 + 5p^6 - 7p^4 + 2p^3 + 3p^2$$

Используя приведенную выше формулу, вычисляем зависимость вероятности существования пути от вероятности существования ребра.

p ребра	p теоретическая	p практическая
0	0	0
0.1	0.0313048	0.031304800000000001
0.2	0.125094	0.1250944000000003
0.3	0.270508	0.27050760000000024
0.4	0.446003	0.44600320000000004
0.5	0.625	0.625
0.6	0.782093	0.7820928
0.7	0.898836	0.8988364000000001
0.8	0.96809	0.9680896
0.9	0.995911	0.9959111999999999
1	1	1

Теоретическая вероятность была вычислена вручную с помощью формулы. Практическая вероятность была вычислена программно с помощью полного перебора.

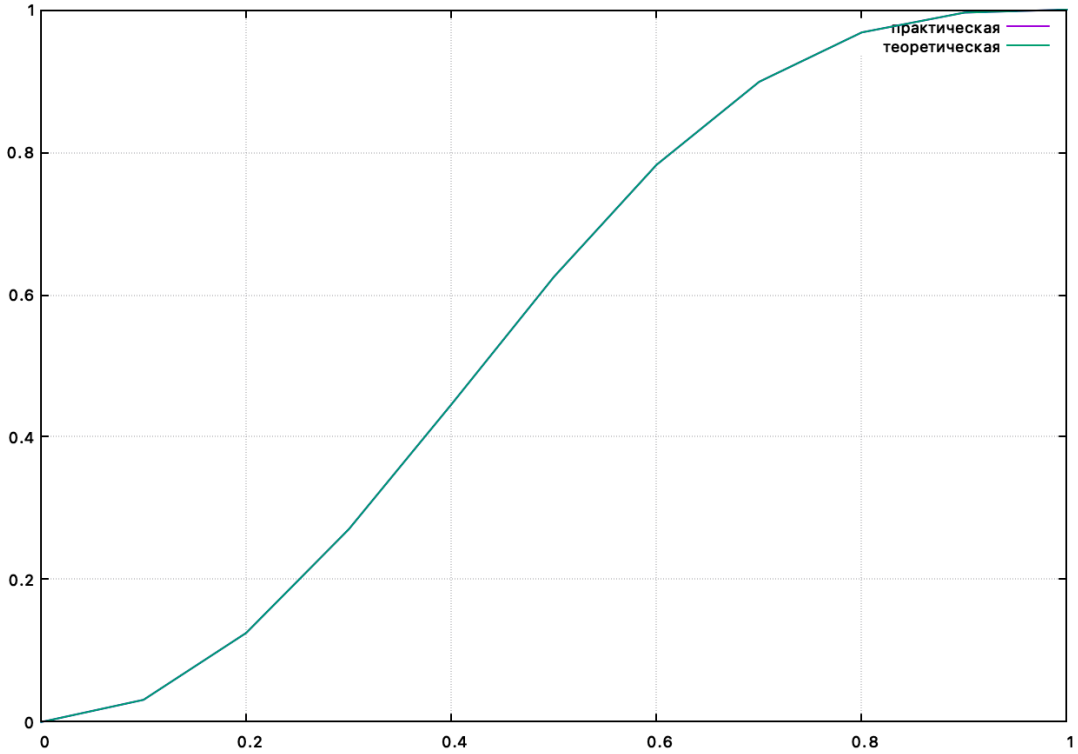

График 1. Зависимость P пути от P ребра в заданном случайном графе

Как видно из графика 1 итоговая формула оказалась верной, т.к. результат вычисления по ней полностью совпал с результатом полного перебора.

Вывод

В ходе выполнения лабораторной работы была выведена формула вероятности существования пути между заданной парой вершин 1 и 4 в графе, правильность которой была подтверждена результатами программного полного перебора всех возможных подграфов случайного графа. Построен график зависимости вероятности существования графа от вероятности существования ребра.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Scanner;

public class Graph {
    public int[][] matrix;
    public LinkedList<Integer> vertex = new LinkedList<>();
    public LinkedList<Pair<Integer, Integer>> edges = new LinkedList<>();
    LinkedList<LinkedList<Integer>> listCombination = new LinkedList<>();
    double[] probability = {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
1};
    double[] pr = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    public void readFile(String filepath) throws FileNotFoundException {
        Scanner scanner = new Scanner(new File(filepath));
        int n = scanner.nextInt();
        matrix = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }
    }

    public void printMatrix() {
        System.out.println("  __1_2_3_4_5_");
        //System.out.println("  ___________");
        int r = 1;
        for (int[] ints : matrix) {
            System.out.print(r++ + " | ");
            for (int j = 0; j < matrix.length; j++) {
                System.out.print(ints[j] + " ");
            }
            System.out.println();
        }
    }

    public boolean findPath(int a, int b) {
        if (a == b)
            return true;

        if (!vertex.contains(a)) {
            vertex.add(a);
        }

        for (int i = 0; i < matrix.length; i++) {
            if (matrix[i][a] == 1 && !vertex.contains(i)) {
                vertex.add(i);
                if (findPath(i, b)) {
                    return true;
                }
            }
        }
        return false;
    }

    public void funct() {
        getEdges();
```

```
        int numE = edges.size();
        for (int i = 0; i <= numE; i++) {
            listCombination.clear();
            int[] a = new int[i];

            combination(a, numE, i, true);
            while (combination(a, numE, i, false)) {}

            for (int j = 0; j < listCombination.size(); j++) {
                matrix = getMatrix(j);
                vertex.clear();

                if (findPath(0, 3)) {
                    probability(i, numE - i);
                }
            }
        }
    }

    public void probability(int a, int b) {
        for (int i = 0; i < probability.length; i++) {
            pr[i] += Math.pow(probability[i], a) * Math.pow(1 -
probability[i], b);
        }
    }

    public int[][] getMatrix(int j) {
        int[][] otherMatrix = new int[matrix.length][matrix.length];
        for (int i = 0; i < listCombination.get(j).size(); i++) {
            int index = listCombination.get(j).get(i) - 1;
            int x = edges.get(index).getVertexOne();
            int y = edges.get(index).getVertexTwo();
            otherMatrix[x][y] = 1;
            otherMatrix[y][x] = 1;
        }
        return otherMatrix;
    }

    public void getEdges() {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = i; j < matrix.length; j++) {
                if (matrix[i][j] == 1) {
                    edges.add(new Pair<>(i, j));
                }
            }
        }
    }

    public boolean combination(int[] a, int n, int m, boolean start) {
        if (start) {
            LinkedList<Integer> listEdges = new LinkedList<>();
            for (int j = 0; j < m; j++) {
                a[j] = j + 1;
                listEdges.add(a[j]);
            }
            listCombination.add(listEdges);
        }
        int k = m;
        for (int i = k - 1; i >= 0; --i) {
            LinkedList<Integer> qw = new LinkedList<>();
            if (a[i] < n - k + i + 1) {
                ++a[i];
                for (int j = i + 1; j < k; ++j) {
                    a[j] = a[j - 1] + 1;
```

```
                }
                for (int j : a) qw.add(j);
                listCombination.add(qw);
                return true;
            }
        }
        return false;
    }

    public void writeToFile() {
        try {
            FileWriter writer = new FileWriter("probability.txt", false);
            for (int i = 0; i < pr.length; i++) {
                String str = probability[i] + " " + pr[i] + "\n";
                writer.write(str);
                writer.flush();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws FileNotFoundException {
        Graph graph = new Graph();
        graph.readFile("граф.txt");
        graph.printMatrix();
        System.out.println(graph.findPath(0, 3));
        System.out.println(graph.vertex);
        graph.funct();
        graph.writeToFile();
    }
}
```