Цель работы

Используя имитационное моделирование, в случайном графе вычислить вероятность существования пути между заданной парой вершин. Построить зависимость вероятности существования пути в случайном графе от вероятности существования ребра. Сравнить результаты моделирования, ускоренного моделирования и полного перебора.

1. Задание

На рисунке 1 изображён случайный граф. Ищем вероятность существования пути из вершины 1 в вершину 4. p1 = p2 = … = p7

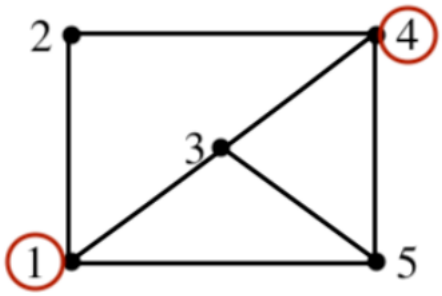

Рисунок 1. Случайный граф

2. Выполнение задания

Для нахождения вероятности существования пути с помощью имитационного моделирования нужно задать количество проводимых экспериментов N.

$$N = \frac{2.25}{\varepsilon^2}, \text{где } \varepsilon - \text{точность моделирования.}$$

P ребра	P пути через полный перебор	Моделирование при E = 0.1	Моделирование при E = 0.01
0	0	0	0
0.1	0.03130480000000001	0.03111111111111112	0.032711111111111114
0.2	0.1250944000000003	0.11555555555555558	0.1252888888888889
0.3	0.27050760000000024	0.2622222222222223	0.26866666666666666
0.4	0.46600320000000004	0.42666666666666675	0.4421333333333333
0.5	0.625	0.6355555555555558	0.6236444444444444
0.6	0.7820928	0.7822222222222224	0.7803555555555556
0.7	0.8988364000000001	0.9155555555555558	0.8979555555555555
0.8	0.9680896	0.9733333333333336	0.9699111111111111
0.9	0.9959111999999999	0.9955555555555559	0.9964
1	1	1	1

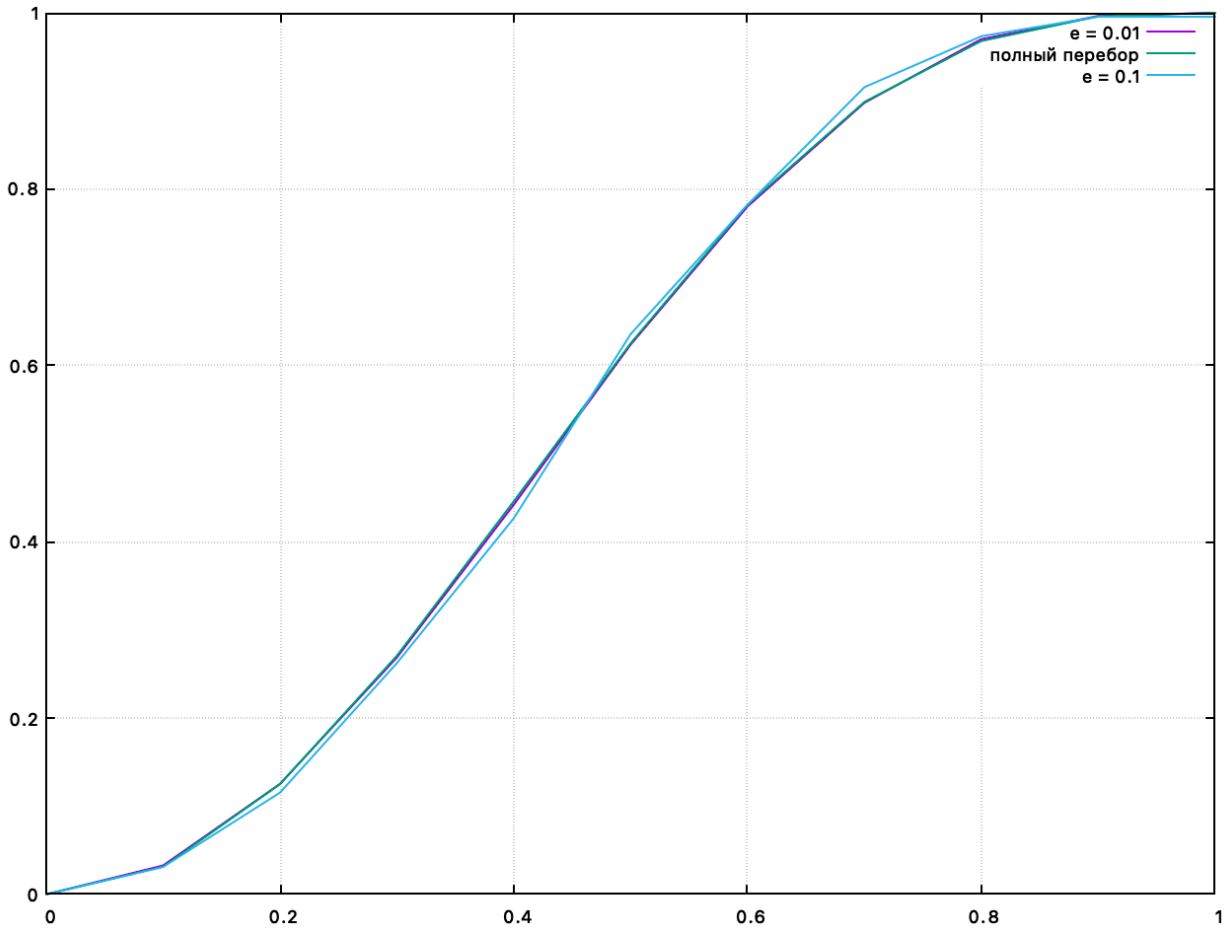

График 1. Зависимость P пути от P ребра при полном переборе и имитационном моделировании

Далее используем алгоритм ускоренного моделирования, представленный на рисунке 2.

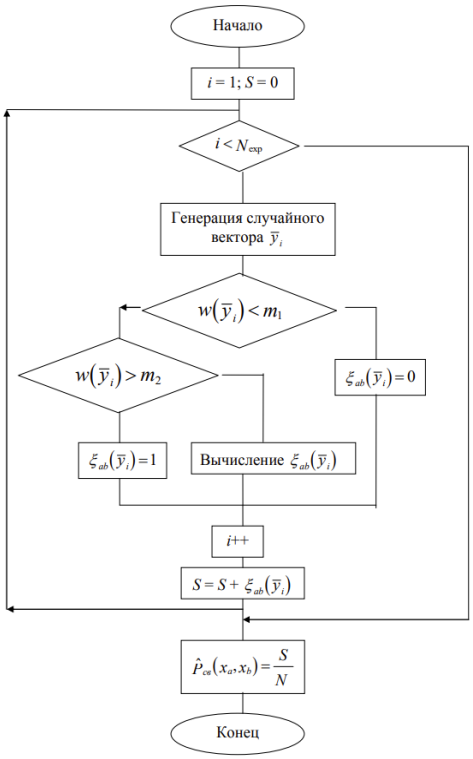

Рисунок 2. Алгоритм ускоренного моделирования

Для рассматриваемого графа l_min = 2, l_max = 5.

P ребра	P пути через полный перебор	Моделирование при E = 0.1	Моделирование при E = 0.01
0	0	0	0
0.1	0.03130480000000001	0.026666666666666672	0.032711111111111114
0.2	0.1250944000000003	0.11555555555555558	0.1252888888888889
0.3	0.27050760000000024	0.2711111111111112	0.26866666666666666
0.4	0.46600320000000004	0.4311111111111112	0.4421333333333333
0.5	0.625	0.6044444444444446	0.6236444444444444
0.6	0.7820928	0.7822222222222224	0.7803555555555556
0.7	0.8988364000000001	0.9100000000000003	0.8979555555555555
0.8	0.9680896	0.9511111111111114	0.9699111111111111
0.9	0.9959111999999999	0.9955555555555559	0.9964
1	1	1	1

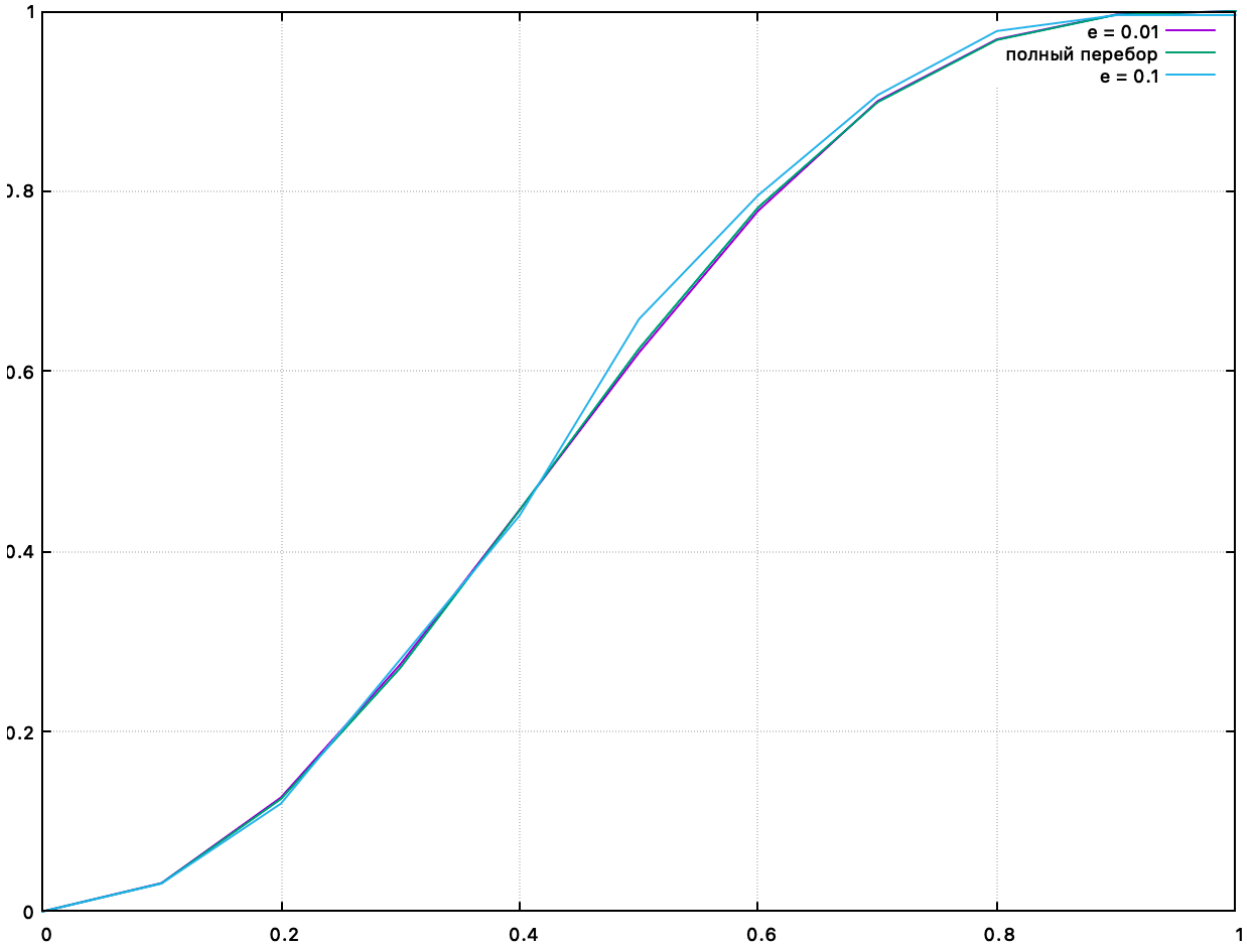

График 2. Зависимость P пути от P ребра при полном переборе и ускоренном имитационном моделировании

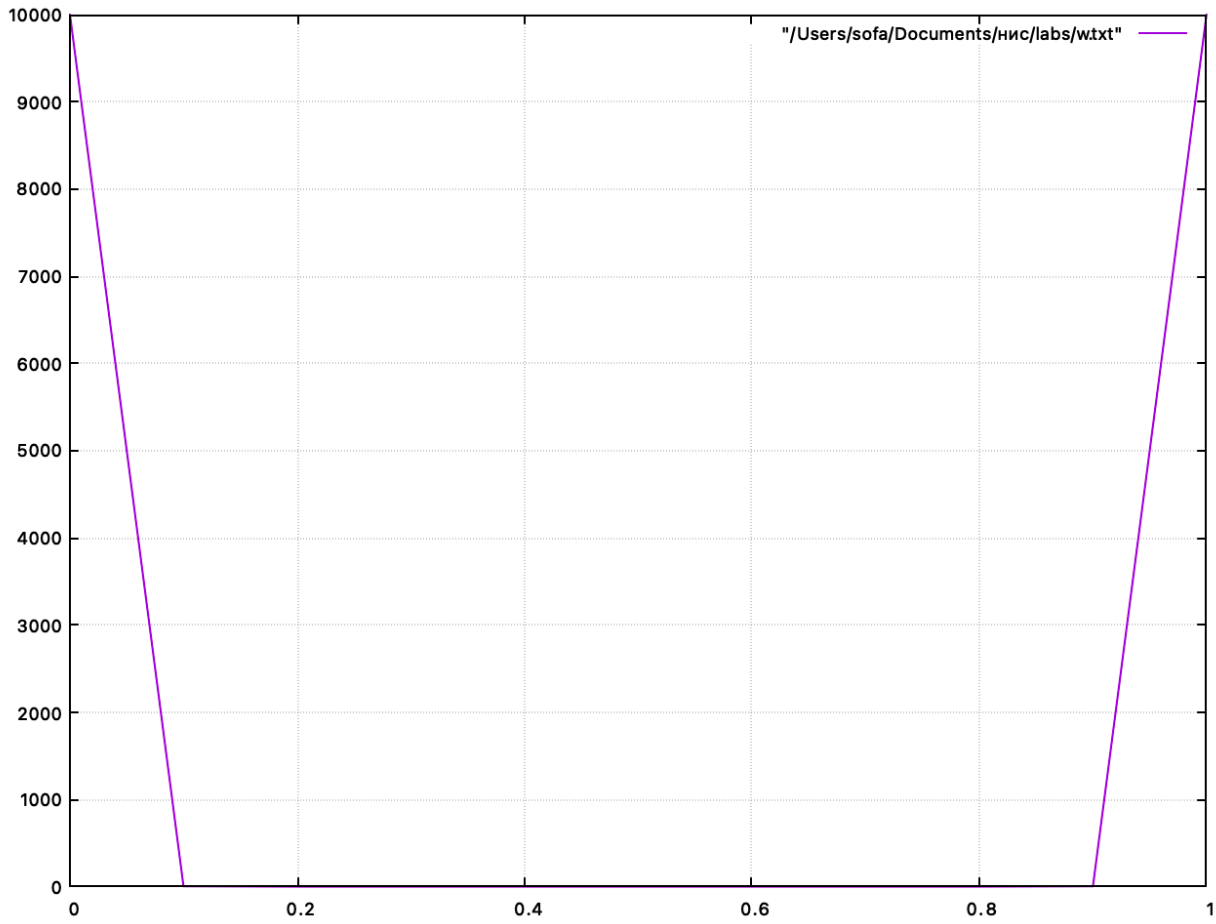

График 3. Выигрыш при использовании ускоренного моделирования

Вывод

В ходе выполнения лабораторной работы была вычислена вероятность существования пути между заданной парой вершин 1 и 4 в графе с помощью обычного и ускоренного имитационного моделирования, правильность которой была подтверждена результатами программного полного перебора всех возможных подграфов случайного графа. Построен график выигрыша ускоренного имитационного моделирования над обычным.

```
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<>();
    public LinkedList<LinkedList<Integer>> listCombination = new
LinkedList<>();
    public LinkedList<LinkedList<Integer>> listAllCombinations = new
LinkedList<>();
    private LinkedList<Double> winList = 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};
    private int minPathLength = 100;
    private int maxPathLength = 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_");
        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)) {
                    int lenPath = vertex.size();
                    if (lenPath < minPathLength)
                        minPathLength = lenPath;
                    if (lenPath > maxPathLength)
                        maxPathLength = lenPath;
                    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);
            listAllCombinations.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);
                listAllCombinations.add(qw);
                return true;
            }
        }
        return false;
    }

    private double comb(int n, int m) {
        return getFactorial(n) / (getFactorial(n - m) * getFactorial(m));
    }

    private double getFactorial(int f) {
        int res = 1;
        for (int i = 1; i <= f; i++) {
            res *= i;
        }
        return res;
    }

    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 int[] getRandomVector(double p) {
        int[] vector = new int[edges.size()];

        for (int i = 0; i < vector.length; i++) {
            boolean val = Math.random() > p;
            if (val)
                vector[i] = 0;
            else
                vector[i] = 1;
        }
        return vector;
    }

    private int getVectorWeight(int[] vector) {
        int weight = 0;
        for (int j : vector) {
            if (j == 1)
                weight++;
```

```
        }
        return weight;
    }

    private int[][] fromVectorToMatrix(int[] vector) {
        int[][] otherMatrix = new int[matrix.length][matrix.length];
        for (int i = 0; i < vector.length; i++) {
            if (vector[i] == 1) {
                int x = edges.get(i).getVertexOne();
                int y = edges.get(i).getVertexTwo();
                otherMatrix[x][y] = 1;
                otherMatrix[y][x] = 1;
            }
        }
        return otherMatrix;
    }

    public double iml(double p, double e, boolean fast) {
        double nExp = 2.25 / Math.pow(e, 2);
        int s = 0;
        int weight;
        double c = 0.0;

        for (int j = getMinPathLength(); j <= getMaxPathLength(); j++) {
            c += comb(edges.size(), j) * Math.pow(p, j) * Math.pow(1 - p,
edges.size() - j);
        }
        winList.add(1 / c);

        for (int i = 1; i < nExp; i++) {
            int[] randVector = getRandomVector(p);
            weight = getVectorWeight(randVector);

            if (fast) {
                if (weight < getMinPathLength()) {
                    s += 0;
                } else {
                    if (weight > getMaxPathLength()) {
                        s++;
                    } else {
                        matrix = fromVectorToMatrix(randVector);
                        vertex.clear();
                        if (findPath(0, 3))
                            s++;
                    }
                }
            } else {
                matrix = fromVectorToMatrix(randVector);
                vertex.clear();
                if (findPath(0, 3))
                    s++;
            }
        }
        return s / nExp;
    }

    public void writeIML(String name, double e, boolean fast) {
        try {
            FileWriter writer = new FileWriter(name, false);
            for (int i = 0; i < probability.length; i++) {
                double prExistence = iml(probability[i], e, fast);
                String str = probability[i] + " " + prExistence + "\n";
                writer.write(str);
                writer.flush();
```

```
            }
        } catch (IOException er) {
            er.printStackTrace();
        }
    }

    public int getMinPathLength() {
        return minPathLength / 2 + 1;
    }

    public int getMaxPathLength() {
        return edges.size() - 2;
    }

    public void writeWinner(String name) {
        try {
            FileWriter writer = new FileWriter(name, false);
            for (int i = 0; i < probability.length; i++) {
                String str = probability[i] + " " + winList.get(i) + "\n";
                System.out.println(winList.get(i));
                writer.write(str);
                writer.flush();
            }
        } catch (IOException er) {
            er.printStackTrace();
        }
    }

    public static void main(String[] args) throws FileNotFoundException {
        Graph graph = new Graph();
        graph.readFile("/Users/sofa/Documents/тп/lab/PIS/граф.txt");
        graph.printMatrix();
        System.out.println(graph.findPath(0, 3));
        System.out.println(graph.vertex);
        graph.funct();
        graph.writeToFile();

        System.out.println(graph.getMinPathLength());
        System.out.println(graph.getMaxPathLength());

        graph.writeIML("simple1", 0.1, false);
        graph.writeIML("simple2", 0.01, false);

        graph.writeIML("fast1", 0.1, true);
        graph.writeIML("fast2", 0.01, true);

        graph.writeIML("p", 0.1);
        graph.writeIML("p2", 0.01);

        graph.writeWinner("w.txt");
    }
}
```