

1. **Цель работы:** получение практических навыков использования моделирования для оценки надежности вычислительных сетей.

2. Ход работы

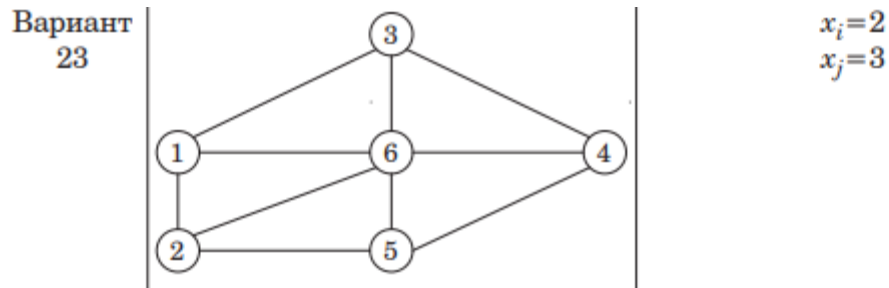


Рис.1 - Топология случайного графа. Необходимо вычислить вероятность пути 1,5

Для выполнения лабораторной работы нужно написать программу обычного и ускоренного имитационного моделирования и вычислить вероятность существования пути с заданной точностью $\varepsilon = 0.01$.



Рис.2 - Блок-схема имитационного моделирования

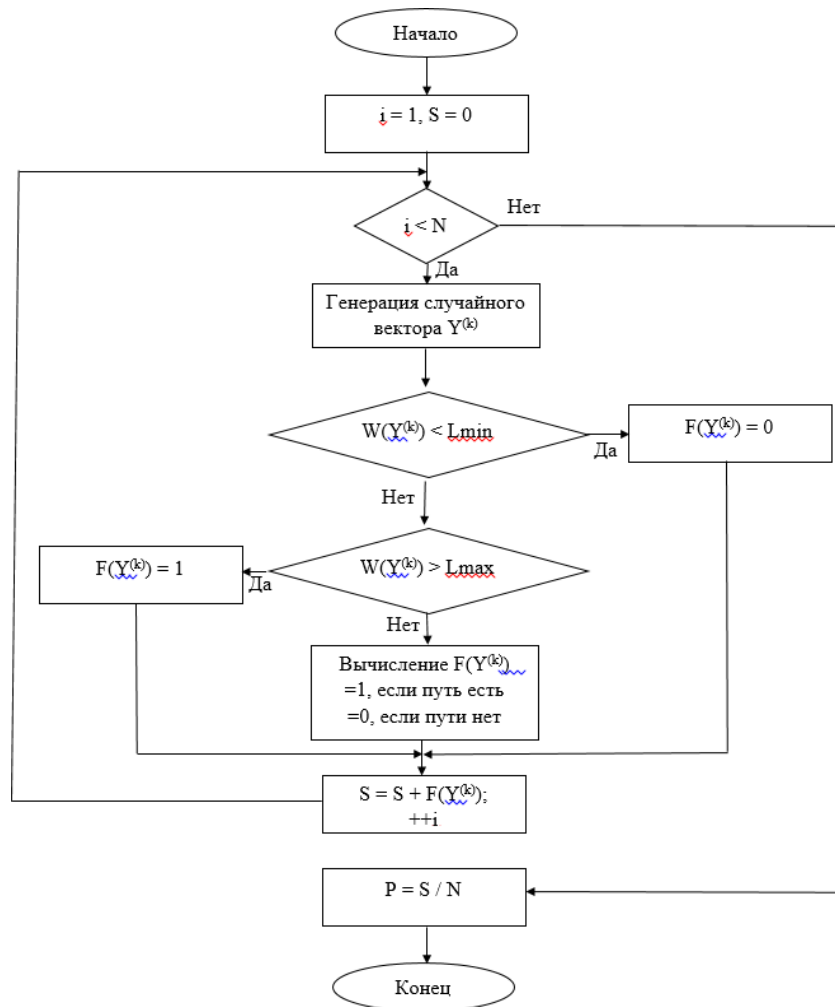


Рис.3 - Блок-схема ускоренного имитационного моделирования

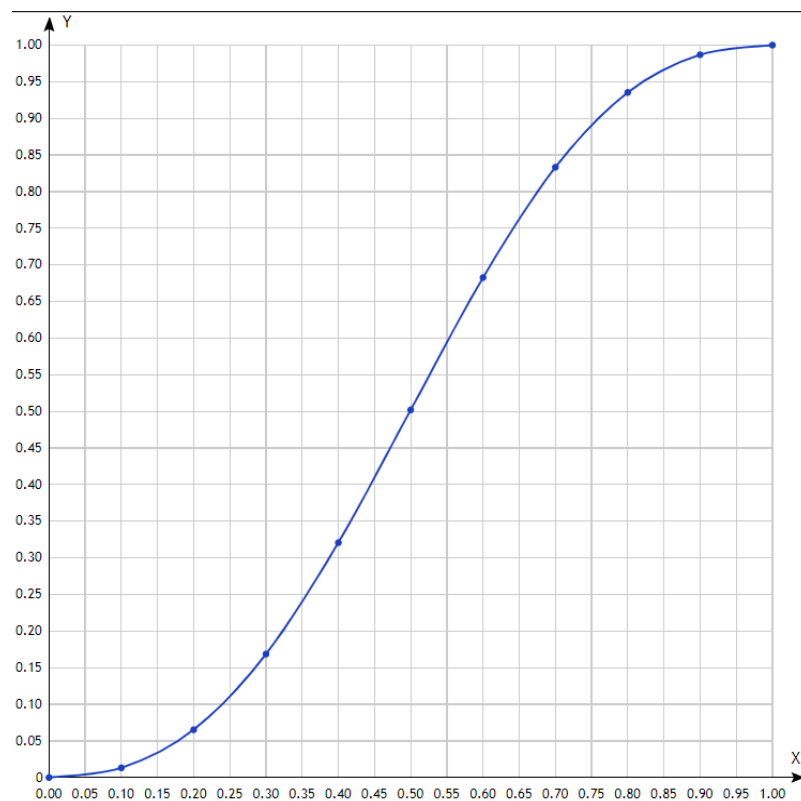


Рис.4 – График вероятности существования пути при полном переборе

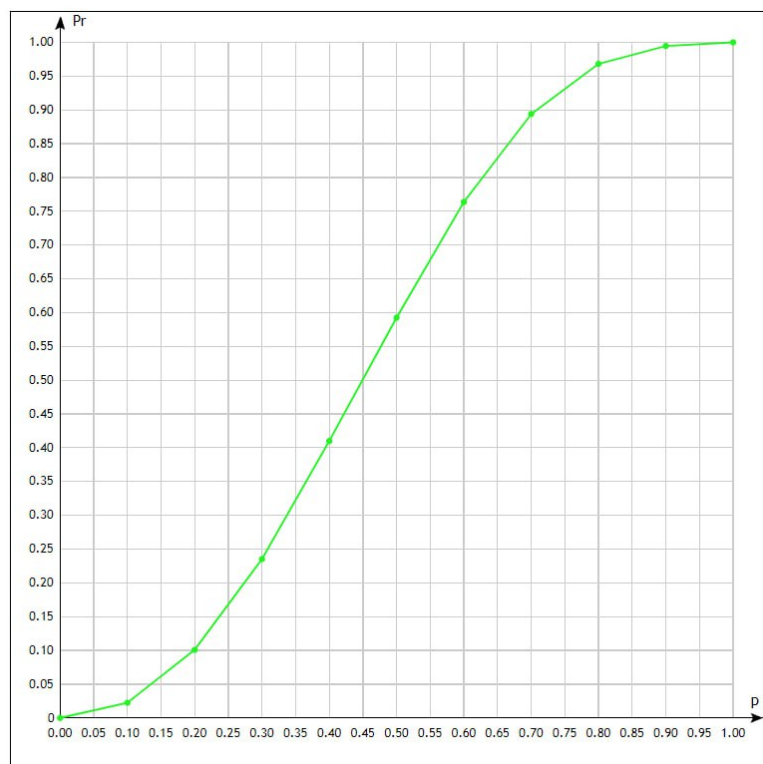


Рис.5 - График вероятности существования пути при имитационном моделировании

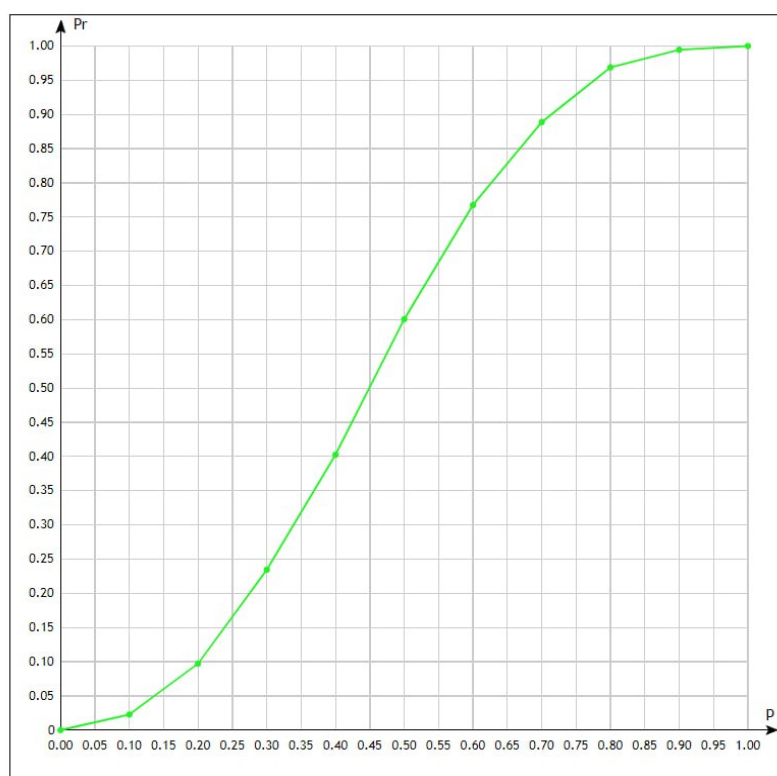


Рис.6 - График вероятности существования пути при ускоренном имитационном моделировании

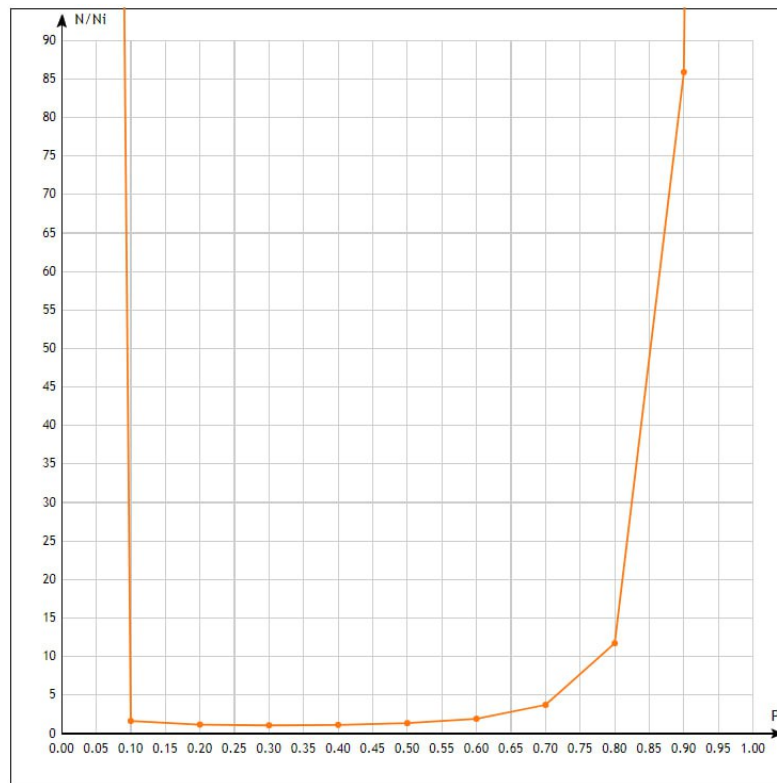


Рис.7 - График применения имитационного моделирования

Вывод

В ходе лабораторной работы были вычислены вероятности существования пути обычным и ускоренным имитационным моделированием. Графики получились близкими друг к другу. Имитационное моделирование имеет небольшое расхождение с заданной точностью

Листинг

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;

public class graphs {
    private int v;
    private int e;
    private int [][] matrix;
    private int visited[];
    private double p;
    private boolean isPath = false;
    private int a;
    private int b;
    private Vector<Integer> o = new Vector<Integer>();
    private int W = 0;

    public graphs(int v, double p, int a, int b) {
```

```

        this.v = v;
        matrix = new int[v][v];
        visited = new int[v];
        this.p = p;
        this.a = a;
        this.b = b;
    }

    private void fill() {
        for(int i = 0; i < v - 1; i++) {
            for(int j = i + 1; j < v; j++) {
                if(matrix[i][j] == 1) {
                    e++;
                }
                o.addElement(matrix[i][j]);
            }
        }
    }

    private void dfs(int cur) {
        visited[cur] = 1;
        if(cur == b) {
            isPath = true;
            return;
        }
        for(int i = 0; i < v; i++) {
            if((matrix[cur][i] == 1) && (visited[i] == 0)) {
                dfs(i);
            }
        }
    }

    private double bruteForce() {
        double g = 0.0;
        for (int k = 0; k < Math.pow(2, e); k++) {

            int[] bs = new int [e];
            int kk = k;

            for(int i = e - 1; i >= 0; i--) {
                if(kk != 0) {
                    bs[i] = kk%2;
                    kk /= 2;
                }
            }

            double pr = 1.0;
            int idx = 0;
            int oidx = 0;

```

```

        for (int i = 0; i < v - 1; i++) {
            for (int j = i + 1; j < v; j++) {
                if ((int)o.get(oidx) == 1) {
                    matrix[i][j] = bs[idx];
                    matrix[j][i] = bs[idx];
                    if(bs[idx] == 1) {
                        pr *= p;
                    }
                    else {
                        pr *= 1-p;
                    }
                    idx++;
                }
                else {
                    matrix[i][j] = 0;
                    matrix[j][i] = 0;
                }
                oidx++;
            }
        }
        isPath = false;

        dfs(a);

        if (isPath) {
            g += pr;
        }
        for (int ii = 0; ii < v; ii++) {
            visited[ii] = 0;
        }
    }
    return g;
}

private int random() {
    int r = (int)(Math.random()*100 + 1);
    if (r <= (int)(p*100)) {
        W++;
        return 1;
    }
    else
        return 0;
}

```

```

public double simulationModeling(double E) {
    int S = 0;
    int N = (int)(2.25 / Math.pow(E, 2));
    for (int i = 0; i < N; ++i){

```

```

int idx = 0;
int oidx = 0;
int[] y = new int [e];
W = 0;
for (int j = 0; j < e; ++j){//генерируем вектор случайной длины
    y[j] = random();
}
for (int ii = 0; ii < v - 1; ii++) { //строим подграф по вектору
    for (int jj = ii + 1; jj < v; jj++) {
        if ((int)o.get(oidx) == 1) {
            matrix[ii][jj] = y[idx];
            matrix[jj][ii] = y[idx];
            idx++;
        }
        else {
            matrix[ii][jj] = 0;
            matrix[jj][ii] = 0;
        }
        oidx++;
    }
}
isPath = false;

dfs(a);

if(isPath) {
    S++;
}
for (int ii = 0; ii < v; ii++)
    visited[ii] = 0;
}
//возвращаем оценку вероятности
return (double)S / N;
}

public double fastSimulationModeling(double E, int Lmin, int Lmax) {
    int S = 0;
    int colvoOtbr = 0;
    int N = (int)(2.25 / Math.pow(E, 2));

    for (int i = 0; i < N; ++i){
        int idx = 0;
        int oidx = 0;
        int[] y = new int [e];
        W = 0;

        for (int j = 0; j < e; ++j) {
            y[j] = random();
        }
    }
}

```

```

        if(W < Lmin) {
            colvoOtbr++;
            continue;
        }
        else if(W > Lmax){
            S++;
            colvoOtbr++;
            continue;
        }

        for (int ii = 0; ii < v - 1; ii++) {
            for (int jj = ii + 1; jj < v; jj++) {
                if ((int)o.get(oidx) == 1) {
                    matrix[ii][jj] = y[idx];
                    matrix[jj][ii] = y[idx];
                    idx++;
                }
                else {
                    matrix[ii][jj] = 0;
                    matrix[jj][ii] = 0;
                }
                oidx++;
            }
        }
        isPath = false;

        dfs(a);

        if(isPath){
            S++;
        }
        for (int ii = 0; ii < v; ii++) {
            visited[ii] = 0;
        }
    }
    fastWin(colvoOtbr);
    colvoOtbr = 0;
    return (double)S / N;
}

public void fastWin(int Ni) {
    double E = 0.01;
    int N = (int) (2.25 / Math.pow(E, 2));

    System.out.println("ПсевдоГрафик: " + (double) N / (N - Ni));
}

int factorial(int a){

```



```

        int res = 1;

        for (int i = 1; i <= a; ++i){
            res *= i;
        }
        return res;
    }

    int combination(int n, int k){
        return factorial(n) / (factorial(n - k) * factorial(k));
    }

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("src/matrix.txt"));
        List<String> lines = new ArrayList<>();

        while (br.ready()) {
            lines.add(br.readLine());
        }

        int size = lines.get(1).split(" ").length;

        String[] line = lines.get(0).split(" ");

        int a = Integer.parseInt(line[0]);
        int b = Integer.parseInt(line[1]);

        double P = 0;
        double E = 0.01;

        while(P < 1) {
            System.out.println("p = " + String.format("%.1f", P));
            graphs graph = new graphs(size, P, a - 1, b - 1);
            for (int i = 1; i <= size; i++) {
                for (int j = 0; j < size; j++) {
                    String[] line_ = lines.get(i).split(" ");
                    graph.matrix[i - 1][j] = Integer.parseInt(line_[j]);
                }
            }
            graph.fill();

            System.out.println("Полный перебор: Pr{путь(" + a + ", " + b + ")} = " +
                String.format("%.5f", graph.bruteForce()));
            System.out.println("Имитационное моделирование: Pr{путь(" + a + ", " + b
                + ")} = " + String.format("%.5f", graph.simulationModeling(E)));
            System.out.println("Ускоренное имитационное моделирование: Pr{путь(" + a
                + ", " + b + ")} = " + String.format("%.5f", graph.fastSimulationModeling(E, 1, 5)));
            System.out.println();
            P += 0.1;
        }
    }
}

```

