МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 52

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент, канд.техн.наук		Марковская Н.В.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

ИССЛЕДОВАНИЕ ВЕРОЯТНОСТИ СУЩЕСТВОВАНИЯ ПУТИ В СЛУЧАЙНОМ ГРАФЕ

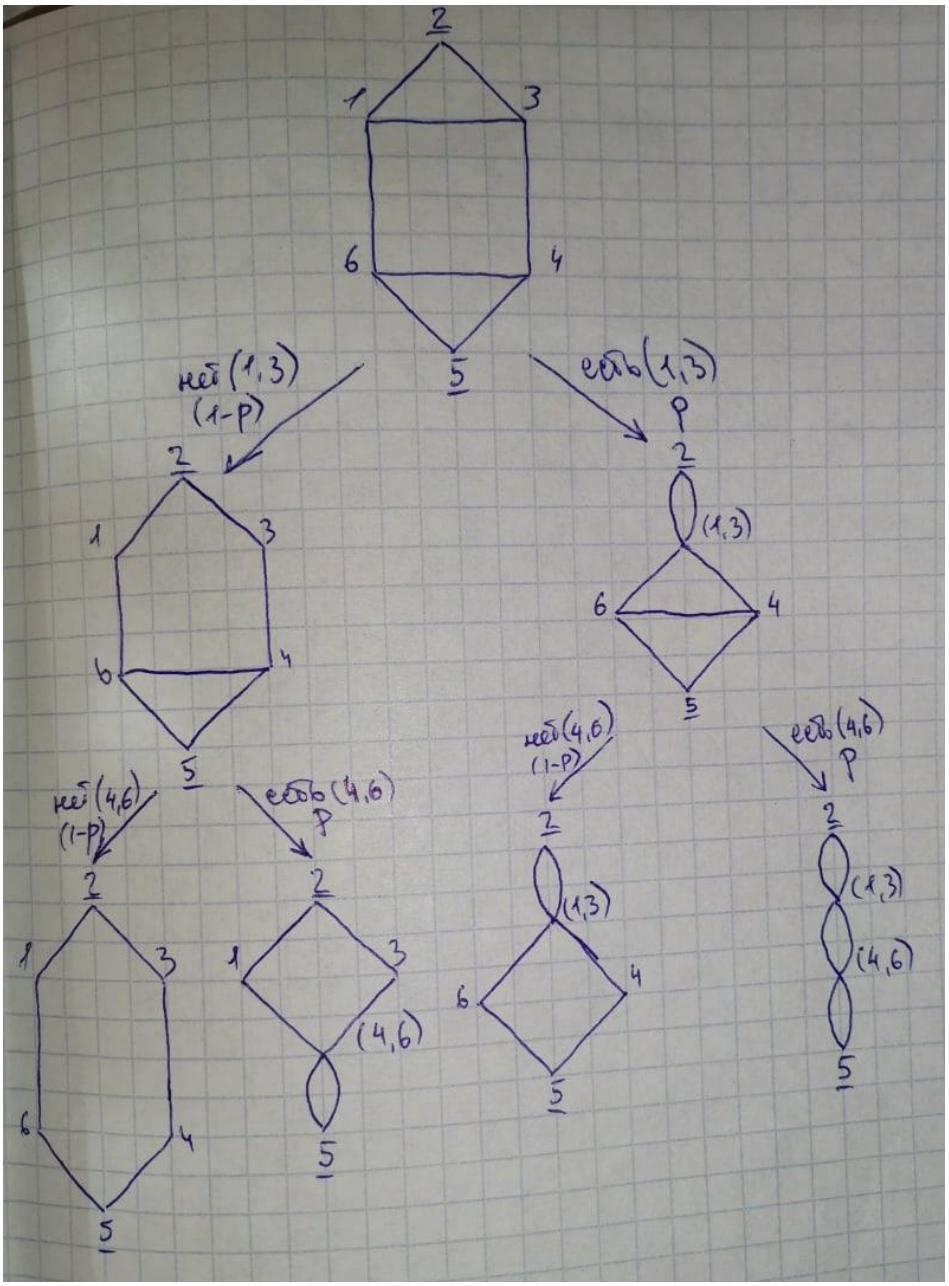
по курсу: Надежность инфокоммуникационных систем

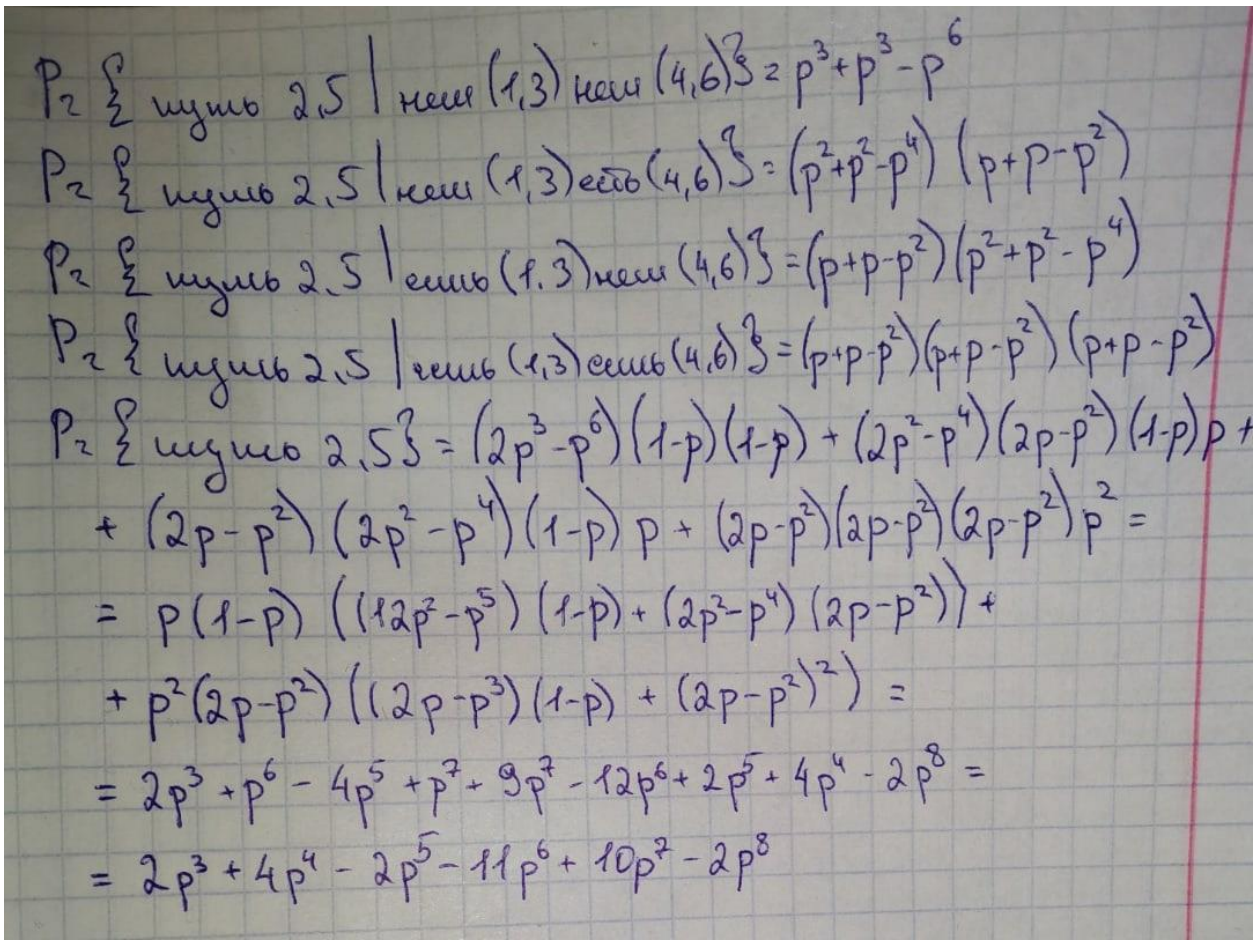
РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5912		Исаева В.И.
	подпись, дата	инициалы, фамилия

Санкт-Петербург
2022

1. Задание
   1.1. Вычислить вероятность $P_{св}$ (x_i, x_j) = ρ_{ij} существования пути между заданной парой вершин x_i, x_j в графе G .
   1.2. Построить зависимость $p_{ij}(\rho)$ вероятности существования пути в случайном графе от вероятности существования ребра.
2. Вычисление вероятности





3. Построение зависимости

Вероятность ∃ ребра	Вероятность ∃ графа	
	Теоретическая	Практическая
0.1	0.00236836	0.002368360000000001
0.2	0.021097	0.021096959999999974
0.3	0.0748624	0.07486235999999984
0.4	0.176988	0.17698816000000045
0.5	0.328125	0.328125
0.6	0.513078	0.5130777600000005
0.7	0.703191	0.7031911599999998
0.8	0.864092	0.8640921600000006
0.9	0.966975	0.9669747600000003
1.0	1.0	1.0

Теоретическая вероятность была вычислена вручную с помощью формулы, выведенной в шаге 2. Практическая вероятность была вычислена программно с помощью полного перебора(прил.1).

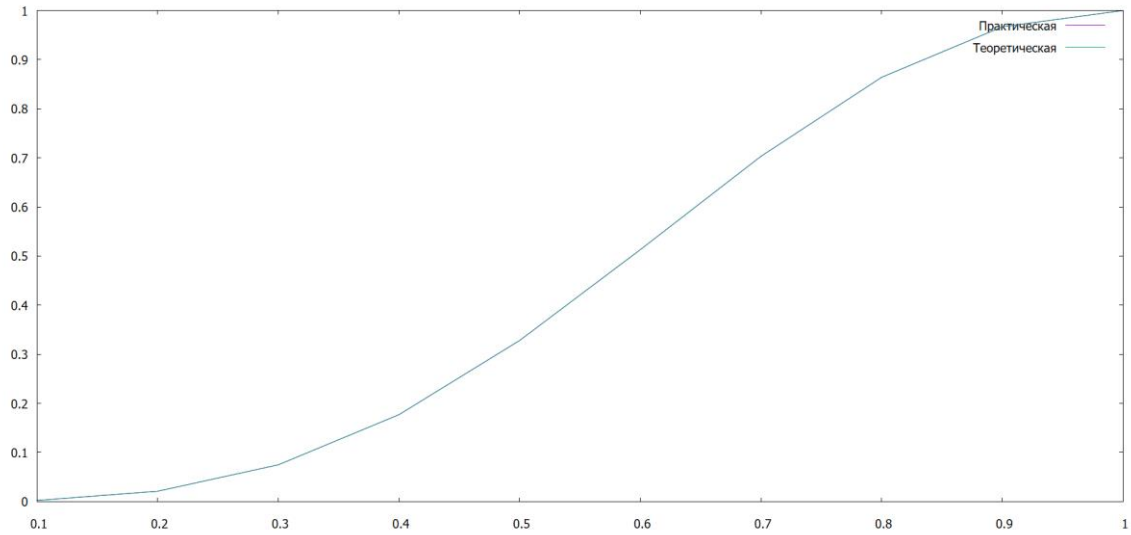


Рисунок 1. График зависимости вероятности существования графа от вероятности существования ребра

Вывод

В ходе выполнения лабораторной работы была вычислена вероятность существования пути между заданной парой вершин 2 и 5 в графе, а также построен график зависимости вероятности существования графа от вероятности существования ребра, на основании которого можно сделать вывод, что с возрастанием вероятности существования ребра возрастает вероятность существования графа.

```
public class Graph {
    int vertices;
    LinkedList<Edge> edges;
    LinkedList<String> paths;

    public Graph(int v, LinkedList<Edge> e){
        vertices = v;
        edges = e;
        paths = new LinkedList<>();
    }

    public double findP(Graph g, double p){
        double sum = 1;
        if(p > 0.9)
            sum = 1;
        for(Edge e : edges){
            if(g.findEdge(e.getPoint1(), e.getPoint2()))
                sum = sum*p;
            else sum= sum*(1-p);
        }
        return sum;
    }

    public boolean findPath(int x, int y){
        if(!findVertex(x)||!findVertex(y))
            return false;
        find(x, y, 0, Integer.toString(x));
        return !paths.isEmpty();
    }

    public boolean find(int x, int y, int num, String path){
        if(num > pow(2, edges.size()))
            return false;

        for(int i = 1; i<= vertices; i++){
            if(!path.contains(Integer.toString(i))) {
                if (findEdge(x, i) && i!=y) {
                    find(i, y, num, path.concat(Integer.toString(i));

                }
            }
        }
        if(findEdge(x, y)) {
            paths.addLast(path.concat(Integer.toString(y)));
            num++;
            return true;
        }
        return false;
    }

    public boolean findEdge(int x, int y){
        for(Edge e : edges){
            if(e.isEdge(x, y))
                return true;
        }
        return false;
    }
```

```
    public boolean findVertex(int x){
        for(Edge e : edges){
            if(e.isVertex(x))
                return true;
        }
        return false;
    }
}


public class Edge {
    int point1;
    int point2;

    public Edge(int x, int y){
        point1 = x;
        point2 = y;
    }

    public boolean isVertex(int x){
        if(point1 == x || point2 == x)
            return true;
        return false;
    }

    public boolean isEdge(int x, int y){
        if(point1 == x && point2 == y || point1 == y && point2 == x)
            return true;
        return false;
    }

    public int getPoint1(){
        return point1;
    }

    public int getPoint2(){
        return point2;
    }
}

public class Main {
    static LinkedList<LinkedList<Edge>> enumerates;

    public static boolean contains(LinkedList<Edge> edges){
        for(LinkedList<Edge> e : enumerates) {
            int sum = 0;
            for (Edge edge : edges){
                if(e.contains(edge))
                    sum++;
            }
            if(sum == edges.size())
                return true;
        }
        return false;
    }

    public static boolean enumerate(LinkedList<Edge> data, int n,
LinkedList<Edge> edges){
```

```
        if(edges.size() == n){
            LinkedList<Edge> e = new LinkedList<>(edges);
            if(!contains(e))
                enumerates.addLast(e);
            return true;
        }
        for(Edge e : data){
            if(!edges.contains(e)){
                edges.addLast(e);
                enumerate(data, n, edges);
                edges.remove(e);
            }
        }
        return  false;
    }

    public static void main(String[] args){
        LinkedList<Edge> edges = new LinkedList<>();
        edges.addLast(new Edge(1, 2));
        edges.addLast(new Edge(1, 3));
        edges.addLast(new Edge(1, 6));
        edges.addLast(new Edge(3, 2));
        edges.addLast(new Edge(3, 4));
        edges.addLast(new Edge(4, 5));
        edges.addLast(new Edge(4, 6));
        edges.addLast(new Edge(5, 6));
        Graph graph = new Graph(6, edges);

        LinkedList<Edge> newList = new LinkedList<>();
        double[] sum = new double[10];
        for(int n = 1; n <=8; n++){
            enumerates = new LinkedList<>();
            enumerate(edges, n, newList);
            for(LinkedList<Edge> es : enumerates) {
                Graph g = new Graph(6, es);
                if(g.findPath(2, 5)){
                    for (int i = 0; i<10;i++){
                        double p = ((double)i+1)/10.0;
                        sum[i] += graph.findP(g, p);
                    }
                }
            }

        }
        for(double d : sum)
            System.out.println(d);


    }
}
```