

Цель работы: получение практических навыков оценки надежности вычислительных сетей.

Задание:

1. Вычислить вероятность $P_{\text{св}}(x_i, x_j) = \rho_{ij}$ существования пути между заданной парой вершин x_i, x_j в графе \tilde{G} .
2. Построить зависимость $\rho_{ij}(p)$ вероятности существования пути в случайном графе от вероятности существования ребра.

Исходные данные: вариант 11.

Необходимы вычислить вероятность пути из вершины 3 в вершину 5 в графе, представленном на рисунке ниже (рисунок 1)

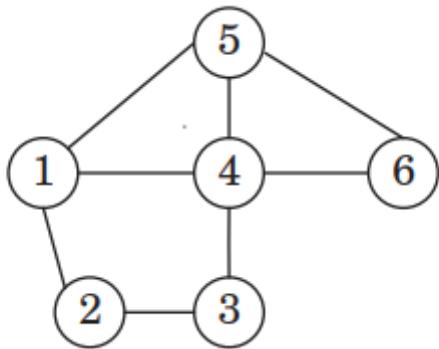
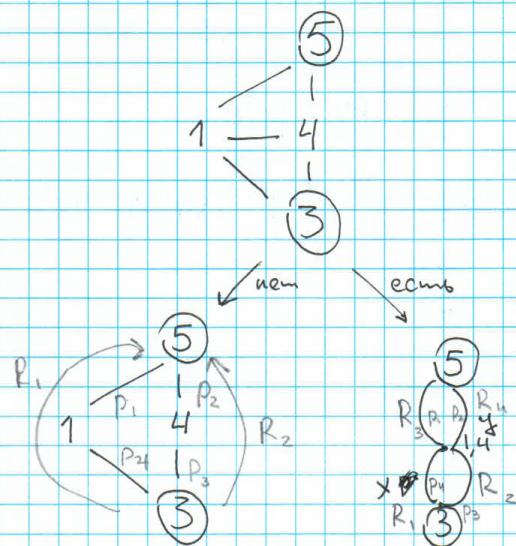
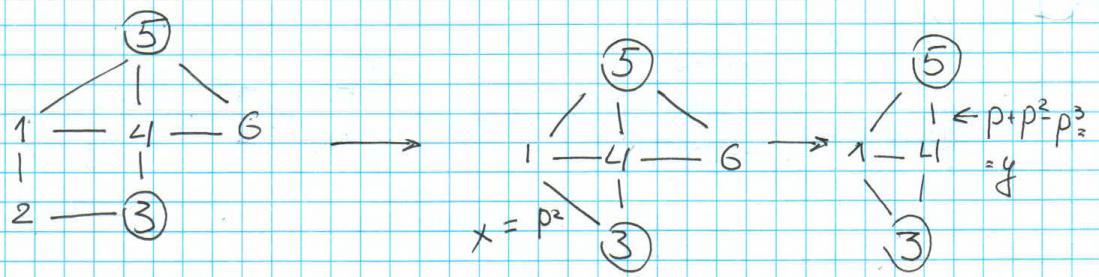


Рисунок 1. Исходный граф.

Ход работы

1) Необходимо вычислить вероятность существования пути в случайном графе, как функции от p . Для этого нужно произвести упрощение графа и вывести результирующую формулу (рисунок 2,3).

Ap S1



$$\begin{aligned}
 \text{I} \quad & \Pr \{ \text{нум} (3, 5) \mid \text{нум} (1, 3) \} = \Pr \{ R_1 \cup R_2 \} = \\
 & = \Pr \{ R_1 \} + \Pr \{ R_2 \} - \Pr \{ R_1 \} \cdot \Pr \{ R_2 \} = \\
 & = p_1 \cdot p_4 + p_2 \cdot p_3 - p_1 \cdot p_2 \cdot p_3 \cdot p_4 = p \cdot p^2 + p(p + p^2 - p^3) - \\
 & - p \cdot p^2 \cdot p(p + p^2 - p^3) = p^3 + p^2 + p^3 - p^4 - p^5 + p^7 - p^6 - \\
 & = p^2 + 2p^3 + p^2 - p^4 - p^5 - p^6
 \end{aligned}$$

$$\begin{aligned}
 \text{II} \quad & \Pr \{ \text{нум} (3, 5) \mid \text{если} (1, 4) \} = \Pr \{ R_1 \cup R_2 \} \cdot (R_3 \cup R_4) \\
 & = (p_4 + p_3 - p_1 \cdot p_3)(p_1 + p_2 - p_1 \cdot p_2) = (p^2 + p - p^3)(p + p + p^2 - p^3) \\
 & - p^2 - p^3 + p^4 = (p^2 + p - p^3)(2p + p^2 - 2p^3) =
 \end{aligned}$$

Рисунок 2. Упрощение графа

$$= 2p^3 + p^6 - 2p^5 + 2p^2 + p^5 - 2p^4 - 2p^1 - p^7 + 2p^6 =$$

$$= -p^7 + 3p^6 - p^5 - 4p^4 + 2p^3 + 2p^2$$

$$\Pr \{ \text{путь } (2, 5) \} = \textcircled{I} \cdot (1-p) + \textcircled{II} p =$$

$$= (p^7 + 2p^3 - p^2 - p^4 - p^5)(1-p) + p(-p^7 + 3p^6 - p^5 - 4p^4 + 2p^3 + 2p^2) =$$

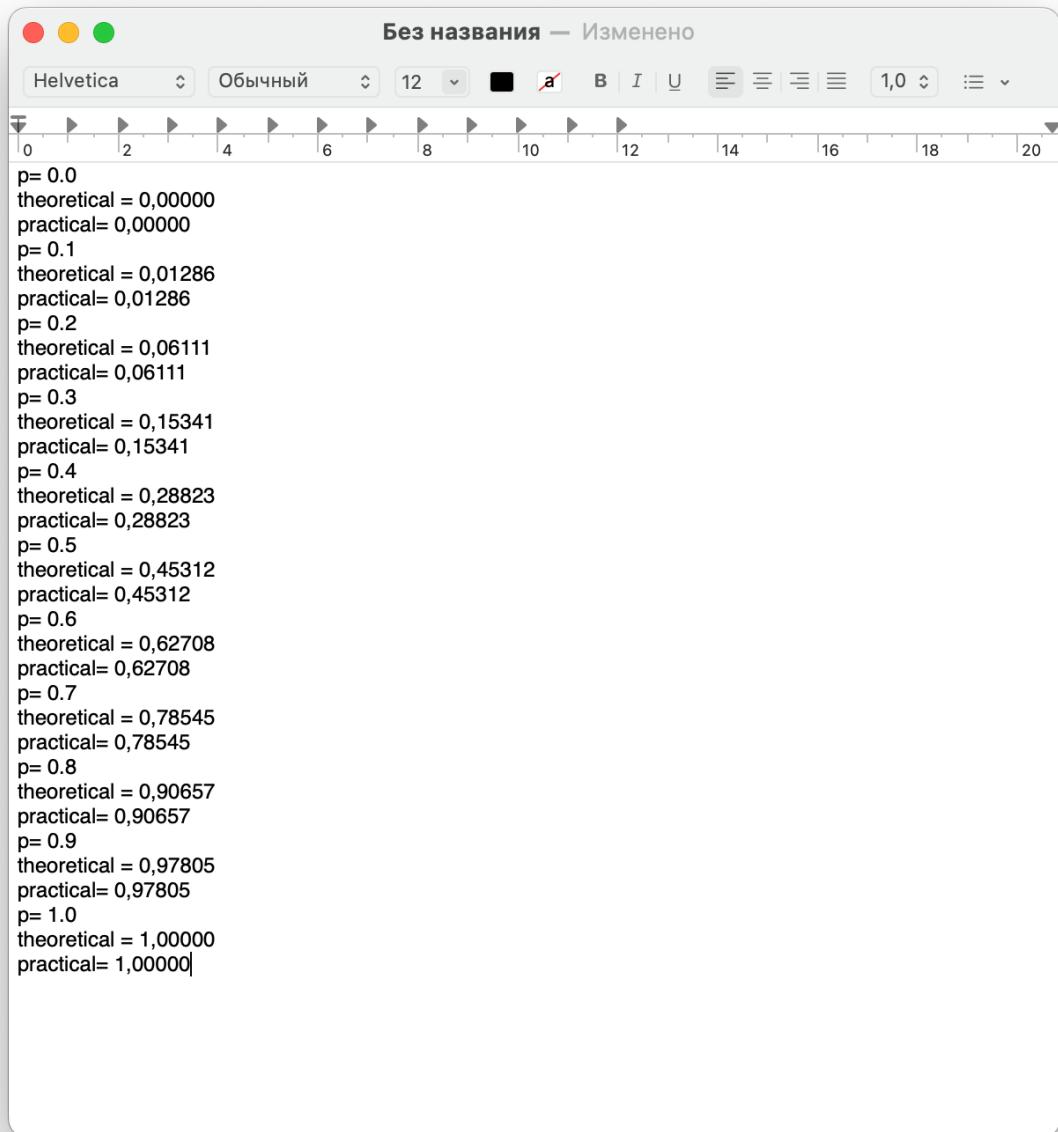
$$= p^7 + 2p^3 + p^2 - p^4 - p^5 - p^8 - 2p^9 - p^3 + p^5 + p^6 + (-p^8 + 3p^7 - p^6 - 4p^5 + 2p^4 + 2p^3) = -2p^8 + 5p^7 - p^6 - 4p^5 - p^4 + 3p^3 + p^2$$

Рисунок 3. Результирующая формула для вероятности пути

2) Разработка программы вычисления вероятности существования пути в случайном графе.

Задача программы состоит в том, чтобы обойти все подграфы и если в этом подграфе существует путь, то зачесть вероятность его появления в общую вероятность.

В программе граф задаётся в виде списка смежности. Перебор осуществляется по всем вероятностям существования рёбер от 0 до 1 с шагом 0.1 и всем подграфам, количество которых 2^L , где L – количество рёбер. Для нахождения пути в подграфе используется алгоритм DFS. Результат работы программы приведен на рисунке ниже (рисунок 4).



```
Без названия — Изменено
Helvetica 12
F 0 2 4 6 8 10 12 14 16 18 20
p= 0.0
theoretical = 0,00000
practical= 0,00000
p= 0.1
theoretical = 0,01286
practical= 0,01286
p= 0.2
theoretical = 0,06111
practical= 0,06111
p= 0.3
theoretical = 0,15341
practical= 0,15341
p= 0.4
theoretical = 0,28823
practical= 0,28823
p= 0.5
theoretical = 0,45312
practical= 0,45312
p= 0.6
theoretical = 0,62708
practical= 0,62708
p= 0.7
theoretical = 0,78545
practical= 0,78545
p= 0.8
theoretical = 0,90657
practical= 0,90657
p= 0.9
theoretical = 0,97805
practical= 0,97805
p= 1.0
theoretical = 1,00000
practical= 1,00000|
```

Рисунок 4. Результат работы программы

Из графиков (рисунок 5,6) видно, что значения вероятности, полученные при полном переборе и упрощении графа совпадают. При возрастании вероятности существования рёбер возрастает вероятность существования пути в обоих случаях .

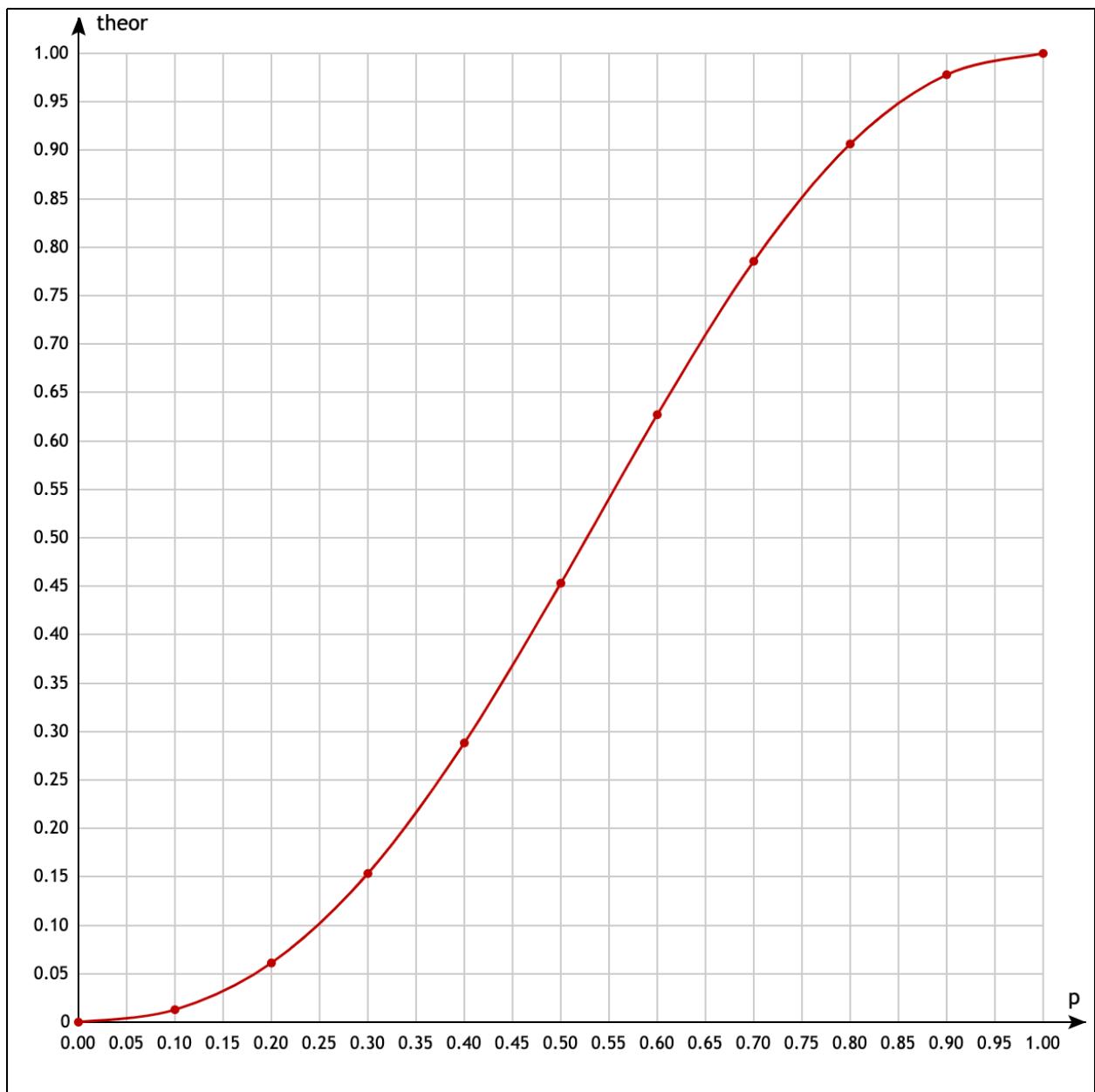


Рисунок 5. Зависимость вероятности существования пути от вероятности существования рёбер при подстановке значений в результирующую формулу

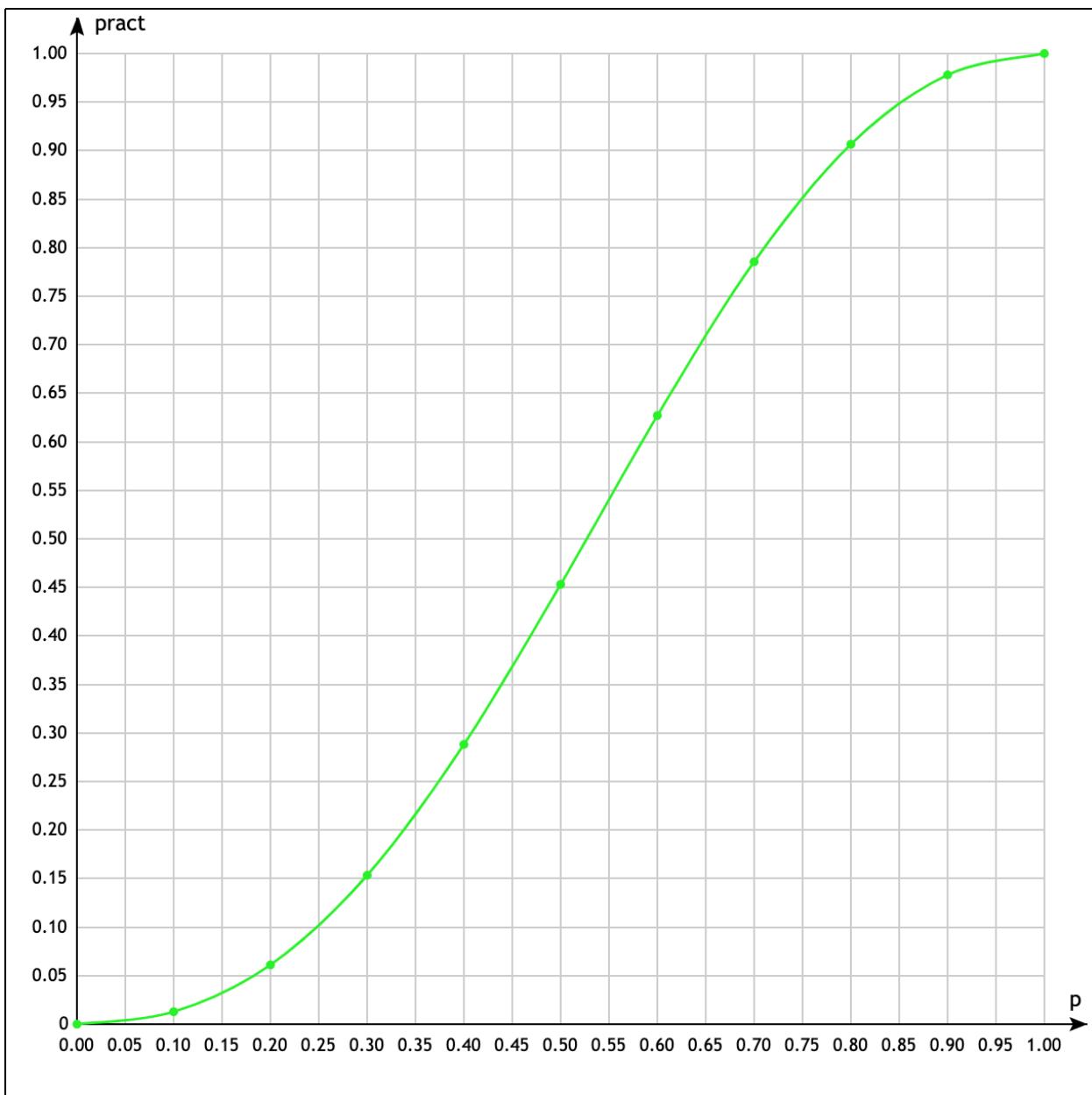


Рисунок 6. Зависимость вероятности существования пути от вероятности существования рёбер при полном переборе всех подграфов

Выводы:

В ходе выполнения лабораторной работы, были получены практические навыки оценки надежности вычислительных сетей, была вычислена вероятность существования пути в случайном графе, как функции от p , построена программа для полного перебора графа

Листинг

```
import java.text.DecimalFormat;
import java.util.LinkedList;

public class Main {
    public static void main(String args[]) {
        double p=0.1;
        int per = 1;
        int bin_code[] = new int[100];
        int flag ;
        double sum;
        double [] arrP = new double[11];
        arrP[0]=0.0;
        arrP[1]=0.1;
        arrP[2]=0.2;
        arrP[3]=0.3;
        arrP[4]=0.4;
        arrP[5]=0.5;
        arrP[6]=0.6;
        arrP[7]=0.7;
        arrP[8]=0.8;
        arrP[9]=0.9;
        arrP[10]=1;

        int s = 2, d = 4;
        Graph g = new Graph(6);

        g.addEdge(0, 1);
        g.addEdge(0, 3);
        g.addEdge(0, 4);

        g.addEdge(1, 0);
        g.addEdge(1, 2);

        g.addEdge(2, 1);
        g.addEdge(2, 3);

        g.addEdge(3, 0);
        g.addEdge(3, 2);
        g.addEdge(3, 4);
        g.addEdge(3, 5);

        g.addEdge(4, 0);
        g.addEdge(4, 3);
        g.addEdge(4, 5);

        g.addEdge(5, 3);
        g.addEdge(5, 4);

        // for (int k = 0; k <= 10 ; k++) {
        //     p = arrP[k];
        //     sum = 0.0;
        //     flag = 0;
        //     int nemM [][] = g.getMatrix();
        //     while (flag != Math.pow(2, 8) - 1) {

        //         per = 1;
        //         for (int i = 7; i >= 0; i--) {
        //             if (per == 1) {
```

```

                if (bin_code[i] == 1) {
                    bin_code[i] = 0;
                } else {
                    bin_code[i] = 1;
                    per = 0;
                }
            }
        }
        flag++;
        /*for (int i = 7; i >= 0; i--) {
            System.out.print(bin_code[i]);
        }
        System.out.print("\n");*/
    }

    int count = 0;
    Graph tmp = g.getTmpGraph(nemM, bin_code);
    if (tmp.DFS(s, d) == 1) {
        count = 0;
        for (int i = 7; i >= 0; i--) {
            if (bin_code[i] == 1) {
                count++;
            }
        }
        sum += Math.pow(p, count) * Math.pow(1 - p, 8
        - count);
    }
}

// System.out.println(g.DFS(s, d, p));
double tmp = -2 * Math.pow(p, 8) + 5 * Math.pow(p, 7) -
Math.pow(p, 6) - 4 * Math.pow(p, 5) - Math.pow(p, 4) + 3 * Math.pow(p, 3) +
Math.pow(p, 2);
String formattedDouble = new
DecimalFormat("#0.0000").format(p);
System.out.println("p= "+p);
String formattedDouble2 = new
DecimalFormat("#0.00000").format(tmp);
System.out.println("theoretical = " + formattedDouble2);
String formattedDouble3 = new
DecimalFormat("#0.00000").format(sum);
System.out.println("practical= "+formattedDouble3);

// }

}

import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;

public class Graph {
    private int V;
    private int e;
    public LinkedList<Integer>[] adj;

    Graph(int v) {
        V = v;

        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList<>();
        }
    }
}
```

```

}

void addEdge(int v, int w) {
    adj[v].add(w);
}

double countPaths(int u, int d, boolean visited[], double pathCount) {
    Iterator<Integer> i = adj[u].listIterator();
    int cout = 0;
    while (i.hasNext()) {
        cout++;
        int n = i.next();
        if (!visited[n]) {
            visited[n] = true;
            pathCount = countPaths(n, d, visited, pathCount);
        }
    }
}

if (u == d) {
    pathCount = 1;
}
return pathCount;
}

public int[][] getMatrix(){
    int matrix [][] = new int[V][V];
    for (int j = 0; j < V; j++) {
        Iterator<Integer> i = adj[j].listIterator();
        int count=0;
        while (i.hasNext()) {
            i.next();
            matrix[j][adj[j].get(count)] = 1;
            count++;
        }
    }
    return matrix;
}

public Graph getTmpGraph(int [][] matrix, int[]bit) {
    Graph tmpAdj = new Graph(V);
    int [][] tmpMatrix = new int[V+1][V+1];

    for (int i = 0; i < V; ++i) {
        tmpAdj.adj[i] = new LinkedList<>();
    }

    int count = 0;
    for (int i = 0; i < V; i++) {
        for(int j = i; j < V; j++) {

            if (matrix[i][j] == 1){
                tmpMatrix[i][j] = bit[count];
                tmpMatrix[j][i] = bit[count];
                count++;
            }
        }
    }

    for (int i = 0; i < V; i++) {
        for(int j = 0; j < V; j++) {
            if(tmpMatrix[i][j] == 1) {

```

```
                tmpAdj.adj[i].add(j);
            }
        }
    return tmpAdj;
}

double DFS(int s, int d) {
//  adj[d].clear();

    boolean visited[] = new boolean[V];
    Arrays.fill(visited, false);
    double pathCount = 0;
    int countVisited = 0;
    pathCount = countPaths(s, d, visited, pathCount);
    return pathCount;
}

}
```