ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

PROJECT – 3

"Dependence of Network Reliability on the reliability of Individual links "

Submitted By –
Siddharth Sundewar
2021177557
Sxs137330@utdallas.edu

CONTENTS

ABSTRACT

- This project is an experimental study to test reliability of a network based on individual link reliability. We have to develop an algorithm to find out the overall network reliability. In this case network reliability only depends on individual link reliability. Each of the links in the network may fail but nodes are always up. The system will be operational if the network topology is connected. We will develop our algorithm to find the overall network by using Exhaustive Enumeration method.

- Create an algorithm to compute the network reliability in the above described situation, using the method of exhaustive enumeration.

- To plot a graph between the obtained networks reliability with the corresponding increase of p in steps of 0.02.

- Flip the corresponding system condition. That is, if the system was up, change it to down, if it was down, change it to up

- To find the reliability of the system when the p value is kept fixed at 0.9 and for different values of k. This also needs to be plotted on a graph.

PSEUDO-CODE

Reliability_Network (G,p)

Input: Complete Graph, G

Link reliability, p

Output: Network Reliability, R

Initialization;

State <- Generate all state of the graph, G;

R <- 0;

For each statei ϵ State **do**

if G is connected **then**

Find reliability r of statei;

Increase R by r;

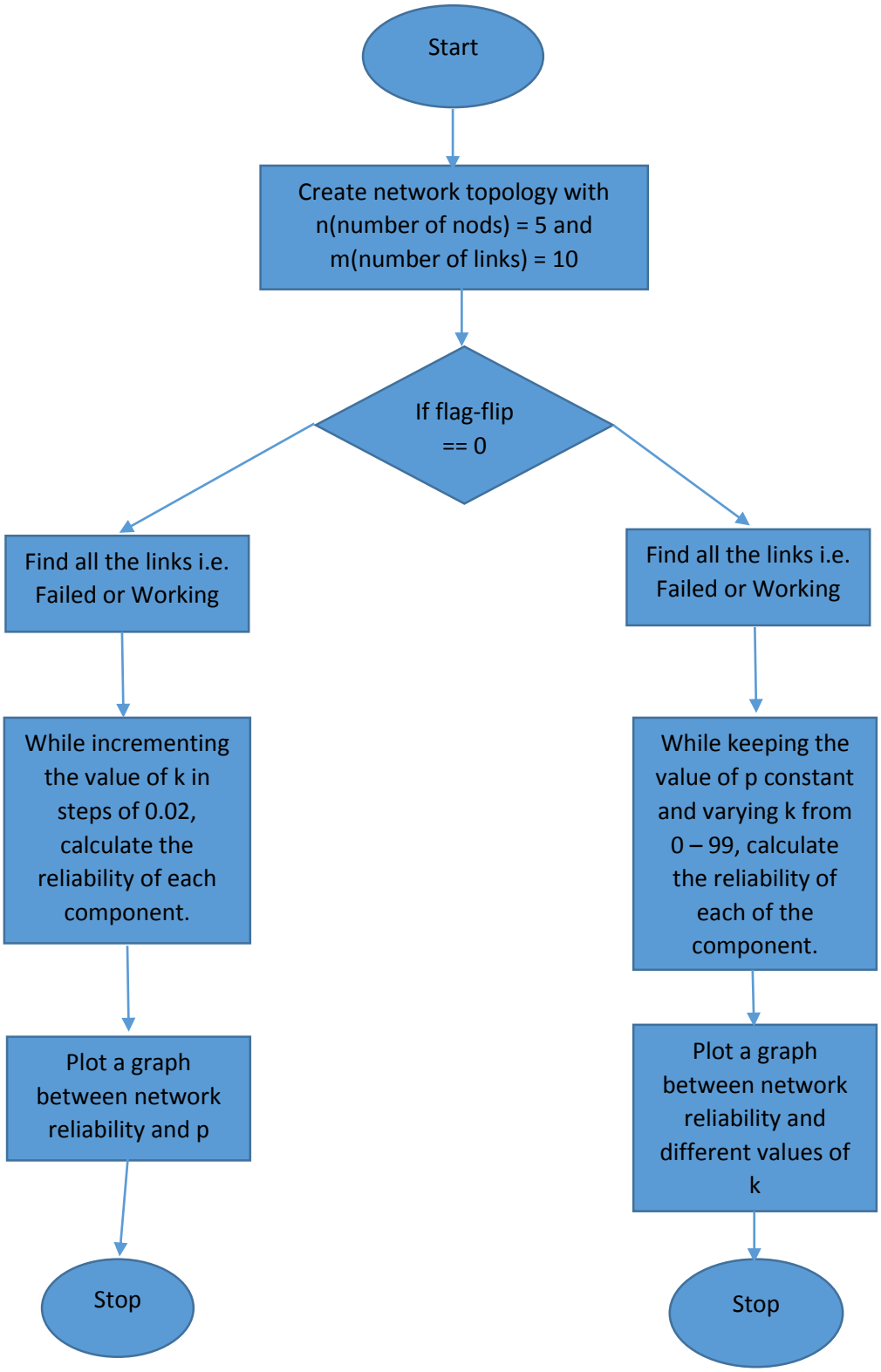
end

end

return R;

end procedure Reliability_Network

FLOWCHART


Start
Create network topology with n(number of nods) = 5 and m(number of links) = 10
If flag-flip == 0
Find all the links i.e. Failed or Working
While incrementing the value of k in steps of 0.02, calculate the reliability of each component.
Plot a graph between network reliability and p
Stop
Find all the links i.e. Failed or Working
While keeping the value of p constant and varying k from 0 – 99, calculate the reliability of each of the component.
Plot a graph between network reliability and different values of k
Stop

Exhaustive_enumeration Algorithm

1. Network is generated.
2. Determine whether a link is UP or DOWN.
3. IF path exists between two nodes i.e. Source and Destination

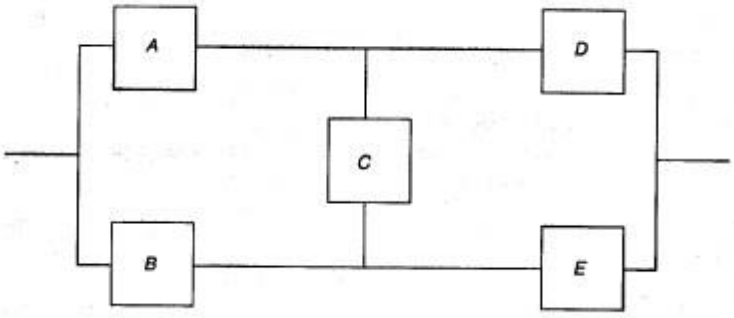
   Link is UP

   ELSE

   Link is DOWN
4. All combinations of state are generated (thus we would have 2^m-1 combinations for m links).
5. The reliability of the network is computed as the sum of these reliabilities whose states are considered to be UP.

Illustration of Exhaustive Enumeration :

For a sample network such as the one shown below, the combination of states is enumerated exhaustively as shown the table.



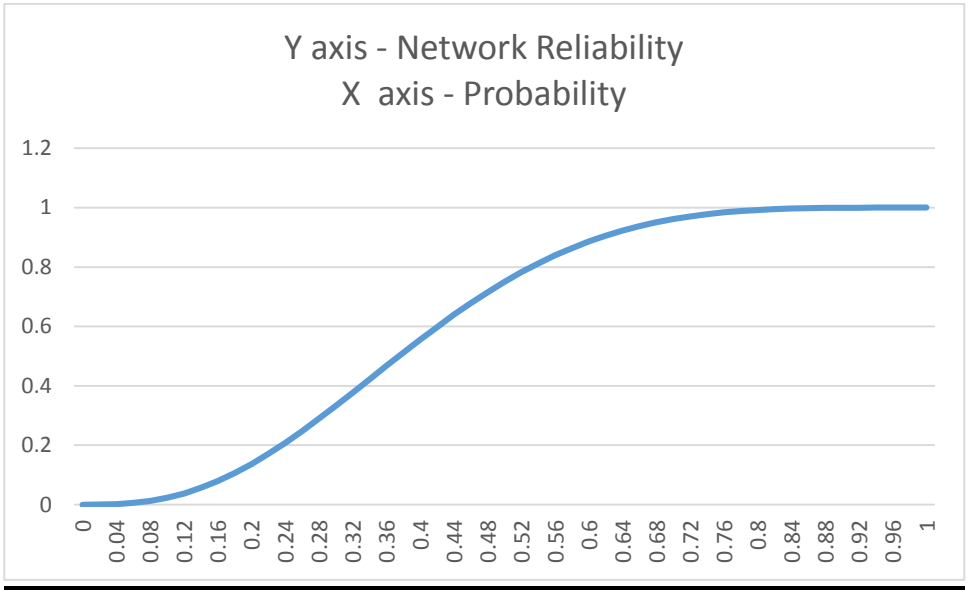
Number of Component Failures	Event	System Condition
0	1. $ABCDE$	Up
1	2. $\overline{A}BCDE$	Up
	3. $A\overline{B}CDE$	Up
	4. $AB\overline{C}DE$	Up
	5. $ABC\overline{D}E$	Up
	6. $ABCD\overline{E}$	Up
2	7. $\overline{A}\overline{B}CDE$	Down
	8. $\overline{A}B\overline{C}DE$	Up
	9. $\overline{A}BC\overline{D}E$	Up
	10. $\overline{A}BCD\overline{E}$	Up
	11. $AB\overline{C}DE$	Up
	12. $A\overline{B}C\overline{D}E$	Up
	13. $A\overline{B}CD\overline{E}$	Up
	14. $AB\overline{C}\overline{D}E$	Up
	15. $AB\overline{C}D\overline{E}$	Up
	16. $AB\overline{C}\overline{D}\overline{E}$	Down
3	17. $A\overline{B}\overline{C}DE$	Down
	18. $A\overline{B}C\overline{D}\overline{E}$	Down
	19. $A\overline{B}\overline{C}D\overline{E}$	Up
	20. $AB\overline{C}\overline{D}\overline{E}$	Down
	21. $\overline{A}BC\overline{D}\overline{E}$	Down
	22. $\overline{A}B\overline{C}D\overline{E}$	Down
	23. $\overline{A}B\overline{C}\overline{D}E$	Up
	24. $\overline{A}\overline{B}CD\overline{E}$	Down
	25. $\overline{A}\overline{B}C\overline{D}E$	Down
	26. $\overline{A}B\overline{C}\overline{D}E$	Down
4	27. $A\overline{B}\overline{C}\overline{D}E$	Down
	28. $\overline{A}B\overline{C}\overline{D}E$	Down
	29. $\overline{A}\overline{B}C\overline{D}E$	Down
	30. $\overline{A}\overline{B}D\overline{D}E$	Down
	31. $\overline{A}\overline{B}\overline{C}\overline{D}E$	Down

The reliability of the above could be computed by summing all the reliabilities of the "up states" combinations as follows:

$$
\begin{aligned}
R_{\text{network}} = {} & R_A R_B R_C R_D R_E + (1 - R_A) R_B R_C R_D R_E \\
& + R_A (1 - R_B) R_C R_D R_E + R_A R_B (1 - R_C) R_D R_E \\
& + R_A R_B R_C (1 - R_D) R_E + R_A R_B R_C R_D (1 - R_E) \\
& + (1 - R_A) R_B (1 - R_C) R_D R_E + (1 - R_A) R_B R_C (1 - R_D) R_E \\
& + (1 - R_A) R_B R_C R_D (1 - R_E) + R_A (1 - R_B)(1 - R_C) R_D R_E \\
& + R_A (1 - R_B) R_C (1 - R_D) R_E + R_A (1 - R_B) R_C R_D (1 - R_E) \\
& + R_A R_B (1 - R_C)(1 - R_D) R_E + R_A R_B (1 - R_C) R_D (1 - R_E) \\
& + R_A (1 - R_B)(1 - R_C) R_D (1 - R_E) \\
& + (1 - R_A) R_B (1 - R_C)(1 - R_D) R_E
\end{aligned}
$$

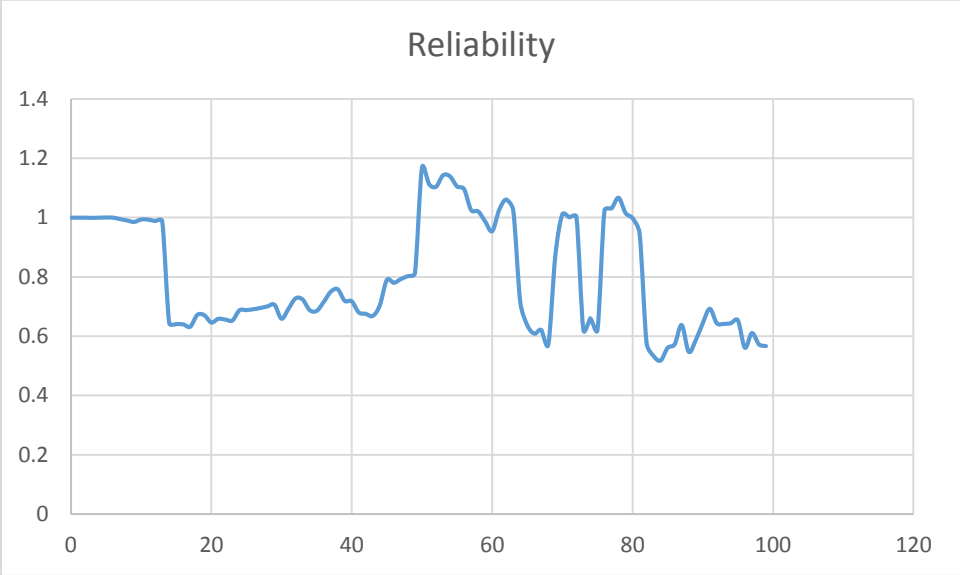
- The states are chosen randomly by Math.random class in all the possible combinations.

GRAPHS



Explanation –

The above graph shows as expected. X axis in the graph is Probability (p) i.e. reliability of each link. Now as the reliability of each link increases, the overall reliability of network should also increase theoretically, which is exactly what we see as a result in the above graph. When the value of p = 1, then value of Reliability is also = 1 which means when all the links are working then the overall reliability of the network will also be 1.



Explanation –

For the second question 'k' is given a value between 0 and 100. 'k' denotes the links whose states have to be flipped. If the link state is 'down' then that link will be included in computing 'R'. A link will not be included in the computation for 'R' if the state is 'up'. As the value 'p' increases the value '1-p' decreases. If a link is 'up' we use 'p' and if the link is 'down' we use '1-p' in calculating 'R'. The flipping of states decreases the value of 'R' as 'p' increases.

OUTPUT

Value of p is incremented with steps of 0.02

P	Network Reliability
0	0
0.02	2.28E-04
0.04	0.001733
0.06	0.005544
0.08	0.012441
0.1	0.022978
0.12	0.037499
0.14	0.05617
0.16	0.078994
0.18	0.105843
0.2	0.136471
0.22	0.170546
0.24	0.20766
0.26	0.247355
0.28	0.289138
0.3	0.332497
0.32	0.376916
0.34	0.421886
0.36	0.466917
0.38	0.511546
0.4	0.555346
0.42	0.597929
0.44	0.638955
0.46	0.67813
0.48	0.71521
0.5	0.75
0.52	0.782357
0.54	0.812184
0.56	0.83943
0.58	0.864086
0.6	0.886182
0.62	0.905784
0.64	0.922986
0.66	0.937911
0.68	0.9507
0.7	0.961514
0.72	0.970523
0.74	0.977906
0.76	0.983846
0.78	0.988526
0.8	0.992123

0.82	0.994811
0.84	0.996749
0.86	0.99809
0.88	0.998967
0.9	0.999501
0.92	0.999795
0.94	0.999935
0.96	0.999987
0.98	0.999999
1	1

P is fixed at 0.9 and k is incremented from 0 – 99 with steps of 1

K	Reliability
0	0.999501
1	0.999501
2	0.999449
3	0.998918
4	0.999449
5	1.000086
6	0.999608
7	0.994825
8	0.990043
9	0.985265
10	0.993395
11	0.993022
12	0.98808
13	0.987808
14	0.644023
15	0.640886
16	0.639882
17	0.631846
18	0.671638
19	0.670616
20	0.646011
21	0.658508
22	0.656023
23	0.652481
24	0.687122
25	0.687762
26	0.690843
27	0.694994
28	0.700582
29	0.705785
30	0.658812
31	0.692724
32	0.72732
33	0.723692
34	0.687943

35	0.685066
36	0.714881
37	0.749528
38	0.75813
39	0.719082
40	0.717708
41	0.680014
42	0.675173
43	0.668107
44	0.703967
45	0.789746
46	0.779923
47	0.792628
48	0.802659
49	0.811857
50	1.167814
51	1.113027
52	1.103666
53	1.142394
54	1.138783
55	1.104949
56	1.095876
57	1.025239
58	1.020927
59	0.987128
60	0.953967
61	1.025705
62	1.060662
63	1.02474
64	0.716184
65	0.636994
66	0.608466
67	0.620893
68	0.571845
69	0.872208
70	1.010746
71	1.0006
72	1.002981
73	0.619937
74	0.660483
75	0.620255
76	1.024047
77	1.030423
78	1.066607
79	1.01529
80	0.997476
81	0.946403
82	0.576641
83	0.532791
84	0.517799
85	0.56074

86	0.572601
87	0.637497
88	0.547238
89	0.586781
90	0.64369
91	0.692396
92	0.643411
93	0.641495
94	0.643667
95	0.653459
96	0.560985
97	0.610454
98	0.572456
99	0.566425

SOURCE CODE

```
public class Reliability_Network {
	private static double sum = 0;
	private static double Row_Probabaility = -1;
	private static int flip_flag = 0;
	private static final int Count_100 = 100;
	private static final int Count_1000 = 1000;
	private static final int Count_1024 = 1024; // total possible combinations of the
components
	private static final int Count_10 = 10;
	private static int[] random_Generator = new int[Count_100];
	private static int[] array = new int[Count_1000];
	private static int[][] edge_Matrix = new int[Count_1024][Count_10];
	private static int[][] Matrix_One = new int[Count_1024][Count_10];
	/*
	 * This method will be used to assign the edge or the connection between the two
	 * nodes as up or down
	 */
	public static int[][] SetLinks(int[][] link_Mat) {
		for (int i = 0; i < Count_1024; i++) {
			String currentString = String.format("%010d",
Integer.parseInt(Integer.toBinaryString(i)));
			for(int j = 0; j < 10; j++)
			{
				if(currentString.charAt(j) == '0')
				{
					link_Mat[i][j] = 0;
				}
				else
				{
					link_Mat[i][j] = 1;
				}
			}
		}
		return link_Mat;
	}


/*
 * This function will round off upto n places
 */
	public static double round(double value, int places) {

		if (places < 0)
		{
			throw new IllegalArgumentException();
		}

		long factor = (long) Math.pow(10, places);
		value = value * factor;
		long tmp = Math.round(value);
		return (double) tmp / factor;
	}

	/*
	 * This method will find if the nodes are up or down
	 */

	public static int[] UP_DOWN_Search(int[][] matrix) {
		int[] temp = new int[Count_1024];
```

```
int counter = 0;
for (int i = 0; i < Count_1024; i++)
{
	temp[i] = 0;
}
for (int i = 0; i < Count_1024; i++)
{
	int[] matrix_chk = new int[5];
	for (int k = 0; k < 5; k++)
	{
		matrix_chk[k] = -1;
	}
	if (matrix[i][0] == 1)
	{
		matrix_chk[0] = 0;
		matrix_chk[1] = 1;
	}
	if (matrix[i][1] == 1)
	{
		matrix_chk[0] = 0;
		matrix_chk[2] = 2;
	}
	if (matrix[i][2] == 1)
	{
		matrix_chk[0] = 0;
		matrix_chk[3] = 3;
	}
	if (matrix[i][3] == 1)
	{
		matrix_chk[0] = 0;
		matrix_chk[4] = 4;
	}
	if (matrix[i][4] == 1)
	{
		matrix_chk[1] = 1;
		matrix_chk[2] = 2;
	}
	if (matrix[i][5] == 1)
	{
		matrix_chk[1] = 1;
		matrix_chk[3] = 3;
	}
	if (matrix[i][6] == 1)
	{
		matrix_chk[1] = 1;
		matrix_chk[4] = 4;
	}
	if (matrix[i][7] == 1)
	{
		matrix_chk[2] = 2;
		matrix_chk[3] = 3;
	}
	if (matrix[i][8] == 1)
	{
		matrix_chk[2] = 2;
		matrix_chk[4] = 4;
	}
	if (matrix[i][9] == 1)
	{
		matrix_chk[3] = 3;
		matrix_chk[4] = 4;
	}
```

```
                int count = 0;

                for (int j = 0; j < 5; j++)
                {
                        count = count + matrix_chk[j];
                }

                if (count == Count_10)
                {
                        temp[counter] = i;
                        counter++;
                }
        }
        return temp;
 }
/*
 *
 */
public static void main(String[] args) {
        int k, i;

        for (i = 0; i < Count_100; i++)
        {
                random_Generator[i] = -1;
        }

        sum = 0;
        float q = (float) 1.0;
        float p = (float) 0.0;

        if (flip_flag == 0)
        {
                edge_Matrix = SetLinks(edge_Matrix);
        }

        array = UP_DOWN_Search(edge_Matrix);

        /*
         *    This case will handle variable p value which will be incremented in
steps
         *    of 0.02
         */

        if (flip_flag != 0)
        {
                p = (float) 0.9;
                q = (float) 0.9;
        }

        System.out.println("P\t\tReliability");

        for (double id = p; id <= q;)
        {
                for (int j = 0; j < Count_1000; j++) {  if (array[j] != 0)
                {
                        for (int ik = 0; ik < Count_10; ik++)
                        {
                                if (edge_Matrix[array[j]][ik] == 1)
                                {
                                        if (Row_Probabaility == -1)
                                        {
                                                Row_Probabaility = id;
                                        }
```

```
                                    else
                                    {
                                          Row_Probabaility = Row_Probabaility * id;
                                    }
                              }
                              else
                              {
                                    if (Row_Probabaility == -1)
                                    {
                                          Row_Probabaility = (1 - id);
                                    }
                                    else
                                    {
                                          Row_Probabaility = Row_Probabaility * (1- id);
                                    }
                              }
                        }
                        sum = sum + Row_Probabaility;
                        Row_Probabaility = -1;
                  }
            }
                  System.out.println(id + "\t\t" + sum);
                  id = round(id + 0.02, 2);
                  sum = 0;
      }

            Matrix_One = edge_Matrix;

/*
* This case will handle constant k value
*/
            flip_flag = 1;
            System.out.println("\n******** Constant Value of p *********\n");
            System.out.println("K\t\tReliability");
            for (k = 1; k <= Count_100; k++)
            {
                  for (i = 0; i < k; i++)
                        random_Generator[i] = (int) (Math.random() * Count_1024);
                  int j = 0;
                  for (i = 0; i < Count_100; i++) ;
                  for (i = 0; i < Count_100; i++)
                  {
                        if (random_Generator[i] != -1)
                        {
                              for (j = 0; j < Count_10; j++)
                              {
                                    if (edge_Matrix[random_Generator[i]][j] == 0)
                                    {
                                          edge_Matrix[random_Generator[i]][j] = 1;
                                    }
                                    else
                                    {
                                          edge_Matrix[random_Generator[i]][j] = 0;
                                    }
                              }
                        }
                  }
                  sum = 0;
                  q = (float) 1.0 * 1;
                  p = (float) 0.0 * 1;
                  if (flip_flag == 0)
                  {
                        edge_Matrix = SetLinks(edge_Matrix);
```

```
                }

                array = UP_DOWN_Search(edge_Matrix);

                if (flip_flag != 0)
                {
                      p = (float) 0.9 * 1;
                      q = (float) 0.9 * 1;
                }
                for (double di = p; di <= q;)
                {
                      for (j = 0; j < Count_1000; j++)
                      {
                            if (array[j] != 0)
                            {
                                  for (int ki = 0; ki < Count_10; ki++)
                                  {
                                        if (edge_Matrix[array[j]][ki] == 1)
                                        {
                                              if (Row_Probabaility == -1)
                                              {
                                                    Row_Probabaility = di;
                                              }
                                              else
                                              {
                                                    Row_Probabaility = Row_Probabaility
* di;
                                              }
                                        }
                                        else
                                        {
                                              if (Row_Probabaility == -1)
                                              {
                                                    Row_Probabaility = (1 - di);
                                              }
                                              else
                                              {
                                                    Row_Probabaility = Row_Probabaility
* (1 - di);
                                              }
                                        }
                                  }
                                  sum = sum + Row_Probabaility;
                                  Row_Probabaility = -1;
                            }
                      }

                      System.out.println(k + "\t\t" + sum);
                      di = round(di + 0.02, 2);
                      sum = 0;
                }

                edge_Matrix = Matrix_One;
                for (i = 0; i < k; i++)
                      random_Generator[i] = -1;
          }
     }
}
```

REFERENCES

- The formulae for computation of reliabilities was borrowed from material posted by Dr Andras Farago in his lecture notes
- Definition of terms was borrowed from www.wikipedia.org