# CS 6385.0l1 Algorithmic Aspects of Telecommunication Networks

# Project 2
# “Effect of Individual Link Reliabilities on Network Reliability”

## Summer 2014


**Report By,**
**Rahul Dhanendran.**
**(rxd123230)**

## Project Description :

The theme of this project is to study experimentally how the network reliability depends on the individual link reliabilities. In order to compute the network reliability as a numerical measure, the method of exhaustive enumeration is used.

## Network Description :

**Network topology:**

A complete undirected graph on n = 5 nodes. This means, every node is connected with every other one (parallel edges and self- loops are excluded in this graph). As a result, this graph has m = 10 edges, representing the links of the network.

**Components that may fail:**

The links of the network may fail, the nodes are always up. The reliability of each link is p, the same for every link. The parameter p will take different values in the experiments.

**Reliability configuration:**

The system is considered operational, if the network topology is connected.

## Inputs and Outputs :

Input is a complete graph with 5 vertices which is auto generated by the program. There will be 10 edges in the graph. All edges will be assigned a minimum weight(1.0) and have the same reliability measure of 'p' which ranges from [0-1].

Output is always a reliability value of the entire network calculated using exhaustive enumeration.

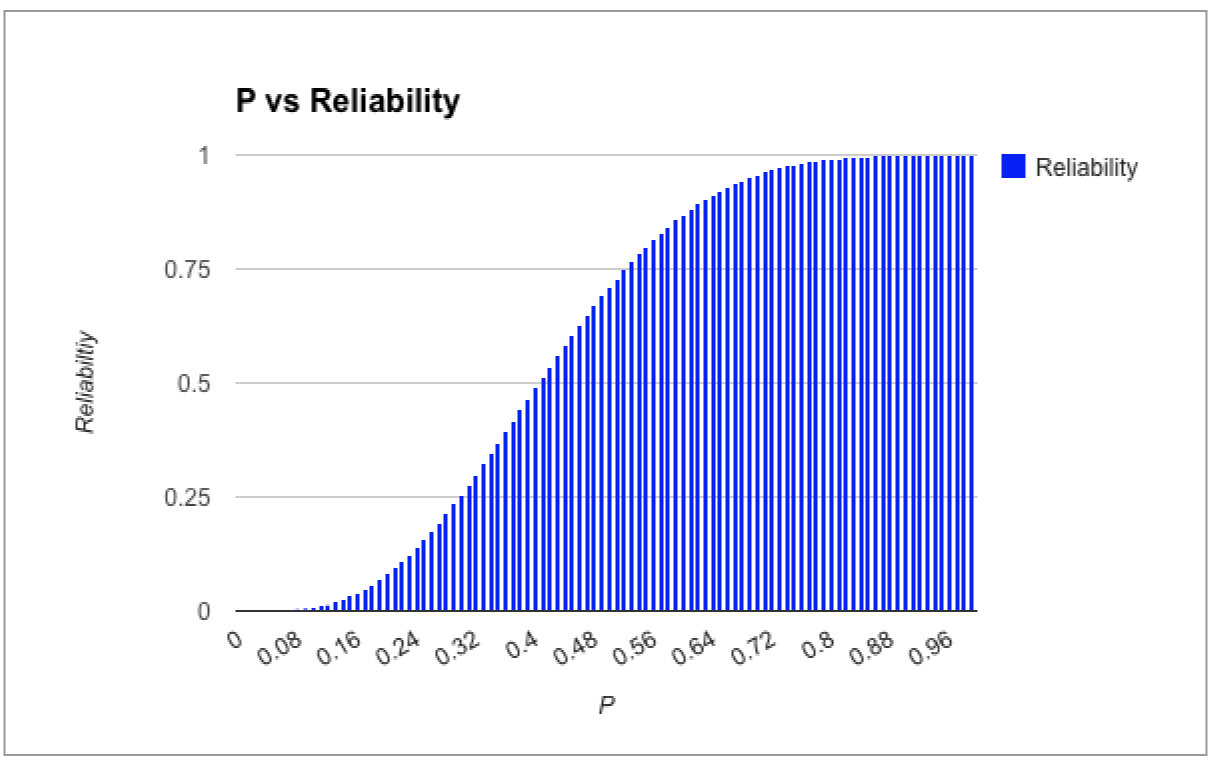
## Scenarios :

1. The first scenario is to run the program for different values of 'p'. The parameter 'p' runs over the [0,1] interval in steps of 0.01.
2. The second scenario is to fix the parameter 'p' at 0.95 and flip the system states of 'k' random combinations; thereby calculating the reliability of the network. The value of 'k' varies from 0 to 99. To reduce the effect of randomness, 100 runs are made for each 'k' value and averaged.

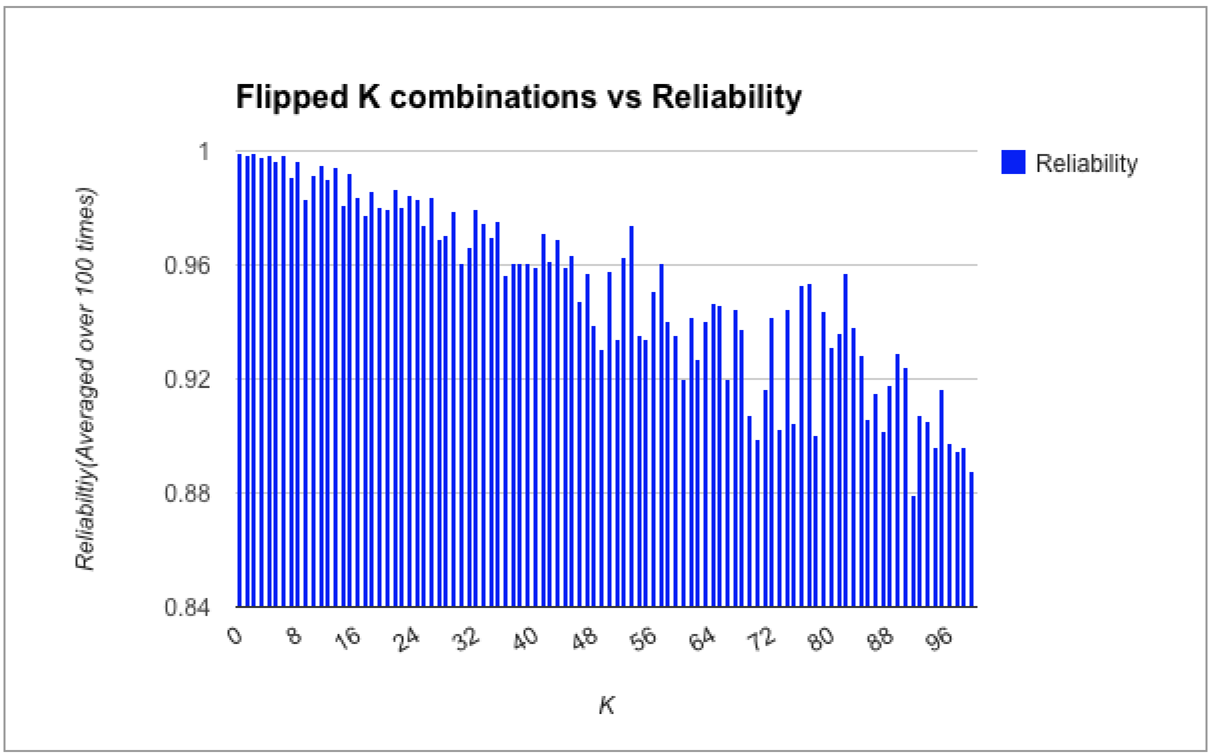
## Pseudocode :

1. Generate a complete graph with 5 vertices and assign proper edge weights and link reliabilities ‘p’.
2. Calculate all the possible combinations of link failures.
3. For each of the combinations, check the connectedness of the graph using Dijkstra’s algorithm[1]. If the graph is connected for a particular combination of link failures, then the system state is assigned to true. Otherwise, assign it to false meaning disconnected.
4. Using the method of exhaustive enumeration, calculate the network reliability for the combinations and the corresponding system states.
5. For scenario 1, repeat steps 1-4 for all values of ‘p’ in the [0,1] interval in steps of 0.01.
6. For scenario 2, do steps 1-3 once, assign k=0 and goto step 7.
7. Flip ‘k’ randomly chosen system states and calculate the network reliability for the combinations and the corresponding system states.
8. Flip back the same ‘k’ system states to original state.
9. Repeat steps 7-8 for 100 times for each value of ‘k’ from 0-99 and average it.
10. Plot graphs for scenario 1 and scenario 2.

## Graphs :



From the graph shown above, it can be clearly observed that the network reliability increases with increase in ‘p’ and stabilises towards the end. Intuitively, the probability of the network state to be ON is higher when there are lesser number of link failures. According to the method of exhaustive enumeration, if there are lesser number of failed links in a particulate combination, then there are lesser number of (1-p) terms and more of ‘p’ terms. So higher values of ‘p’ directly increases the value of network reliability.
Also from the graph, there is a slower increase in reliability for lesser values of ‘p’ from 0.00 to 0.2. After that, there is a steep increase till 0.7 and then it almost flattens.




From the graph above, it can be seen that increase in the value of ‘k’ decreases the reliability of the network. This is because, for a complete graph network of 5 vertices, there can be 1024 possible link failure combinations, out of which more than 70% have the system state as true. Therefore, flipping a few of the 1024 states may not affect the reliability much, but flipping 100 of the 1024 will definitely reduce the network reliability because there will be more number of TRUE states that will become FALSE now. And since majority of the states were TRUE and now many of them have become FALSE, the network reliability drops.

## Results:

### Scenario 1:

Reliability when p = 0.0 : 0.0
Reliability when p = 0.01 : 0.0
Reliability when p = 0.02 : 0.0
Reliability when p = 0.03 : 0.0
Reliability when p = 0.04 : 0.0
Reliability when p = 0.05 : 0.001
Reliability when p = 0.06 : 0.001
Reliability when p = 0.07 : 0.002
Reliability when p = 0.08 : 0.004
Reliability when p = 0.09 : 0.006
Reliability when p = 0.1 : 0.008
Reliability when p = 0.11 : 0.011
Reliability when p = 0.12 : 0.015
Reliability when p = 0.13 : 0.02
Reliability when p = 0.14 : 0.026
Reliability when p = 0.15 : 0.033
Reliability when p = 0.16 : 0.04
Reliability when p = 0.17 : 0.049
Reliability when p = 0.18 : 0.059
Reliability when p = 0.19 : 0.07
Reliability when p = 0.2 : 0.082
Reliability when p = 0.21 : 0.095
Reliability when p = 0.22 : 0.109
Reliability when p = 0.23 : 0.124
Reliability when p = 0.24 : 0.141
Reliability when p = 0.25 : 0.158
Reliability when p = 0.26 : 0.176
Reliability when p = 0.27 : 0.195
Reliability when p = 0.28 : 0.215
Reliability when p = 0.29 : 0.235
Reliability when p = 0.3 : 0.256
Reliability when p = 0.31 : 0.278
Reliability when p = 0.32 : 0.3
Reliability when p = 0.33 : 0.323
Reliability when p = 0.34 : 0.347
Reliability when p = 0.35 : 0.37
Reliability when p = 0.36 : 0.394

Reliability when p = 0.37 : 0.418
Reliability when p = 0.38 : 0.442
Reliability when p = 0.39 : 0.466
Reliability when p = 0.4 : 0.49
Reliability when p = 0.41 : 0.513
Reliability when p = 0.42 : 0.537
Reliability when p = 0.43 : 0.56
Reliability when p = 0.44 : 0.583
Reliability when p = 0.45 : 0.606
Reliability when p = 0.46 : 0.628
Reliability when p = 0.47 : 0.65
Reliability when p = 0.48 : 0.671
Reliability when p = 0.49 : 0.691
Reliability when p = 0.5 : 0.711
Reliability when p = 0.51 : 0.73
Reliability when p = 0.52 : 0.749
Reliability when p = 0.53 : 0.766
Reliability when p = 0.54 : 0.784
Reliability when p = 0.55 : 0.8
Reliability when p = 0.56 : 0.815
Reliability when p = 0.57 : 0.83
Reliability when p = 0.58 : 0.844
Reliability when p = 0.59 : 0.858
Reliability when p = 0.6 : 0.87
Reliability when p = 0.61 : 0.882
Reliability when p = 0.62 : 0.893
Reliability when p = 0.63 : 0.904
Reliability when p = 0.64 : 0.913
Reliability when p = 0.65 : 0.922
Reliability when p = 0.66 : 0.93
Reliability when p = 0.67 : 0.938
Reliability when p = 0.68 : 0.945
Reliability when p = 0.69 : 0.952
Reliability when p = 0.7 : 0.958
Reliability when p = 0.71 : 0.963
Reliability when p = 0.72 : 0.968
Reliability when p = 0.73 : 0.972
Reliability when p = 0.74 : 0.976
Reliability when p = 0.75 : 0.979

Reliability when p = 0.76 : 0.983
Reliability when p = 0.77 : 0.985
Reliability when p = 0.78 : 0.988
Reliability when p = 0.79 : 0.99
Reliability when p = 0.8 : 0.992
Reliability when p = 0.81 : 0.993
Reliability when p = 0.82 : 0.995
Reliability when p = 0.83 : 0.996
Reliability when p = 0.84 : 0.997
Reliability when p = 0.85 : 0.997
Reliability when p = 0.86 : 0.998
Reliability when p = 0.87 : 0.999
Reliability when p = 0.88 : 0.999
Reliability when p = 0.89 : 0.999
Reliability when p = 0.9 : 0.999
Reliability when p = 0.91 : 1.0
Reliability when p = 0.92 : 1.0
Reliability when p = 0.93 : 1.0
Reliability when p = 0.94 : 1.0
Reliability when p = 0.95 : 1.0
Reliability when p = 0.96 : 1.0
Reliability when p = 0.97 : 1.0
Reliability when p = 0.98 : 1.0
Reliability when p = 0.99 : 1.0
Reliability when p = 1 : 1.0

**Scenario 2 :**
Reliability when p=0.95 and k=0 is 0.9999
Reliability when p=0.95 and k=1 is 0.9996
Reliability when p=0.95 and k=2 is 0.9991
Reliability when p=0.95 and k=3 is 0.9975
Reliability when p=0.95 and k=4 is 0.9977
Reliability when p=0.95 and k=5 is 0.9979
Reliability when p=0.95 and k=6 is 0.9984
Reliability when p=0.95 and k=7 is 0.985
Reliability when p=0.95 and k=8 is 0.9906
Reliability when p=0.95 and k=9 is 0.9976
Reliability when p=0.95 and k=10 is 0.995
Reliability when p=0.95 and k=11 is 0.9782

Reliability when p=0.95 and k=12 is 0.9955
Reliability when p=0.95 and k=13 is 0.9871
Reliability when p=0.95 and k=14 is 0.9935
Reliability when p=0.95 and k=15 is 0.993
Reliability when p=0.95 and k=16 is 0.9741
Reliability when p=0.95 and k=17 is 0.986
Reliability when p=0.95 and k=18 is 0.9854
Reliability when p=0.95 and k=19 is 0.9932
Reliability when p=0.95 and k=20 is 0.9775
Reliability when p=0.95 and k=21 is 0.969
Reliability when p=0.95 and k=22 is 0.9802
Reliability when p=0.95 and k=23 is 0.9842
Reliability when p=0.95 and k=24 is 0.9767
Reliability when p=0.95 and k=25 is 0.9793
Reliability when p=0.95 and k=26 is 0.9709
Reliability when p=0.95 and k=27 is 0.9782
Reliability when p=0.95 and k=28 is 0.9819
Reliability when p=0.95 and k=29 is 0.9773
Reliability when p=0.95 and k=30 is 0.9776
Reliability when p=0.95 and k=31 is 0.977
Reliability when p=0.95 and k=32 is 0.9764
Reliability when p=0.95 and k=33 is 0.975
Reliability when p=0.95 and k=34 is 0.9625
Reliability when p=0.95 and k=35 is 0.9725
Reliability when p=0.95 and k=36 is 0.9675
Reliability when p=0.95 and k=37 is 0.9786
Reliability when p=0.95 and k=38 is 0.9522
Reliability when p=0.95 and k=39 is 0.9617
Reliability when p=0.95 and k=40 is 0.937
Reliability when p=0.95 and k=41 is 0.9575
Reliability when p=0.95 and k=42 is 0.9499
Reliability when p=0.95 and k=43 is 0.961
Reliability when p=0.95 and k=44 is 0.9444
Reliability when p=0.95 and k=45 is 0.9567
Reliability when p=0.95 and k=46 is 0.9651
Reliability when p=0.95 and k=47 is 0.9683
Reliability when p=0.95 and k=48 is 0.9569
Reliability when p=0.95 and k=49 is 0.9674
Reliability when p=0.95 and k=50 is 0.9181

Reliability when p=0.95 and k=51 is 0.9602
Reliability when p=0.95 and k=52 is 0.9478
Reliability when p=0.95 and k=53 is 0.9491
Reliability when p=0.95 and k=54 is 0.9596
Reliability when p=0.95 and k=55 is 0.9517
Reliability when p=0.95 and k=56 is 0.9473
Reliability when p=0.95 and k=57 is 0.9192
Reliability when p=0.95 and k=58 is 0.9666
Reliability when p=0.95 and k=59 is 0.9424
Reliability when p=0.95 and k=60 is 0.9392
Reliability when p=0.95 and k=61 is 0.9229
Reliability when p=0.95 and k=62 is 0.9311
Reliability when p=0.95 and k=63 is 0.9553
Reliability when p=0.95 and k=64 is 0.9317
Reliability when p=0.95 and k=65 is 0.9443
Reliability when p=0.95 and k=66 is 0.9251
Reliability when p=0.95 and k=67 is 0.9367
Reliability when p=0.95 and k=68 is 0.9329
Reliability when p=0.95 and k=69 is 0.9074
Reliability when p=0.95 and k=70 is 0.9091
Reliability when p=0.95 and k=71 is 0.9196
Reliability when p=0.95 and k=72 is 0.9529
Reliability when p=0.95 and k=73 is 0.9598
Reliability when p=0.95 and k=74 is 0.9368
Reliability when p=0.95 and k=75 is 0.9163
Reliability when p=0.95 and k=76 is 0.9524
Reliability when p=0.95 and k=77 is 0.933
Reliability when p=0.95 and k=78 is 0.9068
Reliability when p=0.95 and k=79 is 0.9335
Reliability when p=0.95 and k=80 is 0.931
Reliability when p=0.95 and k=81 is 0.9191
Reliability when p=0.95 and k=82 is 0.9087
Reliability when p=0.95 and k=83 is 0.922
Reliability when p=0.95 and k=84 is 0.9456
Reliability when p=0.95 and k=85 is 0.9337
Reliability when p=0.95 and k=86 is 0.9054
Reliability when p=0.95 and k=87 is 0.9137
Reliability when p=0.95 and k=88 is 0.9127
Reliability when p=0.95 and k=89 is 0.9259

Reliability when p=0.95 and k=90 is 0.8986
Reliability when p=0.95 and k=91 is 0.8925
Reliability when p=0.95 and k=92 is 0.9428
Reliability when p=0.95 and k=93 is 0.8815
Reliability when p=0.95 and k=94 is 0.8962
Reliability when p=0.95 and k=95 is 0.9113
Reliability when p=0.95 and k=96 is 0.875
Reliability when p=0.95 and k=97 is 0.9223
Reliability when p=0.95 and k=98 is 0.8763
Reliability when p=0.95 and k=99 is 0.9131

## Conclusion :

Thus from the experiments, two results can be concluded.

1. Higher the reliability of individual links, higher the network reliability.
2. Flipping more number of system states will decrease the network reliability.

# Appendix:

## Source Code:

## Graph.java

```
//Contains all the classes for creating a graph(vertices, edges) and Dijkstra's algorithm
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.PriorityQueue;
/**
 *
 * @author rahulrdhanendran
 */

//This class creates a complete graph for the given number of vertices
public class Graph {
    public ArrayList<Vertex> vertices;
    public ArrayList<Edge> edges;

    public Graph(int n, double p)
    {
        vertices = new ArrayList<>();
        edges = new ArrayList<>();
        for(int i=0; i<n ; i++)
        {
            Vertex v = new Vertex("V"+i);
            v.incidentEdges = new ArrayList<>();
            vertices.add(v);
        }
        for(int i=0; i<n ; i++)
        {
            for(int j=0;j<n;j++)
            {
                if(i==j)
                    continue;
                Edge e=new Edge(vertices.get(i),vertices.get(j),p,1);

         if(!edges.contains(e))
                {
                    edges.add(e);
                }
                vertices.get(i).incidentEdges.add(e);
            }
        }
    }

    public void computePaths(Vertex source)
```

```
    {
        source.minDistance = 0;
        PriorityQueue<Vertex> vertexQueue = new PriorityQueue<Vertex>();
            vertexQueue.add(source);

            while (!vertexQueue.isEmpty()) {
                Vertex u = vertexQueue.poll();

            // Visit each edge exiting u
            for (Edge e : u.incidentEdges)
            {
                Vertex v = e.vertexB;
                double weight = e.weight;
                double distanceThroughU = u.minDistance + weight;
                        if (distanceThroughU < v.minDistance) {
                            vertexQueue.remove(v);
                            v.minDistance = distanceThroughU ;
                            v.previous = u;
                            vertexQueue.add(v);
                        }
            }
        }
    }

    public List<Vertex> getShortestPathTo(Vertex target)
    {
        List<Vertex> path = new ArrayList<Vertex>();
        for (Vertex vertex = target; vertex != null; vertex = vertex.previous)
            path.add(vertex);
        Collections.reverse(path);
        return path;
    }
}

class Vertex implements Comparable<Vertex>
{
    public final String name;
    public ArrayList<Edge> incidentEdges;
    public double minDistance = Double.POSITIVE_INFINITY;
    public Vertex previous;
    public Vertex(int n)
    {
        name = Integer.toString(n);
    }
    public Vertex(String argName) { name = argName; }
    public String toString() { return name; }
    public int compareTo(Vertex other)
    {
```

```
        return Double.compare(minDistance, other.minDistance);
    }
}

class Edge
{
    public final Vertex vertexA;
    public final Vertex vertexB;
    public final double weight;
    public final double probability;
    public Edge(Vertex a,Vertex b,double p, double argWeight)
    {
        vertexA = a;
        vertexB = b;
        weight = argWeight;
        probability = p;
    }

    @Override
    public boolean equals(Object o) {
        Edge o1 = (Edge) o;
if((o1.vertexA==this.vertexA && o1.vertexB==this.vertexB) || (o1.vertexA==this.vertexB && o1.vertexB==this.vertexA) )
            return true;

    return false;
    }

    @Override
    public int hashCode(){
        return 1;
    }
}
```

**MainClass.java**

```
//responsible for implementing the pseudocode mentioned earlier
import java.util.*;
/**
 * @author rahulrdhanendran
 */

public class MainClass {

    public HashSet<String> combinations;
    public ArrayList<TableEntry> table;

    public MainClass()
    {
```

```
    table = new ArrayList<>();
    combinations = new HashSet<>();
}

public void getAllCombinations(int []array, int length, String curr)
{
    if(curr.length() == length) {
        char[] chars = curr.toCharArray();
        Arrays.sort(chars);
        combinations.add(new String(chars));
    } else {
        for(int i = 0; i < array.length; i++) {
            String oldCurr = curr;
            if(curr.contains(Character.toString(Character.forDigit(array[i],10))))
                continue;
            curr += array[i];
            getAllCombinations(array,length,curr);
            curr = oldCurr;
        }
    }
}

public double getReliability(double prob, int scenario)
{
    table.clear();
    int a[] = new int[10];

    for(int i=0;i<10;i++)
    {
        a[i]=i;
    }
    for(int i=0;i<10;i++)
    {
        combinations.clear();
        getAllCombinations(a, i , "");
        if(combinations.size()==1)
        {
            table.add(new TableEntry());
        }
        else
        {
            for(String s : combinations)
            {
                Graph g = new Graph(5,prob);
                TableEntry t = new TableEntry();
                char badEdge[] = s.toCharArray();
                for(int j=0;j<s.length();j++)
                {
```

```
                        int index = Character.getNumericValue(badEdge[j]);
                        t.edgeNumber[index] = 0; //set as bad edge
                        Edge e = g.edges.get(index);

g.vertices.get(Character.getNumericValue(e.vertexA.name.charAt(1))).incidentEdges.remove(e);
g.vertices.get(Character.getNumericValue(e.vertexB.name.charAt(1))).incidentEdges.remove(e);
                    }
                    g.computePaths(g.vertices.get(0));
                    for(int k=0;k<g.vertices.size();k++)
                    {
                        List<Vertex> path = g.getShortestPathTo(g.vertices.get(k));
                        if(path.get(path.size()-1).minDistance == Double.POSITIVE_INFINITY)
                        {
                            t.state = false;
                            break;
                        }
                    }
                    table.add(t);
                }
            }
        }
        TableEntry last = new TableEntry();
        last.state = false;
        for(int m=0;m<10;m++)
        {
            last.edgeNumber[m]=0;
        }
        table.add(last);

        if(scenario == 2)
        {
            flipAndCalculate(table,prob);
        }
        return calculateReliability(table, prob);
    }

    public void flipAndCalculate(ArrayList<TableEntry> t, double prob)
    {
        for(int k=0;k<100;k++)
        {
            double reliability = 0.0;
            for(int n=0;n<100;n++)
            {
                HashSet<Integer> randIndices = getRandomIndices(k);
                //flip
                for(int index : randIndices)
                {
                    t.get(index).state = !t.get(index).state;
```

```
            }
            //calculate
            reliability += calculateReliability(t, prob);
            //flip it back to original
            for(int index : randIndices)
            {
                t.get(index).state = !t.get(index).state;
            }
        }
                                        //System.out.println("Reliability    when    p=0.95    and    k="+k+"    is
"+(double)Math.round(reliability/50.0*10000)/10000);
        System.out.println((double)Math.round(reliability/100.0*10000)/10000);
    }

    System.exit(0);
}

public HashSet getRandomIndices(int i)
{
    HashSet<Integer> hs = new HashSet<Integer>();
    Random rand = new Random();
    do
    {
        int index = rand.nextInt(1024);
        if(!hs.contains(new Integer(index)))
            hs.add(new Integer(index));
    }while(hs.size()<i);
    return hs;
}

public double calculateReliability(ArrayList<TableEntry> t, double prob)
{
    double reliability = 0.0;
    for(TableEntry te : t)
    {
        if(te.state)
        {
            double product=1.0;
            for(int i=0;i<10;i++)
            {
                if(te.edgeNumber[i]==0)
                 product = product*(1-prob);
                else
                 product = product*prob;
            }
            reliability += product;
        }
    }
```

```
        return reliability;
    }

    public static void main(String[] args) {
        MainClass m = new MainClass();
        for(double i=0.0;i<=1;i+=0.01)
        {
                                //System.out.println("Reliability when p = "+(double)Math.round(i*100)/100+" : "+(double)Math.round(m.getReliability((double)Math.round(i*100)/100,1)*1000)/1000);
System.out.println((double)Math.round(m.getReliability((double)Math.round(i*100)/100,1)*1000)/1000);
        }

        //scenario 2
        m.getReliability(0.95, 2);
      }
}

class TableEntry{
    int edgeNumber[] = new int[10];
    boolean state;
    //default: all edges are good and the system is up.
    public TableEntry()
    {
        state = true;
        for(int i=0;i<10;i++)
        {
            edgeNumber[i]=1;
        }
    }
}
```

## References :


- Dijkstra’s Algorithm : http://www.algolist.com/code/java/Dijkstra's_algorithm