# PROJECT NO – 2
# NETWORK RELIABILITY

CS 6385

ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

SUBMITTED BY:

SANJANA MAJETI

SXM220015

# CONTENTS

# OBJECTIVE

The objective of the project is to explore the intricate relationship between network reliability and the reliabilities of individual components within a specific network topology. Emphasizing the analysis of the network's connectivity, the investigation focuses on the repercussions of link failures, each characterized by an independent probability of failure (referred to as 'p'). Employing systematic variations of the parameter 'p' in carefully designed experiments, the primary goal is to quantify the precise impact of link failures on the overall network reliability and gain valuable insights into critical thresholds that trigger the transition from a connected to a disconnected state.

Utilizing the exhaustive enumeration algorithm, this study comprehensively evaluates network reliability by considering all possible scenarios of network failure. Rather than relying on approximations or simplifications, this meticulous approach systematically examines each and every potential combination of component failures, providing a thorough and precise assessment of the network's overall reliability. By exhaustively exploring all possibilities, this methodology ensures a comprehensive understanding of the system's resilience and facilitates informed decision-making for enhancing network robustness. We'll be dealing with the network topology shown below for this project.

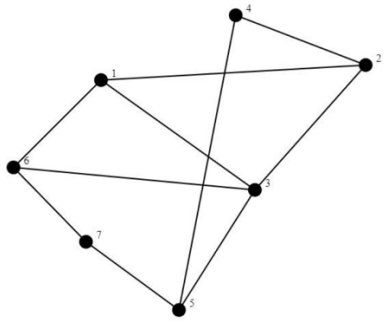



**Fig-1: Network Topology**

# Algorithm

**Step-1:** Find the adjacency matrix for the network topology shown in fig-1.

**Step-2:** Since links are the components that can be failed as given in the project statement. We label the links which will help in finding the combinations.

**Step-3:** Now, generate the combinations of link failures. There are 10 links in our network topology, so we will have 2^10 =1024 combinations and store them in a list.

**Step-4:** Each combination will have list of nodes that are to be deleted. So, for each combination we find the adjacency matrix.

**Step-5:** Now, for the adjacency matrix of each combination we find the graph connectivity.

1. Graph connectivity is found using DFS, create a function with adjacency matrix as input.
2. Create a boolean array of number of nodes to keep track of visited nodes. Initially, all the nodes are unvisited.
3. Create another function and give adjacency matrix, starting node and the boolean array that stores visited. This function is a recursive function used to perform DFS.
4. This will iterate over all the nodes and update the visited nodes array. And in the next iteration it will go to the node that is not visited.
5. Now, once the DFS returns then we check the visited nodes array and if all the indexes are set to “true”. Then, we return a boolean “true” saying the graph is connected.

**Step-6:** We store all the combinations that return “true” saying it is connected in a list.

**Step-7:** Finally, for every combination in the list of connected combinations we add the reliabilities.

Reliability + = (10-len(each_combination))*p + len(each_combination)*(1-p)

**Step-8:** Repeat the above process for different values of p ranging from 0.05 to 1 in intervals of 0.05. And store the values of reliability, which will be used for visualizations.

**Pseudo code:**

```
function getCombinations(Links_data):
  // Initialize an empty list to store all combinations
  allCombinations = new List of Lists of Strings
  // Get the total number of Links_data
  numLinks = length of Links_data
```

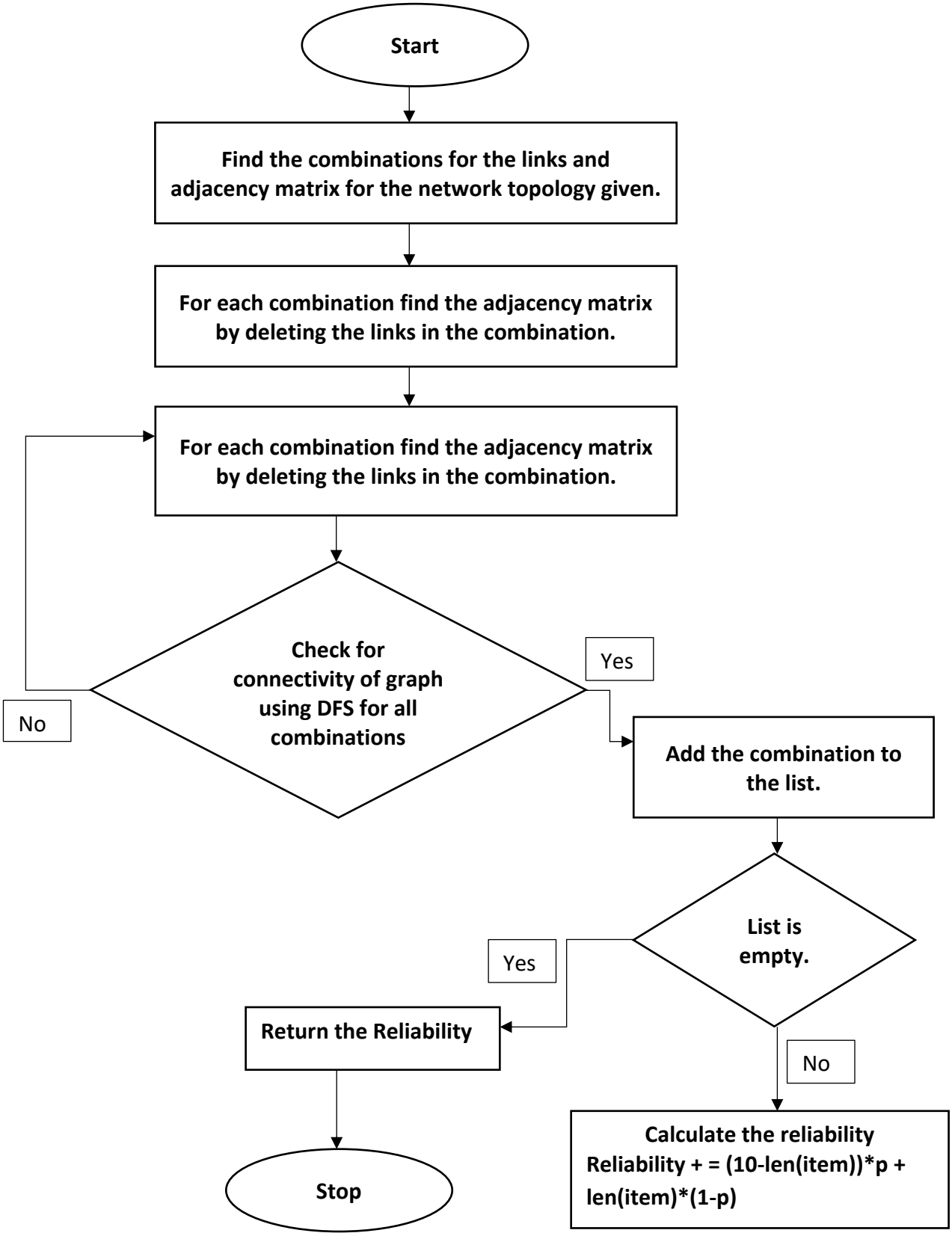
```
    // Loop through different sizes of combinations, from 1 to the length of Links_data
    for r from 1 to numLinks:
        // Call the generateCombinations method to generate combinations of size 'r'
        // Start with an empty currentCombination list and index 0
        generateCombinations(Links_data, r, 0, new ArrayList<>(), allCombinations)
    // Return the list of all combinations
    return allCombinations


fuction generateCombinations(Links_data, r, start, currentCombination, allCombinations):
    // If r is 0, add the currentCombination to allCombinations and return
    if r == 0:
        allCombinations.add(new ArrayList<>(currentCombination))
        return
    // Get the total number of Links_data
    numLinks = length of Links_data
    // Iterate through the Links_data from the 'start' index up to numLinks - r
    for i from start to numLinks - r:
        // Add the current Links_data[i] to the currentCombination
        currentCombination.add(Links_data[i])
        // Recursive call to generateCombinations with reduced r, updated start, and updated
currentCombination
        generateCombinations(Links_data, r - 1, i + 1, currentCombination, allCombinations)
        // Backtrack by removing the last element from currentCombination to explore other
combinations
        currentCombination.remove(last element of currentCombination)

function getAdjacencyMatrix(temp, link):
    // Based on the value of 'link', update the elements in the 'temp' adjacency matrix
accordingly
    switch link:
        case "a":
            set temp[0][1] to 0
            set temp[1][0] to 0
            break
        case "b":
            set temp[0][2] to 0
            set temp[2][0] to 0
            break
        case "c":
            set temp[0][5] to 0
            set temp[5][0] to 0
            break
        case "d":
            set temp[1][3] to 0
            set temp[3][1] to 0
```

```
            break
        case "e":
            set temp[3][4] to 0
            set temp[4][3] to 0
            break
        case "f":
            set temp[1][2] to 0
            set temp[2][1] to 0
            break
        case "g":
            set temp[2][5] to 0
            set temp[5][2] to 0
            break
        case "h":
            set temp[5][6] to 0
            set temp[6][5] to 0
            break
        case "i":
            set temp[4][6] to 0
            set temp[6][4] to 0
            break
        case "j":
            set temp[2][4] to 0
            set temp[4][2] to 0
            break
    // Return the updated 'temp' adjacency matrix
    return temp


function isGraphConnected(adjacencyMatrix):
    // Get the total number of nodes in the graph
    numNodes = length of adjacencyMatrix
    // Create a boolean array to track visited nodes, initialized to all false
    visited = new boolean[numNodes] initialized to false
    // Start the Depth-First Search (DFS) from the first node (node 0)
    dfs(adjacencyMatrix, 0, visited)
    // Check if all nodes were visited during the DFS traversal
    for each node in visited:
        if node is not visited:
            return false
    // If all nodes were visited, the graph is connected
    return true

function dfs(adjacencyMatrix, node, visited):
    // Mark the current node as visited
    set visited[node] to true
```

```
    // Loop through all nodes in the adjacency matrix
    for i from 0 to length of adjacencyMatrix - 1:
        // Check if there is an edge between the current node and node i, and if node i is not
visited
        if adjacencyMatrix[node][i] is 1 AND visited[i] is false:
            // Recursively call dfs with node i as the new starting node
            dfs(adjacencyMatrix, i, visited)

function network_reliability(p, adj_matrix, combinations):
    // Initialize the overall reliability with all links being active (reliability when all links are up)
    reliability = p raised to the power of 10
    probability = p
    // Create a copy of the adjacency matrix and a temporary matrix
    matrix = a 7x7 array filled with 0s
    temp = a 7x7 array filled with 0s
    for x from 0 to 6:
        for y from 0 to 6:
            temp[x][y] = adj_matrix[x][y]
            matrix[x][y] = adj_matrix[x][y]
    // Create lists to store information about up combinations
    up_size = an empty list to store the number of links that are up
    up_combinations = an empty list to store combinations where the graph is connected
    // Iterate through all combinations
    for i from 0 to size of combinations - 1:
        // Copy the adjacency matrix to the temporary matrix
        for x from 0 to 6:
            for y from 0 to 6:
                temp[x][y] = matrix[x][y]
        // Get the current combination
        Curr_data = combinations[i]
        // Disable links in the current combination and update the temporary matrix
        for j from 0 to size of Curr_data - 1:
            item = Curr_data[j]
            temp = getAdjacencyMatrix(temp, item)
        // Check if the graph is connected using the temporary matrix
        if isGraphConnected(temp) is true:
            // The graph is connected, so store the combination and the number of disabled links
            up_combinations.add(Curr_data)
            up_size.add(size of Curr_data)
    // Calculate the reliability for disconnected links and update the overall reliability
    for i from 0 to size of up_size - 1:
        size_data = up_size[i]
        reliability += probability raised to the power of (10 - size_data) multiplied by (1 - p) raised
to the power of size_data
    // Return the overall reliability of the network
    return reliability
```

```
function main(args):
    // Define the adjacency matrix representing the network
    adj_matrix = a 2D array with the given values
    // Set the initial value of p to 0.05
    p = 0.05
    // Create an array of strings representing links between nodes
    Links_data = an array containing the link names "a", "b", "c", ..., "j"
    // Generate all combinations of links from Links_data
    combinations = getCombinations(Links_data)
    // Create an empty list to store reliabilities for different values of p
    Reliabilities = an empty list
    // Loop from p=0.05 to p=1.0 with an increment of 0.05
    while p < 1.05:
        // Calculate the network reliability for the current value of p
        reliability_scale = network_reliability(p, adj_matrix, combinations)
        // Print the result with the current value of p
        print "The Reliability is " + reliability_scale + " for the value of p " + p
        // Add the reliability to the list
        Reliabilities.add(reliability_scale)
        // Increment p by 0.05
        p = p + 0.05
```

# Flowchart


Start
Find the combinations for the links and adjacency matrix for the network topology given.
For each combination find the adjacency matrix by deleting the links in the combination.
For each combination find the adjacency matrix by deleting the links in the combination.
Check for connectivity of graph using DFS for all combinations
Yes
No
Add the combination to the list.
List is empty.
Yes
No
Return the Reliability
Calculate the reliability
Reliability + = (10-len(item))*p + len(item)*(1-p)
Stop

# Result Analysis

The network reliability is calculated for different values of p with an interval of 0.05, and the result shows that with increase in the value of “p”, the network reliability also increases. So, we can say that p and network reliability are proportional to each other.

For lower values of p, let's say p = 0.05, the reliability of the network is extremely low. On the other hand, for higher values of p, let's say p = 1, the reliability of the network is significantly higher, reaching a level of 100% reliability.

```
onMessages -cp /Users/majetisanjana/Downloads/Reliability.java/bin App
The Reliability is 1.274200585937502E-6 for the value of p 0.05
The Reliability is 6.961240000000008E-5 for the value of p 0.10
The Reliability is 6.728452154296899E-4 for the value of p 0.15
The Reliability is 0.00318760959999999 for the value of p 0.20
The Reliability is 0.010183334350585938 for the value of p 0.25
The Reliability is 0.025280553599999988 for the value of p 0.30
The Reliability is 0.05259018298105454 for the value of p 0.35
The Reliability is 0.09587261439999974 for the value of p 0.40
The Reliability is 0.15762072973183672 for the value of p 0.45
The Reliability is 0.23828124999999997 for the value of p 0.50
The Reliability is 0.33579627347793106 for the value of p 0.55
The Reliability is 0.44558346239999974 for the value of p 0.60
The Reliability is 0.5609911623443362 for the value of p 0.65
The Reliability is 0.6741758295999962 for the value of p 0.70
The Reliability is 0.777265548706055 for the value of p 0.75
The Reliability is 0.8636071936000037 for the value of p 0.80
The Reliability is 0.9288579701880857 for the value of p 0.85
The Reliability is 0.9716867244000013 for the value of p 0.90
The Reliability is 0.9939085566654368 for the value of p 0.95
The Reliability is 1.0 for the value of p 1.00
majetisanjana@Majetis-Air Reliability.java %
```

**Fig-2: Output of the algorithm for different values of p**

The plot demonstrates a positive correlation between probability and reliability. As the probability of successful functioning increases, the system's reliability also increases. The system's reliability is lowest at very low probabilities, after which it remains high and approaches 100% as the probability approaches 1.0.

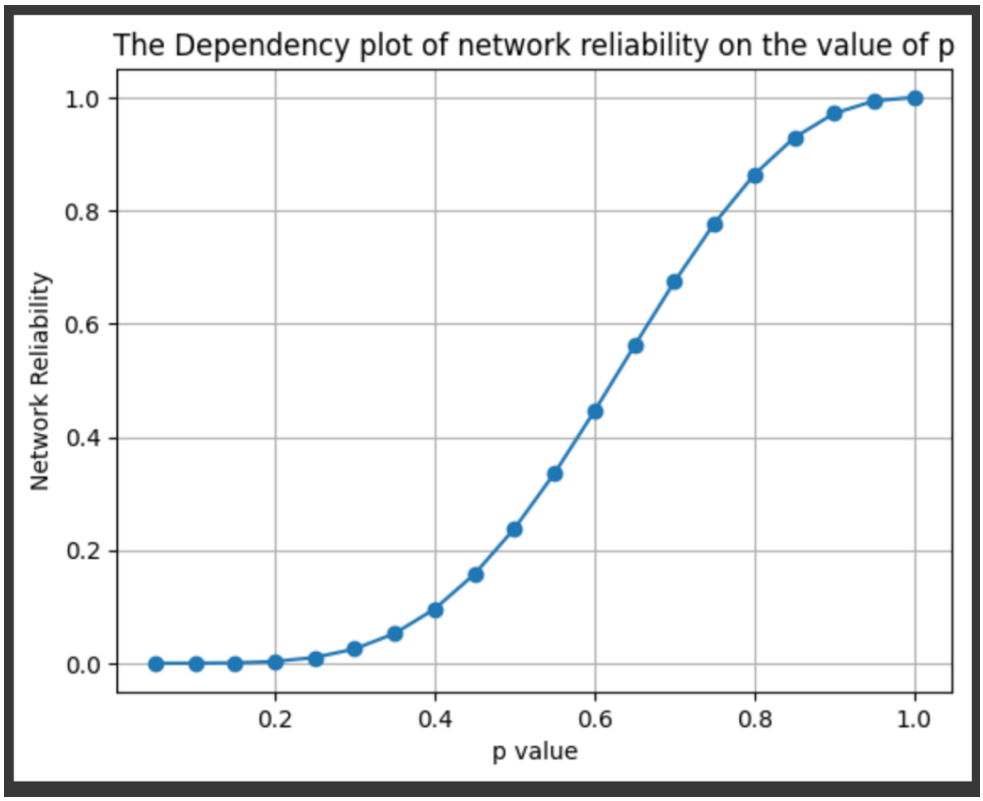



**Fig-3: The plot of p value and Network Reliability**

The use of print statements to display the reliability results for different values of "p". This is helpful for quickly observing the reliability values and checking if they match the expectations. Some other print statements are used to display intermediate results or debug specific parts of the program. The code is divided into multiple functions, each with a specific purpose. This modularity makes it easier to locate potential issues in specific functions and allows to focus on debugging one part at a time.

The getCombinations function generates all combinations of links, which can be reused in other parts of your code or other projects. This reduces the likelihood of writing redundant code and helps maintain consistency in the application. The program allows to experiment with different values of "p" and observe how the network reliability changes. By adjusting the values of "p" or modifying the network topology, we can explore the impact of different scenarios on the network's reliability.

# Appendix

App.java -

```
import java.util.ArrayList;
import java.util.List;
import java.lang.Math;

public class App {
   //To get combinations
   public static List<List<String>> getCombinations(String[] Links_data) {
      List<List<String>> allCombinations = new ArrayList<>();
      int numLinks = Links_data.length;
      for (int r = 1; r <= numLinks; r++) {
         generateCombinations(Links_data, r, 0, new ArrayList<>(), allCombinations);
      }

      return allCombinations;
   }
   //To generate combinations
   public static void generateCombinations(String[] Links_data, int r, int start, List<String>
currentCombination, List<List<String>> allCombinations) {
      if (r == 0) {
         allCombinations.add(new ArrayList<>(currentCombination));
         return;
      }

      int numLinks = Links_data.length;
      for (int i = start; i <= numLinks - r; i++) {
         currentCombination.add(Links_data[i]);
         generateCombinations(Links_data, r - 1, i + 1, currentCombination, allCombinations);
         currentCombination.remove(currentCombination.size() - 1);
      }
   }
   //To get adjacency matrix values deleted
   public static int[][] getAdjacencyMatrix(int[][] temp, String link){
      switch (link) {
         case "a":
            temp[0][1]=0;
            temp[1][0]=0;
            break;
         case "b":
            temp[0][2]=0;
            temp[2][0]=0;
```

```
            break;
        case "c":
            temp[0][5]=0;
            temp[5][0]=0;
            break;
        case "d":
            temp[1][3]=0;
            temp[3][1]=0;
            break;
        case "e":
            temp[3][4]=0;
            temp[4][3]=0;
            break;
        case "f":
            temp[1][2]=0;
            temp[2][1]=0;
            break;
        case "g":
            temp[2][5]=0;
            temp[5][2]=0;
            break;
        case "h":
            temp[5][6]=0;
            temp[6][5]=0;
            break;
        case "i":
            temp[4][6]=0;
            temp[6][4]=0;
            break;
        case "j":
            temp[2][4]=0;
            temp[4][2]=0;
            break;
    }
    return temp;
}
public static boolean isGraphConnected(int[][] adjacencyMatrix) {
    int numNodes = adjacencyMatrix.length;
    boolean[] visited = new boolean[numNodes];

    // Start the DFS from the first node (node 0)
    dfs(adjacencyMatrix, 0, visited);

    // Check if all nodes were visited
    for (boolean isVisited : visited) {
        if (!isVisited) {
```

```
            return false;
        }
      }

      return true;
   }

   public static void dfs(int[][] adjacencyMatrix, int node, boolean[] visited) {
      visited[node] = true;
      for (int i = 0; i < adjacencyMatrix.length; i++) {
         if (adjacencyMatrix[node][i] == 1 && !visited[i]) {
            dfs(adjacencyMatrix, i, visited);
         }
      }
   }
   //To check connectivity
   public static double network_reliability( double p, int[][] adj_matrix, List<List<String>>
combinations){
      //Intializing with all values active
      double reliability = Math.pow(p,10);
      double probability=p;
      int[][] matrix = new int[7][7];
      int[][] temp = new int[7][7];
      for(int x=0; x< adj_matrix.length; x++){
         for(int y=0;y<adj_matrix.length;y++){
            temp[x][y]=adj_matrix[x][y];
            matrix[x][y]=adj_matrix[x][y];
         }
      }
      List<Integer> up_size = new ArrayList<>();
      //Links that are up
      List<List<String>> up_combinations = new ArrayList<>();
      for( int i=0; i<combinations.size();i++){
         for(int x=0; x< matrix.length; x++){
            for(int y=0;y<matrix.length;y++){
               temp[x][y]=matrix[x][y];
            }
         }
         List<String> Curr_data = combinations.get(i);


         for (int j = 0; j < Curr_data.size(); j++){
            String item = Curr_data.get(j);
            temp=getAdjacencyMatrix(temp, item);
         }
         //check for connectivity
```

```
        if(isGraphConnected(temp)== true){
            up_combinations.add(combinations.get(i));
            up_size.add(combinations.get(i).size());
        }

    }
    //up_size has number of links that are disconnected that is probability will be (1-p)
    for(int i=0;i<up_size.size();i++){
        int size_data =up_size.get(i);
        reliability += (Math.pow((probability),(10- size_data))*Math.pow((1-p),(size_data)));

    }

    //check reliability
    return reliability;

}

public static void main(String[] args) throws Exception {
    int[][] adj_matrix = {{0,1,1,0,0,1,0},{1,0,1,1,0,0,0},{1,1,0,0,1,1,0},{0,1,0,0,1,0,0}
    ,{0,0,1,1,0,0,1},{1,0,1,0,0,0,1},{0,0,0,0,1,1,0}};
    double p = 0.05;
    //Indicating each link as a alphabet
    //a--> Link between 1 and 2
    //b--> Link between 1 and 3
    //c--> Link between 1 and 6
    //d--> Link between 2 and 4
    //e--> Link between 4 and 5
    //f--> Link between 2 and 3
    //g--> Link between 3 and 6
    //h--> Link between 6 and 7
    //i--> Link between 5 and 7
    //j--> Link between 3 and 5
    String[] Links_data = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j"};

    List<List<String>> combinations = getCombinations(Links_data);
    List<Double> Reliabilities = new ArrayList<>();

    while(p<1.05){
        double reliability_scale = network_reliability(p,adj_matrix,combinations);
        System.out.printf("The Reliability is %f for the value of p %.2f%n", reliability_scale, p);
        Reliabilities.add(reliability_scale);
        p=p+0.05;
    }
```

```
   }
}
```

To plot the graph:

Using python matplotlib library to plot the graph.

Graph.py

```
import matplotlib.pyplot as plt

Reliabilities=[1.274200585937502E-6 ,6.961240000000008E-5 ,6.728452154296899E-4
,0.00318760959999999 ,0.010183334350585938 ,
        0.025280553599999988 ,0.05259018298105454 ,0.09587261439999974
,0.15762072973183672 ,0.23828124999999997,
        0.33579627347793106 ,0.44558346239999974 ,0.5609911623443362
,0.6741758295999962 ,0.777265548706055 ,0.8636071936000037,
        0.9288579701880857 ,0.9716867244000013 ,0.9939085566654368 ,1.0]

Probabilities = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65, 0.70,
0.75, 0.80, 0.85, 0.90, 0.95, 1.00]

plt.plot(Probabilities, Reliabilities,marker='o')
plt.xlabel('p value')
plt.ylabel('Network Reliability')
plt.title('The Dependency plot of network reliability on the value of p')
plt.grid(True)
```

# README

1. Install latest version of JDK and an IDE like visual studio code.
2. Install latest version of Python3 and install the package matplotlib.
3. Run App.java and check the results.
4. Run Graph.py to obtain the visualizations of the results from App.java .

# References

https://www.geeksforgeeks.org/implementation-of-dfs-using-adjacency-matrix/

https://www.programiz.com/dsa/graph-adjacency-matrix