Project 2: Network reliability dependency on the individual link reliabilities

Algorithm Aspects of

Telecommunication Networks

CS 6385

Submitted by:

Niveditha Sasalapura Puttaiah (Net ID: nxs210042)

Table of Contents:

Introduction:

This project is to determine how the network reliability changes based on individual link reliabilities. This will be checked in different situations like changing link probability, flipping the link states and more. The system is considered operational, if the network is connected.

Network reliability is the service that notifies the user if delivery fails, while an unreliable one does not notify the user if delivery fails. It specifies the guarantees that the protocol provides with respect to the delivery of messages to the intended recipient(s). TCP(Transmission Control Protocol) is one of the protocol which provides network reliability.

Problem Description:

Problem 1: An algorithm needs to be constructed to compute the network reliability of a complete graph with 5 nodes and 10 edges. The links can fail anytime and every time the reliability of the system needs to be checked. This can be done by assigning reliability to individual links using the below formula.

$$p_i = p^{\lceil d_i/3 \rceil},$$

Where di is our student id. Since this is a complete graph, we will be having 1024 collection of link states, where one or more links could be down in each system. For different values of the probability 'p' in the interval [0.05,1] in steps of 0.05, the reliability needs to be calculated and checked.

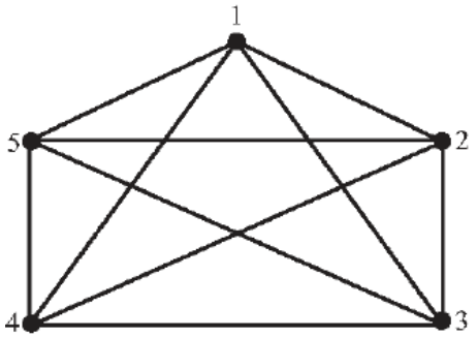
Solution: Using exhaustive enumeration, each state will be assigned with 'up' and 'down'. Then the reliability can be calculated by summing the probability of all the 'up' states. Graph will be plotted to determine how the network reliability of the system varies for varying values of 'p'.

Problem 2: In the second part, the probability will be fixed to 0.9. Picking k number of links randomly where k=0,1,…,20, and flipping the states of these links and again checking the reliability of the updated system. This is to determine how the reliability of the system changes due to such errors.

Solution: In this case, the reliability will be calculated for 0.9 probability. Errors like changing the link states from the original one and again calculating the reliability to check how it gets affected by such alterations.

Algorithm:

Step 1: Generating a complete graph with 5 nodes and 10 edges. Indexing the links from 1,…,10.




```
g = Graph()
g = Graph(directed=False)
g.add_vertex(5)
for a in range(0, 5):
    for b in range(a+1, 5):
        g.add_edge(a, b)
```

Step 2: Calculating all the possible 2^10 system states i.e., 1024 in binary and saving it in a master list. In the code, it is ‘Currentstate’.

```
def NetworkProbility():

    for i in range(0, 1024):
        CurrentState.append(format(i, "010b"))
```

Step 3: Exhaustive Enumeration is used here to list all the possible states of the system as UP/DOWN**.** For each system state, BFS checks if it is connected or not connected by traversing from source to target and storing all the visited nodes in a list. If the node list contains all the 5 nodes, then that means the system is in connected state, otherwise it is not. Once this is done, list out all the possible states of the system as UP/DOWN. The same is shown below for directed graph which generates 32 states.

Breadth First Search: This algorithm is used to traverse the tree structure or the network level by level to cover all the vertices that are connected to the source node. This will have a list to separate the states already visited from those yet to be visited.

```
    flag = False
    visited = [False for i in range(5)]
    Visitednode = []
    countVisited = 0
    for src in range(len(Adjmatrix)):
        for idx in range(5):
            #checking if edge present between 2 nodes
            if (not(visited[idx]) and Adjmatrix[src][idx] == 1 and (idx not in Visitednode)):
                Visitednode.append(idx)
                countVisited += 1
                visited[idx] = True
    # Verification and returning the decision
    if (countVisited == 5):
        flag = True

    return flag
```

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. $A\overline{B}\overline{C}DE$	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. $ABC\overline{D}\overline{E}$	Down
3	17. $A\overline{B}\overline{C}\overline{D}E$	Down
	18. $A\overline{B}C\overline{D}\overline{E}$	Down
	19. $A\overline{B}\overline{C}D\overline{E}$	Up
	20. $\overline{A}B\overline{C}\overline{D}E$	Down
	21. $\overline{A}BC\overline{D}\overline{E}$	Down
	22. $\overline{A}B\overline{C}D\overline{E}$	Down
	23. $\overline{A}\overline{B}C\overline{D}E$	Up
	24. $\overline{A}\overline{B}CD\overline{E}$	Down
	25. $\overline{A}BC\overline{D}\overline{E}$	Down
	26. $\overline{A}B\overline{C}D\overline{E}$	Down
4	27. $A\overline{B}\overline{C}\overline{D}\overline{E}$	Down
	28. $\overline{A}B\overline{C}\overline{D}\overline{E}$	Down
	29. $\overline{A}\overline{B}C\overline{D}\overline{E}$	Down
	30. $\overline{A}\overline{B}D\overline{D}\overline{E}$	Down
	31. $\overline{A}\overline{B}\overline{C}D\overline{E}$	Down
5	32. $\overline{A}\overline{B}\overline{C}\overline{D}\overline{E}$	Down

Step 4: Then the reliability will be calculated individually for all the 'UP' systems using the below formula.

$$p_i = p^{\lceil d_i/3 \rceil}$$

di will take the values from the student id for each link state and 'p' will vary from 0.05 to 1 in the intervals of 0.05 i.e., after calculating the reliability of all the 'UP' systems with 'p' value 0.05, then again the reliability will be calculated for the same systems with 0.1 probability and so on.

```
Student_id = "2021474685"
temp = []

for j in range(0, 10):

    if itr_list[j] == '1':

        temp.append(pow(baseProbability, ((int(Student_id[j]) / 3.0))))
    else:
        temp.append(1-(pow(baseProbability, ((int(Student_id[j]) / 3.0)))))


return math.prod(temp)
```

Step 5: Now we have the reliability of all the 'UP' systems for different probability values. All these reliabilities will be summed up to determine the entire network reliability. Plot the graph showing variation in reliability value for different probability values 'p'.

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

```
master_LinkProabilityvalue.append(sum(batch_LinkProabilityvalue))
```

Extended algorithm: Flipping the states.

Step 6: Selecting ‘k’ number of states randomly out of 1024 where k=0,1,2,..,20. In each iteration, the k system states are flipped(1 will be 0 and vice versa) and again checking if it is connected or not i.e., UP/DOWN.

```
for a in range(0,4):       #Calculating the reliability multiple times

    Averagevalue=0
    var=random.sample(backup,k)

    for j in range(0,len(var)):

        str1 =''
        for p in range(0,10):

            if var[j][p]=='1':
                str1+='0'

            else:
                str1+='1'

        x=backup.index(var[j])
        var[j] = str1
        backup[x]=str1
```

Step 7: Fixing the probability to 0.9. For each ‘k’ value, random k number of states are selected multiple times from the network and reliability will be calculated for these different states. There average is getting computed to determine the reliability for each ‘k’ value.

```
Averagevalue=np.mean(master_LinkProabilityvalue)
```

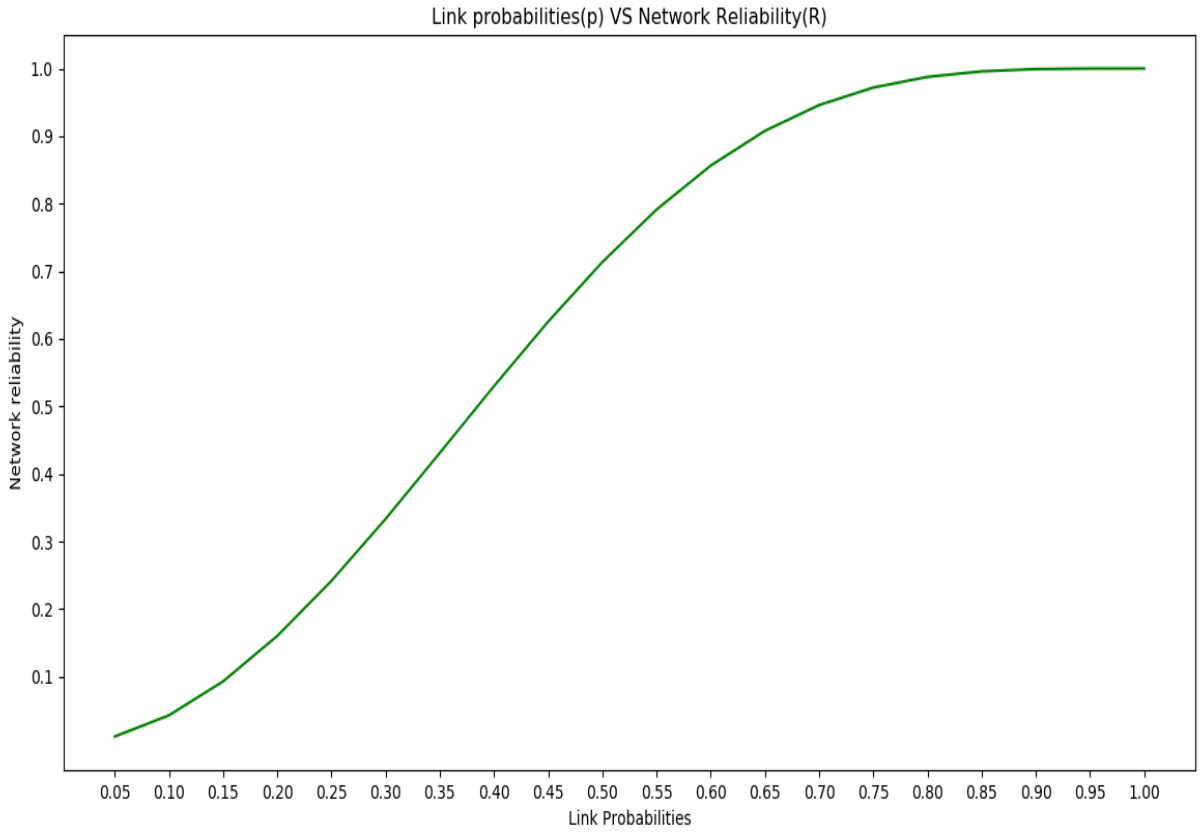
Step 8: Finally, these averaged reliabilities are summed up to determine the network reliability. Plotted the graph showing variation in reliability value for different values of ‘k’.

Outputs and Graphs:

Output when the probability value changes,

```
The Network Reliability with probability 0.05 is 0.011515381488299288
The Network Reliability with probability 0.1 is 0.042867883387975154
The Network Reliability with probability 0.15 is 0.09310908522609301
The Network Reliability with probability 0.2 is 0.16043280007942595
The Network Reliability with probability 0.25 is 0.2419008375251075
The Network Reliability with probability 0.3 is 0.33366106516700117
The Network Reliability with probability 0.35 is 0.4312691352402698
The Network Reliability with probability 0.4 is 0.5300581247157665
The Network Reliability with probability 0.45 is 0.6255198503500446
The Network Reliability with probability 0.5 is 0.7136637219223112
The Network Reliability with probability 0.55 is 0.7913207027744174
The Network Reliability with probability 0.6 is 0.8563640468848699
The Network Reliability with probability 0.65 is 0.9078254140386122
The Network Reliability with probability 0.7 is 0.9458943665082123
The Network Reliability with probability 0.75 is 0.9718002572998246
The Network Reliability with probability 0.8 is 0.9875868597191261
The Network Reliability with probability 0.85 is 0.9958001552622167
The Network Reliability with probability 0.9 is 0.9991166445765571
The Network Reliability with probability 0.95 is 0.9999414088275366
The Network Reliability with probability 1.0 is 1.0
```

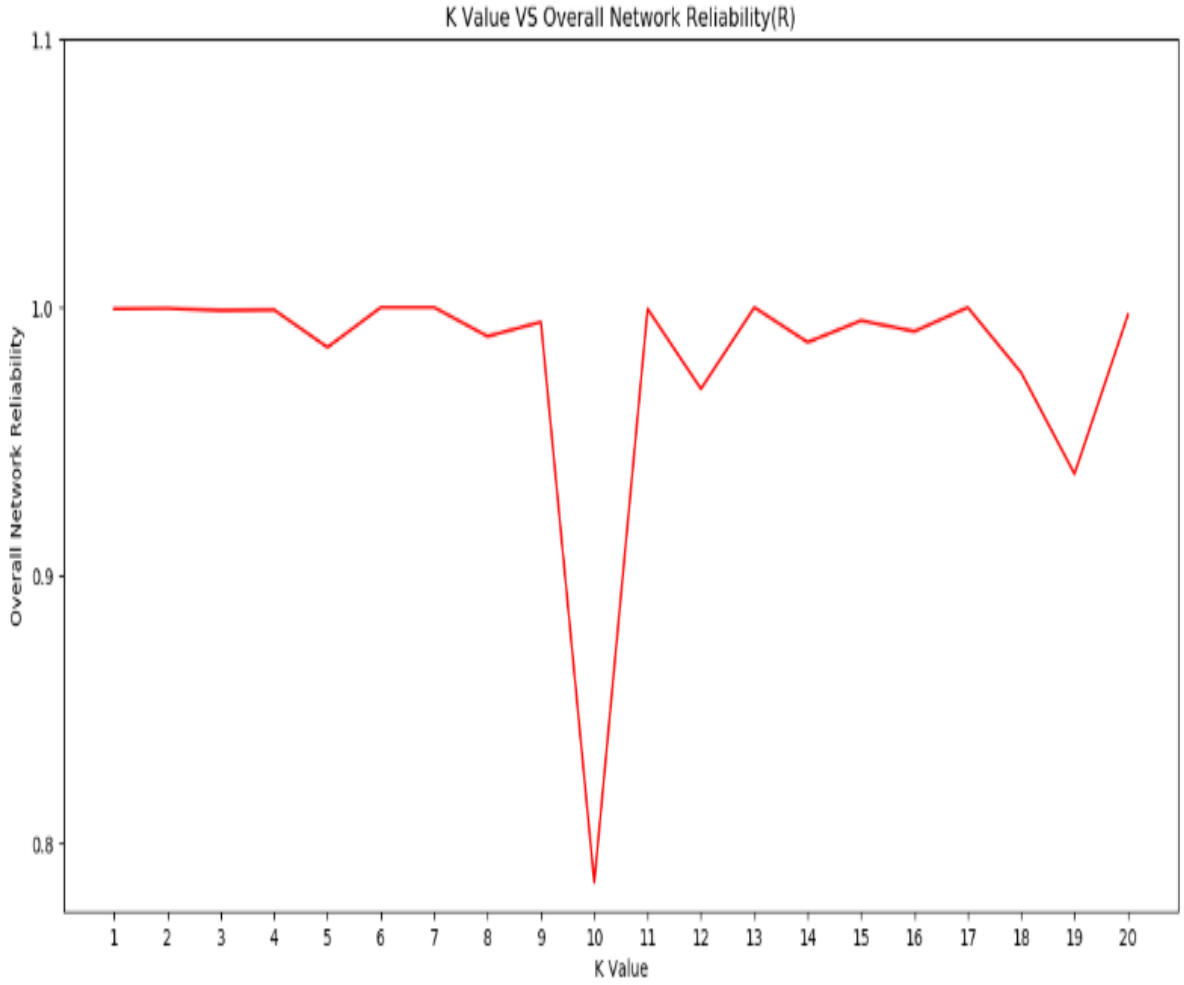
As it is shown in the above output, it can be said that the network reliability increases as the probability of the system increases. That is, network becomes more resistant to losses. The same is shown graphically below.


Link probabilities(p) VS Network Reliability(R)
Network reliability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
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
Link Probabilities

Output when k states get flipped with fixed probability of 0.9,

```
The Network reliability when k=1 is 0.9995249018082571
The Network reliability when k=2 is 0.9996857302096361
The Network reliability when k=3 is 0.9988694568698803
The Network reliability when k=4 is 0.9990802122949993
The Network reliability when k=5 is 0.9851374506340186
The Network reliability when k=6 is 1.0
The Network reliability when k=7 is 1.0
The Network reliability when k=8 is 0.9892380136551573
The Network reliability when k=9 is 0.9945652576681847
The Network reliability when k=10 is 0.7855328262039414
The Network reliability when k=11 is 0.9994462683525223
The Network reliability when k=12 is 0.9696182883068358
The Network reliability when k=13 is 1.0
The Network reliability when k=14 is 0.9870209795454077
The Network reliability when k=15 is 0.995096935069019
The Network reliability when k=16 is 0.9910844158637473
The Network reliability when k=17 is 1.0
The Network reliability when k=18 is 0.9757077930102702
The Network reliability when k=19 is 0.937922242714277
The Network reliability when k=20 is 0.9971115462149323
```

As shown in above output, different network reliabilities are shown for different values of ‘k’. Since random links will be selected out of 1024 and flipped, there are chances that more ‘DOWN’ states might result due to the flipping. The same is shown graphically below.


K Value VS Overall Network Reliability(R)
Overall Network Reliability
1.1
1.0
0.9
0.8
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
K Value

References:

1. MATPLOTLIB Library
   https://matplotlib.org/devdocs/gallery/subplots_axes_and_figures/subplots_demo.html
2. Lecture Notes by Professor Andras Farago
3. Wikipedia
   https://en.wikipedia.org/wiki/Reliability_(computer_networking)

Appendix:

```
import networkx as netx
from graph_tools import *
import random
import math
import numpy as np
import copy
import matplotlib.pyplot as pyt

#Declaring global variables
Iterate = []
CurrentState = []
Newstate = []
baseProbability = 0.05
master_LinkProabilityvalue = []

# Generating graph with 5 nodes and 10 edges
g = Graph()
g = Graph(directed=False)
g.add_vertex(5)
for a in range(0, 5):
   for b in range(a+1, 5):
      g.add_edge(a, b)


def createTheGraph(linkProbabilities,reliabilities):
          fig = pyt.figure()
```

```
        ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
        pyt.rcParams["figure.figsize"] = [7.00, 3.50]
        pyt.rcParams["figure.autolayout"] = True
        ax.set_title("Link probabilities(p) VS Network Reliability(R)")
        ax.set_xlabel("Link Probabilities")
        ax.set_ylabel("Network reliability")
        tick1 = [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8,
0.85, 0.9, 0.95, 1]
        tick2 = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
        ax.set_xticks(tick1)
         ax.set_yticks(tick2)
        ax.plot(linkProbabilities, reliabilities, color="green")
         pyt.show()



def createTheGraph1(kvalue,reliabilities):
        fig = pyt.figure()
        ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
        pyt.rcParams["figure.figsize"] = [7.00, 3.50]
        pyt.rcParams["figure.autolayout"] = False
        pyt.plot(kvalue, reliabilities,color="red")


        pyt.title("K Value VS Overall Network Reliability(R)")
        pyt.xlabel("K Value")
        pyt.ylabel("Overall Network Reliability")
        tick1 = kvalue
        pyt.xticks(tick1)
        pyt.yticks(np.arange(0.8, 1.2, 0.1))
```

```
        pyt.show()


# Calculating probability of individual states

def LinkProability1(Itr):
    itr_list = list(Itr)
    Student_id = []
    Student_id = "2021474685"
    temp = []

    for j in range(0, 10):
        if itr_list[j] == '1':
            temp.append(pow(baseProbability, ((int(Student_id[j]) / 3.0))))
        else:
            temp.append(1-(pow(baseProbability, ((int(Student_id[j]) / 3.0)))))


    res = 1
    for item in temp:
        res *= item
    return res


# Generating 1024 link states

def NetworkProbility():
```

```
    for i in range(0, 1024):
        CurrentState.append(format(i, "010b"))



# Checking if link states are up/down by node traversing


def updownstatecheck():
    global baseProbability
    // Assigning values to edges
    statelist = ['AB', 'AC', 'AD', 'AE', 'BC', 'BD', 'BE', 'CD', 'CE', 'DE']


    cnt = 0
    master_LinkProabilityvalue = []
    for j in range(0, 20):
        batch_LinkProabilityvalue = []


        for ptr in range(0, 1024):
            // Creating the adjacency matrix to store the connected edges
            Adjmatrix = [[0]*5 for _ in range(5)]
            Iterate = CurrentState[ptr]
            list2 = []
            list2 = [char for char in Iterate]
            new_list = list(zip(statelist, list2))
             output_list = [item for item in new_list if item[1] == '1']


            for itm1 in output_list:
                r = 0
```

```
c = 0
for idx in range(len(itm1[0])):
    if itm1[0][idx] == 'A':
        if idx == 0:
            r = 0
        else:
            c = 0
    if itm1[0][idx] == 'B':
        if idx == 0:
            r = 1
        else:
            c = 1
    if itm1[0][idx] == 'C':

        if idx == 0:
            r = 2
        else:
            c = 2
    if itm1[0][idx] == 'D':
        if idx == 0:
            r = 3
        else:
            c = 3
    if itm1[0][idx] == 'E':
        if idx == 0:
            r = 4
        else:
```

```
                    c = 4

                Adjmatrix[r][c] = 1
                Adjmatrix[c][r] = 1

            if BFS(Adjmatrix):
                batch_LinkProabilityvalue.append(LinkProability1(Iterate))

        master_LinkProabilityvalue.append(sum(batch_LinkProabilityvalue))

        print('The Network Reliability with probability {} is {}' .format(round(baseProbability, 3),
master_LinkProabilityvalue[j]))

        baseProbability += 0.05

    base_list = np.linspace(0.05,1.0,20)
    createTheGraph(base_list,master_LinkProabilityvalue)

def BFS(Adjmatrix):

    flag = False
    visited = [False for i in range(5)]
    Visitednode = []
    countVisited = 0
    for src in range(len(Adjmatrix)):
        for idx in range(5):
            if (not(visited[idx]) and Adjmatrix[src][idx] == 1 and (idx not in Visitednode)):
                Visitednode.append(idx)
```

```
            countVisited += 1
            visited[idx] = True


    # Verification and returning the decision
    if (countVisited == 5):    #Checking if the graph is connected or not
        flag = True

    return flag


def updownstatecheckagain(Newstate):
    global baseProbability
    statelist = ['AB', 'AC', 'AD', 'AE', 'BC', 'BD', 'BE', 'CD', 'CE', 'DE']
    cnt = 0
    batch_LinkProabilityvalue = []
    master_LinkProabilityvalue = []

    for ptr in range(0, 1024):
        Adjmatrix = [[0]*5 for _ in range(5)]
        Iterate = Newstate[ptr]
        list2 = []
        list2 = [char for char in Iterate]
        new_list = list(zip(statelist, list2))
        output_list = [item for item in new_list if item[1] == '1']

        for itm1 in output_list:
```

```
r = 0
c = 0
for idx in range(len(itm1[0])):
    if itm1[0][idx] == 'A':
        if idx == 0:
            r = 0
        else:
            c = 0
    if itm1[0][idx] == 'B':
        if idx == 0:
            r = 1
        else:
            c = 1
    if itm1[0][idx] == 'C':
        if idx == 0:
            r = 2
        else:
            c = 2
    if itm1[0][idx] == 'D':
        if idx == 0:
            r = 3
        else:
            c = 3
    if itm1[0][idx] == 'E':
        if idx == 0:
            r = 4
        else:
```

```
            c = 4

        Adjmatrix[r][c] = 1
        Adjmatrix[c][r] = 1

    if BFS(Adjmatrix):
        itr_list = list(Iterate)
        Student_id = []
        Student_id = "2021474685"
        temp = []
        baseProb = 0.9
        #p += 1

        for j in range(0, 10):
            if itr_list[j] == '1':
                temp.append(pow(baseProb, ((int(Student_id[j]) / 3.0))))
            else:
                temp.append(1-(pow(baseProb, ((int(Student_id[j]) / 3.0)))))

        res = 1
        for item in temp:
            res *= item

        batch_LinkProabilityvalue.append(res)

return sum(batch_LinkProabilityvalue)
```

```
# Function to flip the states based on k value and calculate the reliability

def part2_linkreliability():
    k=1
    avg_value_prob_list = []

    for i in range(0,20):
        var=[]
        backup = []
        backup=copy.deepcopy(CurrentState)
        master_LinkProabilityvalue = []
        for a in range(0,4):
            Averagevalue=0
            var=random.sample(backup,k)

            for j in range(0,len(var)):
                str1 =''
                for p in range(0,10):           # flipping the states
                    if var[j][p]=='1':
                        str1+='0'
                    else:
                        str1+='1'

                x=backup.index(var[j])
                var[j] = str1
```

```
            backup[x]=str1

        master_LinkProabilityvalue.append(updownstatecheckagain(backup))

    Averagevalue=np.mean(master_LinkProabilityvalue)    #averaging the reliability of multiple
k states
    if Averagevalue > 1.0:
        Averagevalue = 1.0
    avg_value_prob_list.append(Averagevalue)
    print("The Network reliability when k={} is {}".format(k,Averagevalue))
    k+=1

  kvalue=list(range(1, 21))
  createTheGraph1(kvalue,avg_value_prob_list)


if __name__ == "__main__":
  NetworkProbility()

  updownstatecheck()

  part2_linkreliability()
```