RELIABILITY CONFIGURATION

ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

TE 6385.001

V. PURNA CHANDRA VINAY KUMAR

PXV171630

Table of Contents

RELIABILITY CONFIGURATION OF THE NETWORK 3
PROBLEM STATEMENT 3
Tasks: 3
APPROACH 4
ALGORITHM 4
Task 1: 4
Task 2: 7
FLOW CHART 8
PLOTS 9
Reliability vs p: 9
Reliability vs K: 9
REFERENCES 11
APPENDIX 11
READ ME: 13

RELIABILITY CONFIGURATION OF THE NETWORK

The reliability of the network depends on the topology of the network and the probability of the failure of the hosts and the links. The system is said to be 'up' if all the nodes of the network is connected i.e. at least one of the links connecting all other nodes should be 'up'. The reliability(R) of the network is calculated by multiplying the 'up' and 'down' probabilities of the links for the 'up' condition of the system state, and adding up all the possible combinations. Since, the reliability defines the probability that the system is 'up' it is always less than one (R<=1).

The complete undirected graph network topology is constructed for the given number of nodes '***N***' eliminating the self-loops and parallel edges. The system condition is checked using the standard DFS or BFS algorithms and if connected the 'up' and 'down' probabilities are multiplied to calculate the reliability for the configuration of the network. Since for every link the possible states of the links are '**2**', 'up' or 'down', the number of possible system states is '$\mathbf{2^N}$'. All the possible '$\mathbf{2^N}$' states are checked for connectivity of the network and reliability is calculated if the system is 'up'. Summing the reliabilities of all the possible connected states of the network gives the reliability of the system.

PROBLEM STATEMENT

The aim of the project is to compute the network reliability for the complete undirected graph of '***N***' nodes eliminating parallel edges and self-loops, using the method of exhaustive enumeration. The project aims at fulfilling the following tasks.

Tasks:

- To design an algorithm to construct the complete undirected graph for '***N***' nodes and to check all the possible connected graphs from '$\mathbf{2^N}$' possible states and compute the network reliability for all the connected states.
- To write a computer program that implements the above algorithm.
- The program takes the number of nodes '***N***', the reliability of the link '***p***' as inputs.
- It computes the reliability of the system for different values of '***p***' plots the dependency of the network reliability on '***p***'.
- For the next experiment the reliability value of the link '***p***' is fixed to a value. Among all the '$\mathbf{2^N}$' possible combinations of component sates, '***k***' states are picked randomly and the system state is altered i.e. if the system is 'up' it is changed to 'down' and if it is 'down' changed to 'up' and the reliability of the network is then computed.

- The connectivity of the system is checked using the Depth First Search(DFS) non recursive algorithm and if the nodes are connected the system is said to be in 'up' state.
- The total reliability of the network vs 'k' is plotted for different values of '***k***'. The experiment is repeated several times for the value of '***k***' and then averaged to reduce the effect of randomness.

APPROACH

The approach in calculating the network reliability is described in the following steps:

- The program takes the number of nodes which is '***N***' and constructs the network topology as described above.
- The program then calculates the number of links for the given number of nodes and creates a list of link states consisting of 0's and 1's.
- For the first task, i.e. to find the dependency of the network reliability of the link reliability, the DFS algorithm is applied on the link state list and connectivity of the graph is verified.
- The reliability of the state if it is connected is calculated by multiplying all the link reliability probabilities, i.e. '***p***' for all the operational links and '***1-p***' for all the down links.
- The total reliability of the network is then calculated by adding all the connected network reliabilities and the graph is plotted.
- For the second task the link reliability is fixed to a value and '***k***' random numbers generated between the range '**$0\text{-}2^{N}$**' and the respective mapped link state condition is inverted. The total reliability is then calculated for all the 'up' states.
- The experiment is repeated several times and averaged and plotted the reliability vs '***k***'.

ALGORITHM

The programming language Python v.3.6.4 is used to calculate the reliability of the network and to plot the required plots.

Task 1:

<u>High Level Description</u>:

1. The network topology is constructed for '***N = 5***' nodes eliminating self-loops and parallel edges as described in the problem statement. Therefor the number of links can be calculated as '***(N-1)*N*0.5***' which is '***10***'.

2. The program creates a list of link states consisting of ***0's*** and ***1's***. The list can be mapped to the link condition upper triangular '***5 x*** 5' matrix of the nodes.

$$M_{i\times j} = \begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 & 0 & 0 \\ 4 & 1 & 0 & 1 & 0 & 0 \end{array}$$

3. The above matrix defines the links states between different nodes '***i x j***'. '***1***' denotes that the link between the ***i,j*** and is 'up' and '***0***' denotes the link is 'down'. Since the network is undirected and there are no parallel edges, if the link '***i x j'*** is 'up' then the link '***j x i'*** is also 'up' and if link ***i,j*** is down then link ***j,I*** is also 'down'.
4. Therefore, the matrix '$M_{i\times j}$***'*** is diagonally symmetric i.e. $M^T = M$. Hence, the upper triangular matrix of the network link state can be mapped to a list of link states as follows.

$$\bar{M}[i,j] = [(0,1),(0,2),\dots,(0,4),(1,2),\dots,(1,4),(2,3),\dots,(2,4),(4,3)]$$

5. The link $M(0,1)$ corresponds to the $\bar{M}[0]$, and $M(0,2)$ to $\bar{M}[1]$ soon, and $M(4,3)$ corresponds to $\bar{M}[9]$. Let $'t'$ denotes the index of link state list $'\bar{M}'$ which corresponds to the link $M(i,j)$, the $'i,j'$ can be mapped to $'t'$ as follows:

$$t = i \times N + j - \frac{(i+1)\times(i+2)}{2}$$

6. The mapping can be done for all '$i > j$'and $'N'$ denotes the number of nodes. Therefore, the above network link state matrix $M(i,j)$, mapped to list as $\bar{M}[t]$ gives,

$$\bar{M}[t] = [1,1,0,1,0,1,0,1,1,0]$$

7. The DFS algorithm is then implemented to the above graph and connectivity of the graph is verified. If the graph is connected, the reliability of the configuration is then calculated by multiplying $'p'$ as many times $1's$ are present in the list and $'1-p'$ as many times $1's$ are present in the list.
8. A loop runs over the range $0 - 2^N$ i.e. $0 - 1023$ and the decimal value is converted to corresponding binary value and is mapped to the list $\bar{M}[t]$. If the value of the loop variable

is $'4'$ which corresponds to binary value $'0000000100'$ is mapped to $\bar{M}[t] = [0,0,0,0,0,0,0,1,0,0]$ which defines the network's link state.

9. The step 7 is repeated to find the connectivity and to calculate the reliability of the configuration.
10. The total reliability is then calculated by summing up all the operational network configurations.
11. The whole process is repeated for different values of $'p'$ ranging from $[0-1]$ in steps of 0.04 and the graphs are plotted for reliability vs $'p'$.

Pseudo Code

```
def Reliability(link_state, p):

    up_prob = link_state*p
    invert all ones and zeros in link_state
    down_prob = link_state*(1-p)
    R = add up_prob and down_prob and multiply all the elements.
    return R

def DFS_algorithm(link_state, nodes):
    let Stack be a stack
    Stack.push(nodes[0])
    while(True):
        current_node = Stack.pop()
        for i in nodes:
            if i and current_node connected and not searched:
                label i as searched
                Stack.push(i)
        If no nodes connected to current_node:
            Label current_node as visited
        If stack is empty:
          Break
  If all nodes visited:
    Graph is connected

def Total_Reliability():
    for p in range of [0-1] in step 0.04:
        for i in all the configurations [0-1024]:
            convert to binary value and map to link_state list.
            Apply DFS_algorithm(link_state, nodes)
            If connected calculate Reliability(link_state,p)
Plot(p,Total_Reliability)
```

Task 2:

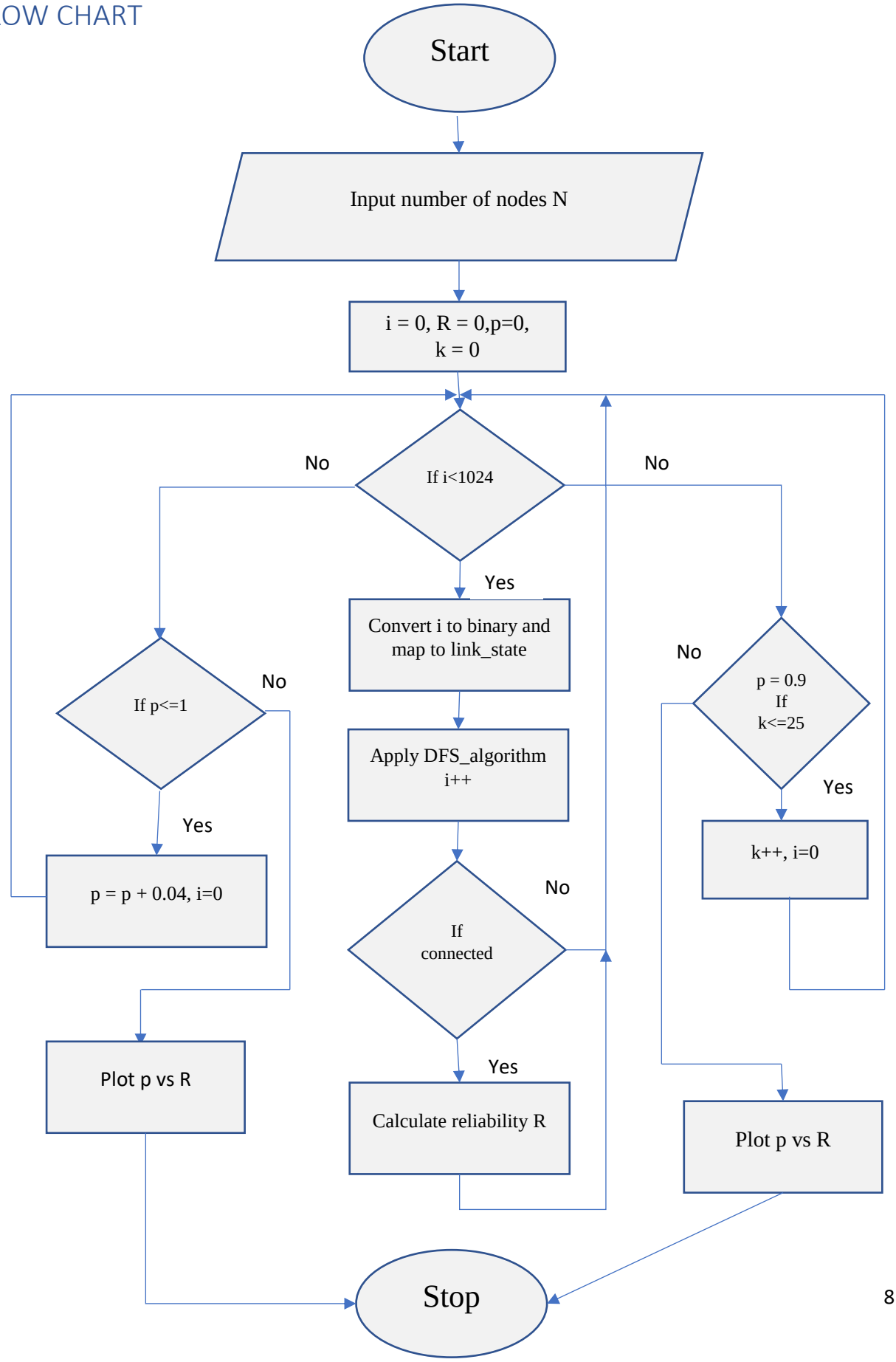
1. For this task the link reliability is fixed to $p = 0.9$ and k random combinations among the $2^{10} = 1024$ possible combinations are taken and the system state should be inverted and then the reliability is calculated.
2. To do this, the loop that runs over the range of k $[0 - 25]$, generates k random numbers. The generated decimals number are converted to binary equivalent and mapped to $\bar{M}[t]$ as described in task 1, step -6.
3. The system state for the combination is then inverted and total reliability is then calculated. The whole steps all through $1 - 11$ of task 1 is repeated to calculate the reliability.
4. The loop inside the k loop repeats the experiment several times and the total reliability is averaged for all the experiments.
5. The total reliability vs k is then plotted for all the values of k.

Pseudo Code

```
def K_var_fun():
    link_probability is fixed to p = 0.9
    for k in range of [0-25]:
        generate k random samples in range [0-1024]
        for i in all the configurations [0-1024]:
            convert to binary value and map to link_state list.
            Apply DFS_algorithm(link_state, nodes)
            If k == i invert the system condition
            If connected calculate Reliability(link_state,p)
   return reliability


plot(k,reliability)
```

FLOW CHART


Start
Input number of nodes N
i = 0, R = 0,p=0,
k = 0
If i<1024
No
No
Yes
If p<=1
No
Yes
p = p + 0.04, i=0
Plot p vs R
Convert i to binary and
map to link_state
Apply DFS_algorithm
i++
If
connected
No
Yes
Calculate reliability R
p = 0.9
If
k<=25
No
Yes
k++, i=0
Plot p vs R
Stop

PLOTS

Reliability vs p:

- 'p' defines the link probability i.e. the probability that the link is operational. As the value of p, the probability that link is operational increases.
- Therefore as the plot shows that as p increases the total reliability of the network increases and approaches 1 as p approaches to 1.

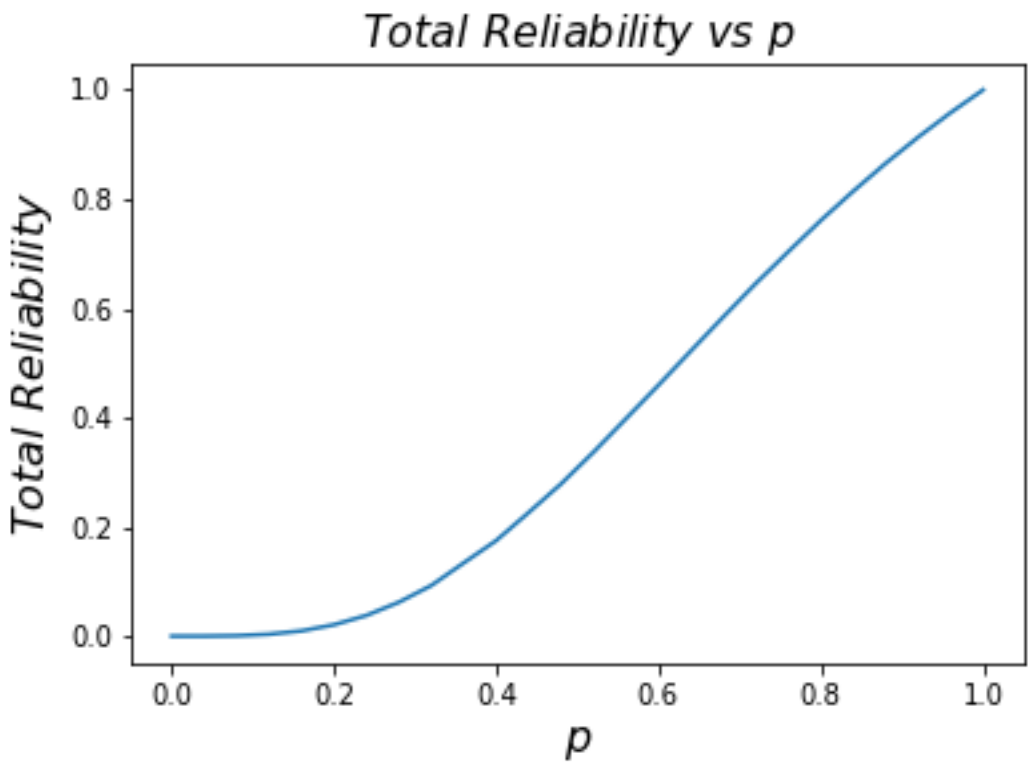
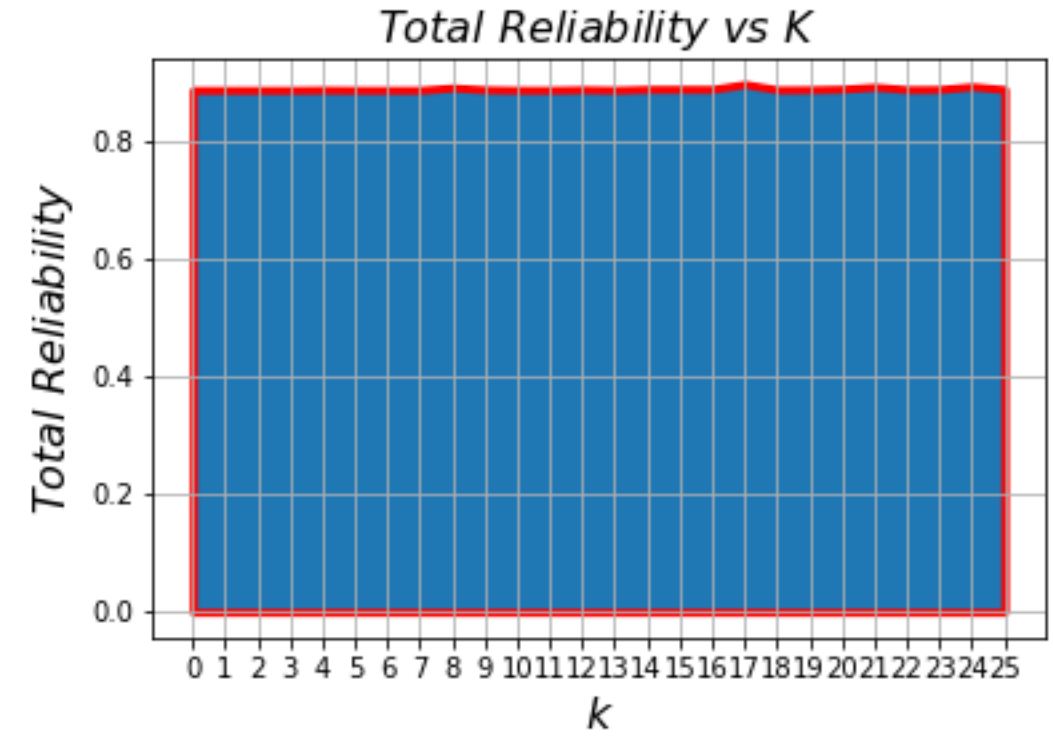


Fig: Plot reliability vs p

Reliability vs K:

- K is chosen randomly from the range of [0-1024) combinations of the network.
- Since K is random number of combinations that is used to alter the system states, K has a minimum random effect on the link reliability.



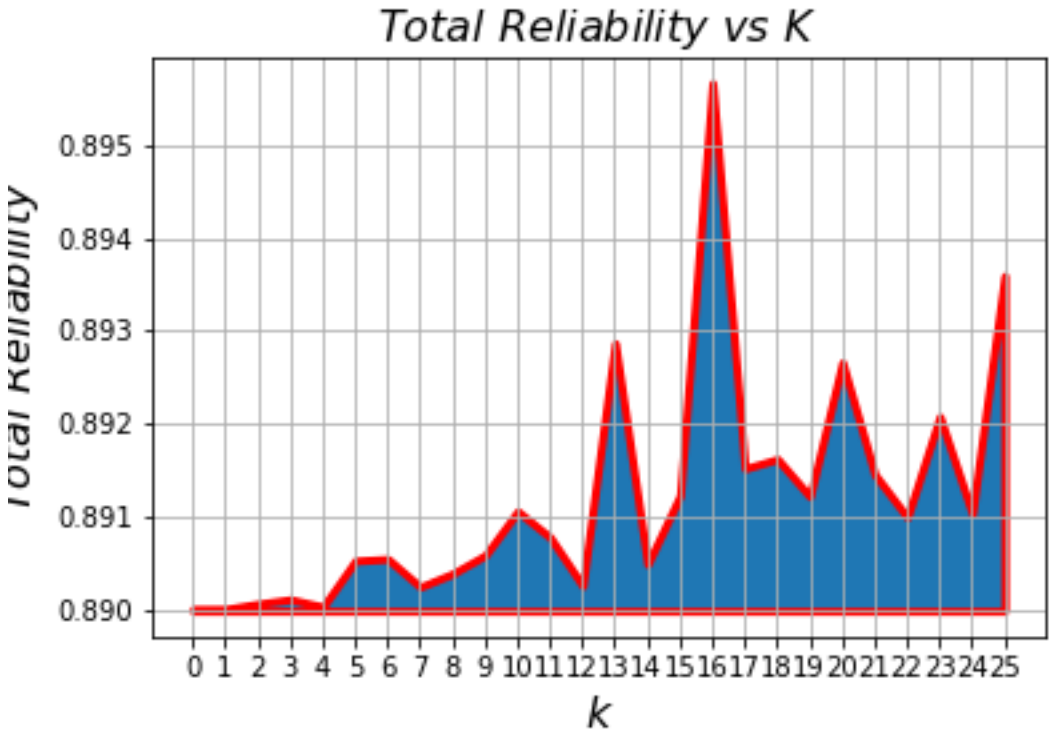


Fig: Reliability vs K

REFERENCES

1) https://en.wikipedia.org/wiki/Depth-first_search
2) https://en.wikipedia.org/wiki/Reliability_(computer_networking)

APPENDIX

```
#   --------- Network Reliability Problem ----------- #

import numpy as np
import matplotlib.pyplot as plt

#%matplotlib

# Create a list, which stores the Link condition 'up' or 'down'
# Initially set the all the links state to 'down'
# list contains series of '1' (defines 'up' state) and '0' (defines 'down'
state)
# Number of links n = 10

# Link[i+j] defines the link condition between the nodes i,j or j,i


def Reliability(link_state,p):
    link_up_prob = np.array(link_state)*p
    swap = {0:1,1:0}
    inverting = lambda i: swap[i]
    link_dw_prob = np.array(list(map(inverting,link_state)))*(1-p)
    link = link_up_prob+link_dw_prob
    R = 1
    for i in link:
        R*=i # Calculate the reliability of the configuration
    return R

def DFS_algorithm(link_state,nodes):
    # Implements the DFS non-recursive algorithm
    stack = []
    visited = []         # list that holds the visited nodes
    neighbour_nodes = [] # list that saves the neighbouring nodes
    stack.append(nodes[0])
    neighbour_nodes.append(nodes[0])
    while(True):
        linked = False
        current_node = stack[-1]
        for i in nodes:
            temp = int((current_node+1)*(current_node+2)/2)
            if(link_state[current_node*len(nodes)+i-(temp)]==1 and (i not
in neighbour_nodes) and (i>stack[-1])):
```

```
                stack.append(i)
                neighbour_nodes.append(i)
                linked = True

        if(linked==False):
            visited.append(stack.pop())

        if(not bool(stack)):
            break
    # if the length of visited list and nodes is equal then the graph is
connected
    if(len(visited)==len(nodes)):
        return True
    else:
        return False

def Total_reliability(K,p,k_gen):
    nodes = list(map(int,np.linspace(0,4,5)))
    R = 0
    n = 0
    for i in range(1024):

        # converting decimal number i to binary value
        BIN = np.binary_repr(i,width=10)
        link_state = list(map(int,BIN))

        # applying DFS algorithm to check the conectivity of the graph
        condition = DFS_algorithm(link_state,nodes)

        if((i in K) and k_gen==True):
            condition = bool(~condition)

        if(condition==True):
            n+=1
            R += Reliability(link_state,p)
    return R

def K_var_fun():
    T = []
    p = 0.9
    k_samples = np.arange(0,26,1)
    k = 0
    while(k<=25):
        reliability = 0
        number_of_experiments = 10
        for i in range(number_of_experiments):
            samples = np.random.choice(range(1024),k,replace=False)
            reliability+=Total_reliability(samples,p,True)

        T.append(reliability/number_of_experiments)
        k+=1
    return T,k_samples

def P_var_fun():
```

```
    P = []
    h = 1
    p = np.arange(0,1.04,0.04)
    for t in p:
        k = np.random.choice(range(1024),h,replace=False)
        P.append(Total_reliability(k,t,True))

    return P,p

K,k = K_var_fun()
P,p = P_var_fun()
plt.figure()
plt.fill_between(k,0,K,edgecolor='r',lw=3)
plt.xticks(k)
plt.xlabel('$k$',fontsize=16)
plt.ylabel('$Total \ Reliability$',fontsize=16)
plt.title('$Total \ Reliability \ vs\ K$',fontsize = 16)
plt.grid()
plt.savefig('Total Reliability vs K plot 2.png')
plt.show()

plt.figure()
plt.plot(p,P)
plt.xlabel('$p$',fontsize=16)
plt.ylabel('$Total \ Reliability$',fontsize=16)
plt.title('$Total \ Reliability \ vs\ p$',fontsize = 16)
plt.savefig('Total Reliability vs p.png')
plt.show()
```

READ ME:

1. Python v.3.6.4 is used as a programming software.
2. To run the program, the file is saved into a folder and is imported into the python environment for compilation.
3. Python Jupyter notebook environment is used to compile the program.
4. The module is imported to the notebook, then the module is run to generate the plots.
5. Alternatively, the program can be compiled from the terminal window by running the module using the command ‘import filename’ .