NETWORK RELIABILITY

PRESENTED BY: RONAK SHAH

NET ID: RXS144130

TABLE OF CONTENTS

INTRODUCTION

Our goal is to design an algorithm to evaluate the reliability of a given network configuration. For obtaining this goal, we rely on the PROBABILITY of up and down states of links in the network. We use these probabilities to determine the overall probability of the network. We start by assuming a link reliability for all the links in the network. Then, we consider all the possible combinations in which the given network can be connected with the same number of links. For all these combinations, we calculate the overall reliability of the network. This kind of approach is called exhaustive enumeration and is useful for small network topologies without any particular connectivity pattern like series or parallel or series-parallel.

We also experiment by flipping the states of some links in the network from up to down. Keeping the link reliabilities same, we study the reliability of the network under these network changes.

SIMULATION ENVIORNMENT

NETWORK TOPOLOGY

We simulate the network topology as an undirected graph with 5 nodes and 10 edges. In this topology, every node is connected to every other node, and we avoid parallel and self edges, making the number of edges in the network to be 10. This leads to 2^10 = 1024 possible combinations in which the network topology can be laid out.

LINK RELIABILITY

The link reliability, p, for each node is the same for a particular configuration of the network. For the first part of the experiment, we vary the reliability p between [0.0,1.0] in steps of 0.05. For each of these values of p, we will find the network reliability. For the second part of the experiment, we flip the states of links from k randomly chosen network combinations out of the 1024 combinations. Here, k goes from 0 to 25 and for each of this k values, we find the network reliability.

ALGORITHM

PART I

1. To determine the network reliability, we consider all the possible 1024 combinations of the topology as described above. This can be done by generating binary numbers from 0 to 1023 of 10-bit size. For example, consider 5 nodes as n0, n1, n2, n3, n4

1 0 0 1 0 0 0 1 1 1
0 1 2 3 4 5 6 7 8 9

The first four bits, 0,1,2,3 represent edge state from n0 to n1, n2, n3 and n4 respectively.

The next 3 bits, 4,5,6, represent edge state from n1 to n2, n3,4

The next 2 bits, 7,8, represent edge state from n2 to n3, n4

The bit, 9, represent edge state from n3 to n4

We call this edge_matrix in the Python code.

2. We process edge_matrix to find combinations which represent a connected graph. This is done by transforming the edge-matrix into an adjaceny matrix. Then we run Depth-First-Search to find whether the connected components go through all the nodes in the network, to confirm the network is connected.

3. For each connected combination, we calculate the reliability of that combination using its edge-matrix. This is done as follows

Set reliability as 1.0

For each edge

    If edge is up

        reliability = reliability * p

    if edge is down

        reliability = reliability * 1-p

        where 1-p is the probability that the link is down.

4. We calculate the total network reliability by summing up the reliability for all the 1024 possible network combinations

5. We repeat steps 2-5 for p values in [0.0, 1.0] in steps of 0.05

PART II

1. Same as above

2. For the obtained edge-matrix, we pick k combinations randomly from 1024 and flip their 0 and 1 bits. This results in changing the state of the topology, for each link that was up is now down and each link that was down is now up.

3. For each connected combination, we calculate the reliability of that combination using its edge-matrix. This is done as follows

Set reliability as 1.0

For each edge

    If edge is up

        reliability = reliability * p

    if edge is down

        reliability = reliability * 1-p

        where 1-p is the probability that the link is down.

4. We calculate the total network reliability by summing up the reliability for all the 1024 possible network combinations

5. We repeat steps 2-5 for p value 0.9 and k value ranging from [0,25] in steps of 1

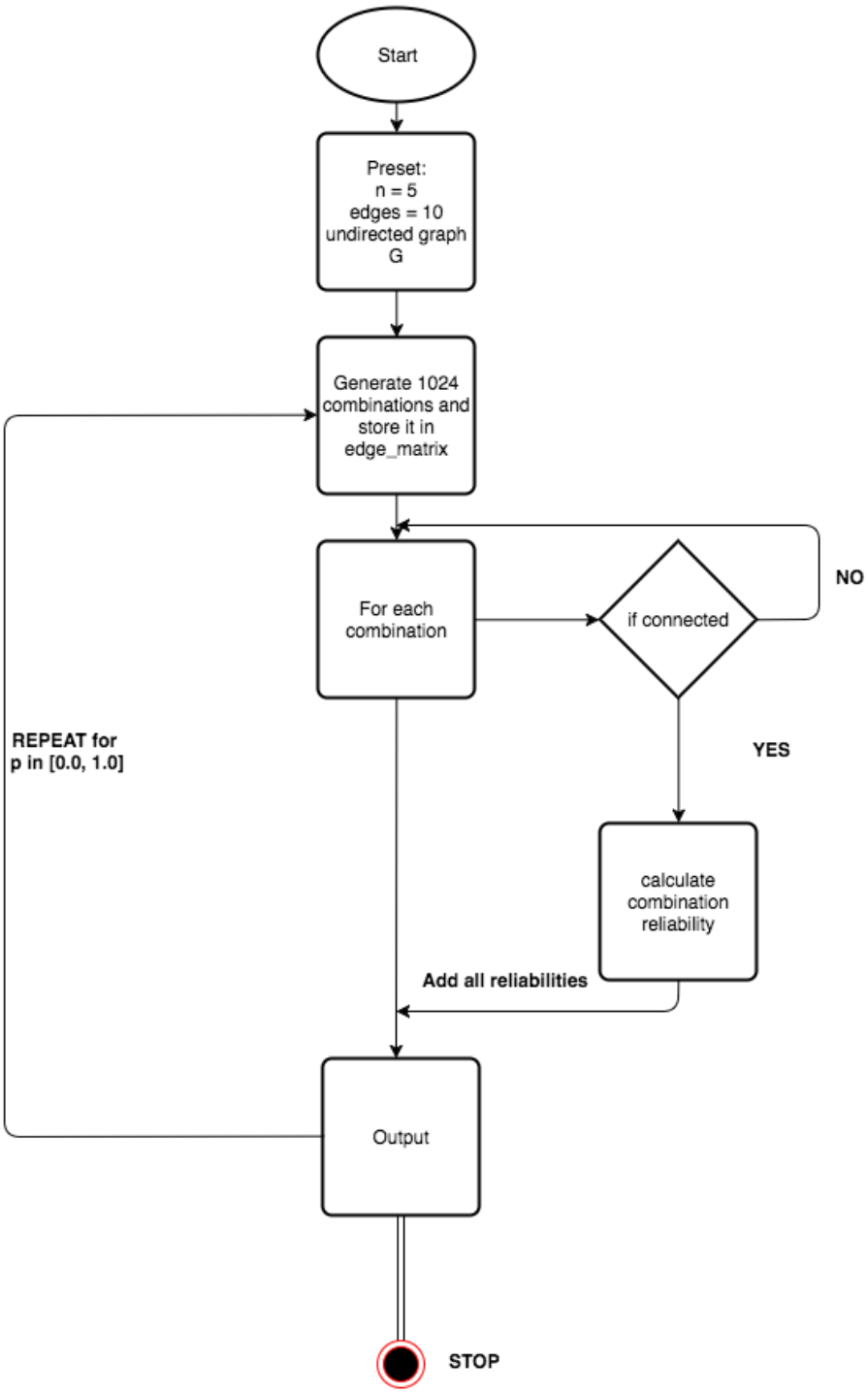

FIGURE 1: FLOW CHART PART I

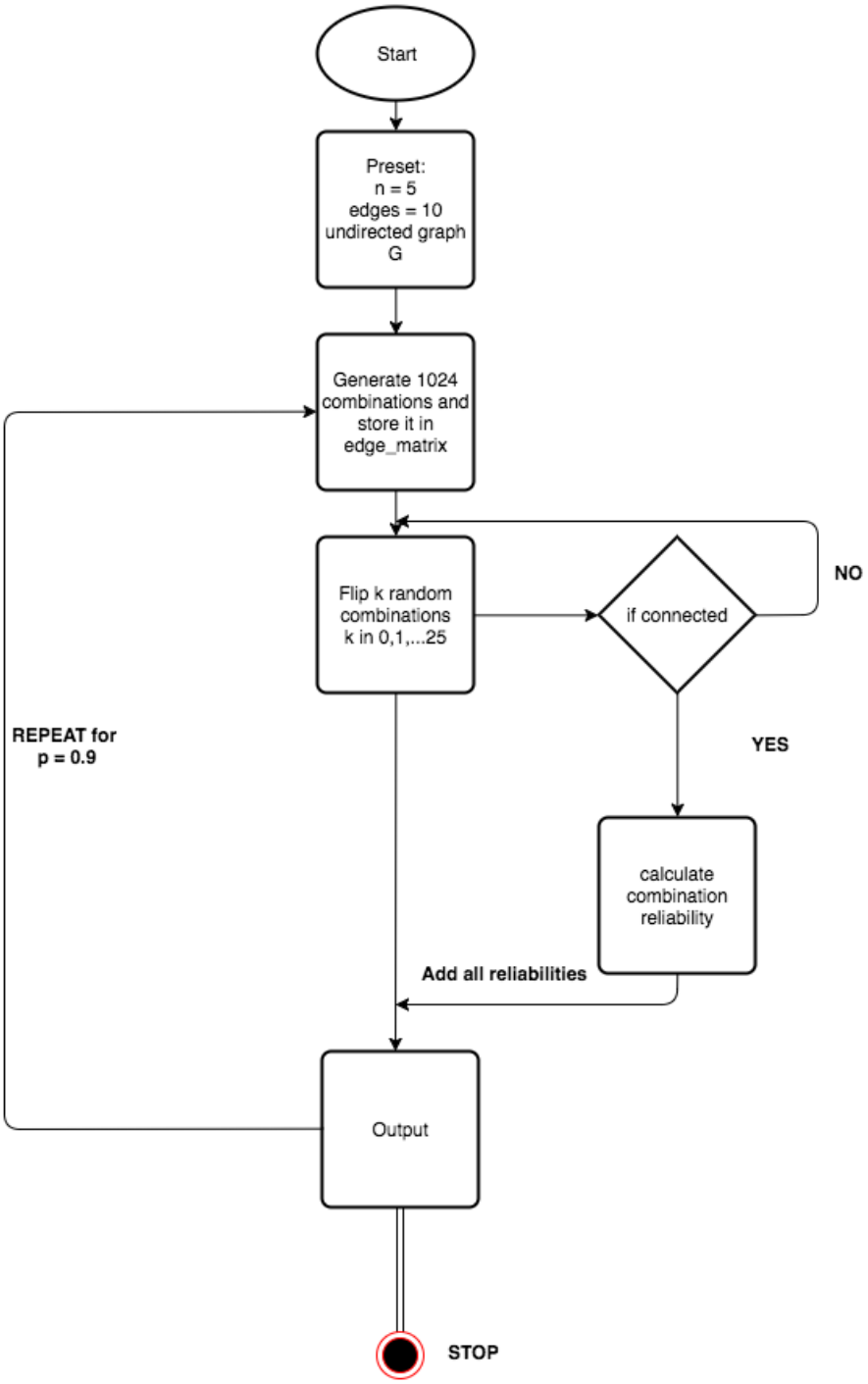

FIGURE 2: FLOW CHART PART II

EXPERIMENT RESULTS

EXPERIMENT 1

GATHERED DATA

p	Network Reliability
0	0
0.05	6.31E-04
0.1	8.10E-03
0.15	3.27E-02
0.2	8.19E-02
0.25	1.58E-01
0.3	2.56E-01
0.35	3.70E-01
0.4	4.90E-01
0.45	6.06E-01
0.5	0.7109375
0.55	0.799881673
0.6	0.870256742
0.65	0.922142692
0.7	0.957513038
0.75	0.979499817
0.8	0.991664538
0.85	0.997394536
0.9	0.999492242
0.95	0.99996861

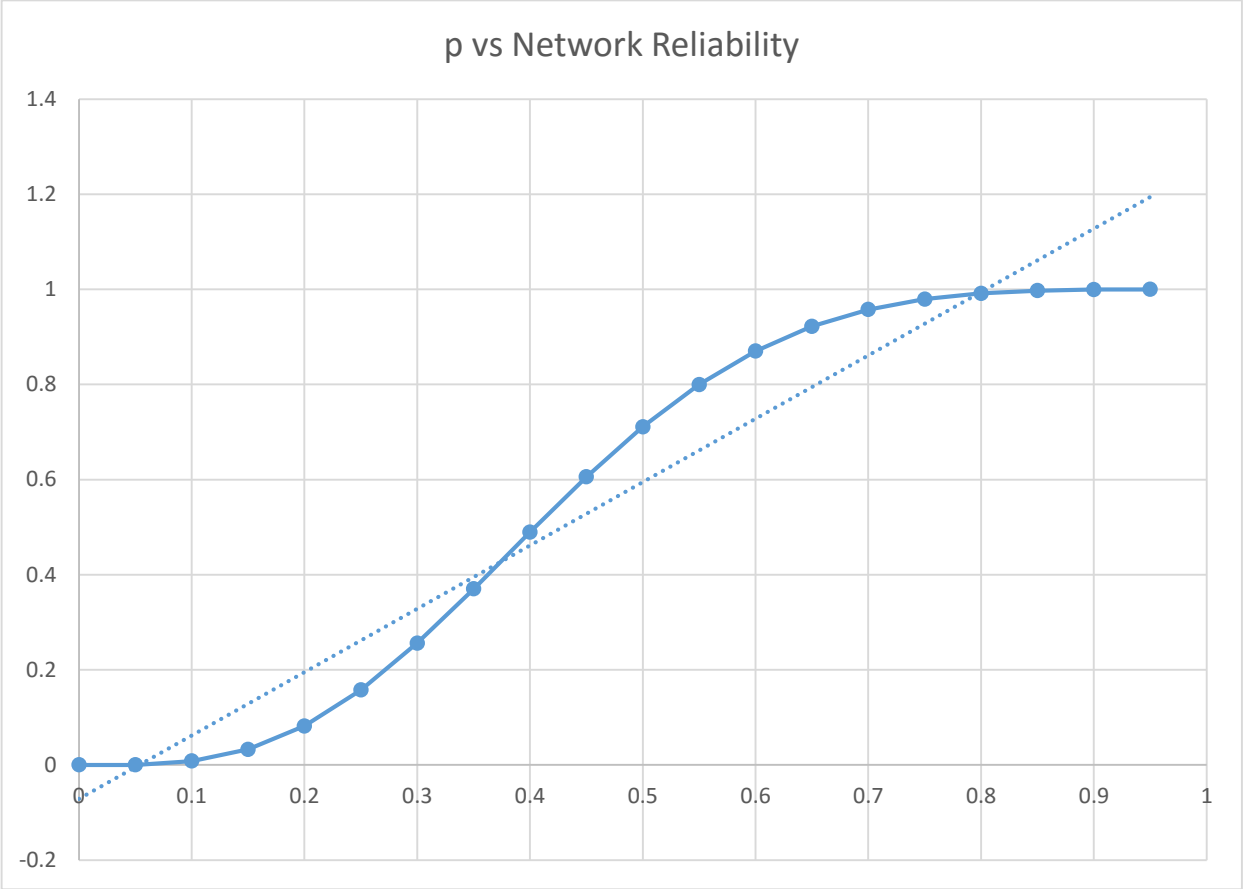

FIGURE 3 GRAPH OF P VS NETWORK RELIABILITY

Correlation 0.968820932

INFERENCE I

- There is a strong correlation between the link probability and the corresponding networking reliability.
- Initially there is a slow start in the improvement in reliability, for smaller values in p.
- After a cut-off, there is a steady increase in reliability for higher values in p
- But, for higher values of p from 0.8, the reliability becomes almost constant
- More denser networks have a better network reliability

These facts state that higher the probability of individual links staying up, better is the reliability of the network. To guarantee a very high availability, we should make sure that the individual links are configured such that they work at least 8/10 times. But to confirm such a claim, we need to run the experiment changing

different parameters like performance of link under low and high traffic, performance with different hardware like co-axial or optical fiber.

EXPERIMENT II

GATHERED DATA

We run this experiment 4 times to reduce the effect of randomness and then averaging the reliability values. P value is fixed at 0.9 and the number of combinations flipped is denoted by k.

K	RELIABILITY				AVERAGE
	RUN 1	RUN 2	RUN 3	RUN 4	
0	0.999492242	0.999492242	0.999492242	0.999492242	0.999492242
1	0.999970539	0.99954473	1.038234291	0.999439754	1.009297329
2	1.000442931	0.999013946	1.038718493	0.999498803	1.009418543
3	1.00006961	1.037277698	1.03829334	0.999551948	1.018798149
4	1.000017122	1.041535787	1.042172204	0.998536305	1.020565354
5	1.004746947	1.041104073	1.045041985	0.999014602	1.022476902
6	1.004694459	1.041314681	0.701152419	0.994551809	0.935428342
7	1.009477428	1.041367169	0.705829756	1.033247931	0.947480571
8	1.013251316	1.036054071	0.705246483	1.06705414	0.955401502
9	1.013675812	1.041262193	0.666983387	1.032668595	0.938647497
10	1.014312885	1.046682235	0.662784347	1.032667283	0.939111687
11	1.010539654	1.04556096	0.632971819	1.042391341	0.932865944
12	1.015376423	1.062364993	0.628140955	1.042018676	0.936975262
13	1.014472974	1.05885289	0.700836835	1.002851474	0.944253543
14	0.972595423	1.058742009	0.693981902	0.999502084	0.931205354
15	0.969457953	1.060760173	0.692756307	0.995622565	0.929649249
16	0.97046113	1.038392411	0.69130895	0.994240162	0.923600663
17	0.966521905	1.039979517	0.730102831	0.988179766	0.931196005
18	0.965081766	1.030252835	0.7159127	0.988816183	0.925015871
19	0.968861558	1.023079037	0.714151072	0.992688485	0.924695038
20	0.970185568	1.024507367	0.717919054	0.953631509	0.916560874
21	0.973644527	1.018927237	0.722808968	0.946025997	0.915351682

22	0.931122686	1.019834623	0.728169961	0.9474622	0.906647367
23	0.9263863	1.019210672	0.726951583	0.939750401	0.903074739
24	0.923399733	0.986221308	0.722758448	0.586870296	0.804812446
25	0.928456952	0.976026826	0.723277423	0.58751065	0.803817963

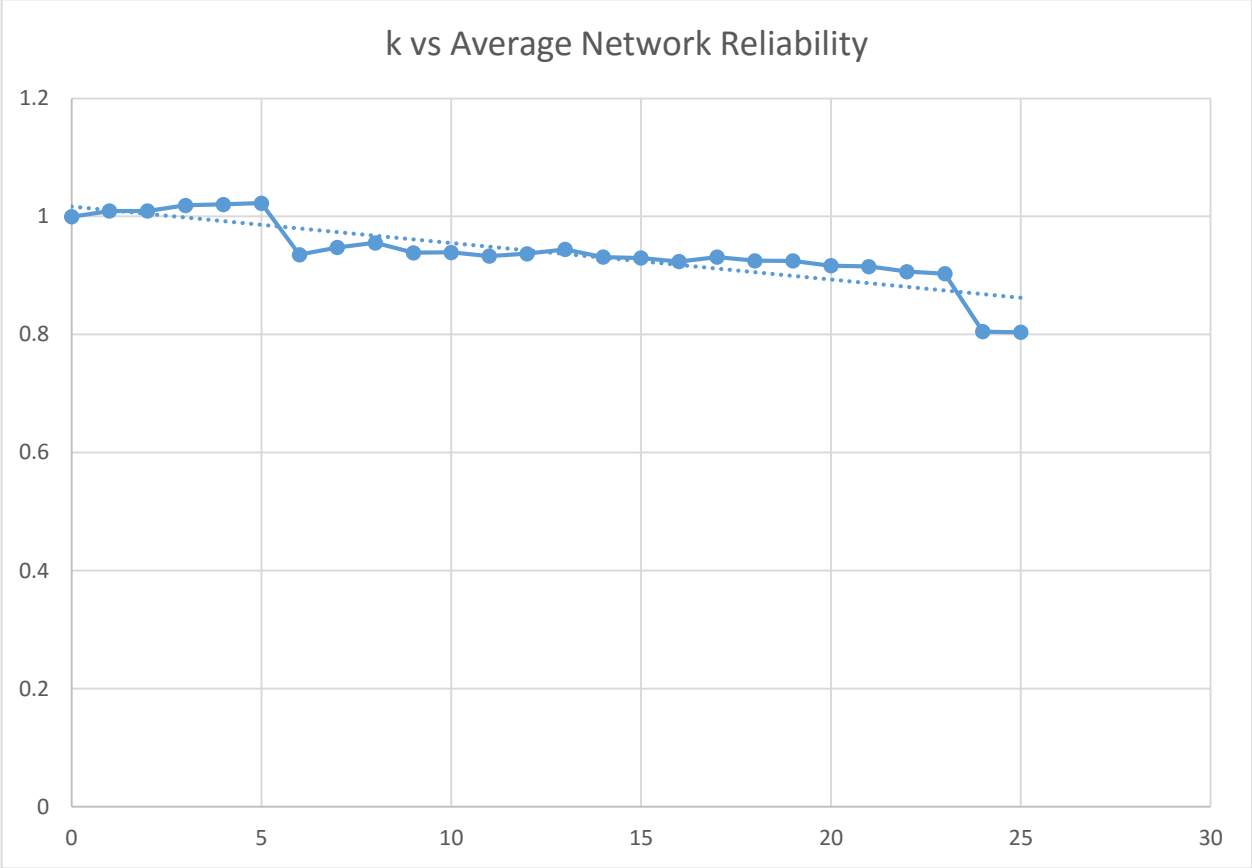

FIGURE 4: GRAPH OF K VS AVERAGE RELIABILITY

INFERENCE II

In this experiment we keep the value of p stable at 0.9. Since this is a very good link reliability value, we can see that the average network reliability is quite high, mostly in the range of 0.8 to 1.0, which also approves with experiment 1.

The flipping of network bits does not have a great effect in these experimental values, but it could have substantial difference if different random values get chose. There is a slight decrease in the reliability as the number of combinations flipped increases. This change can be attributed to the fact that my random sampling seeded from combinations already with high reliability, flipping whom led to reduction in the average reliability.

CONCLUSION

From our experimentation, we can confirm that the overall network reliability can be measured by using a probabilistic model for the up/down states of a link. The better the probability to stay up, the better is the reliability. Also, we learn that denser networks have a better network reliability.

APPENDIX

CODE

Programming language: Python 2.7
File: reliability.py

```
from connected import connected
import random


# Generate 10 bit binary numbers from 0 to 1024

def make_combination():
    edge_mat = [get_combination(i) for i in range(1024)]
    return edge_mat

# Convert given number to a 10 bit binary number

def get_combination(n):
    combination = [i for i in range(10)]
    ne = len(combination)
    for i in range(ne - 1, -1, -1):
        combination[i] = n % 2
        n /= 2
    return combination
```

```
# Calculate the connectivity for the given combination matrix edge_matrix and
# link reliability p
def calc_reliability(edge_mat, p):
    if p == 0.0:
        return 0.0

    comb_reliability = 0.0
    for i in edge_mat:

        adj_mat = [[0 for x in range(5)] for x in range(5)] #generate a 5X5
matrix to transform edge-mat to adj-mat
        adj_i, adj_j = 0, 1
        for j in range(10):

            if adj_j == 5 and adj_i == 0:
                adj_i, adj_j = 1, 2
            if adj_j == 5 and adj_i == 1:
                adj_i, adj_j = 2, 3
            if adj_j == 5 and adj_i == 2:
                adj_i, adj_j = 3, 4

            if adj_j <= 4:
                # print m,',',n
                adj_mat[adj_i][adj_j] = i[j]
                adj_mat[adj_j][adj_i] = i[j]
                adj_j += 1

        if connected(adj_mat):
            comb_reliability += get_reliability(i, p) # sum up the reliability
to find the total network reliability
            # count += 1
            # print True , ' | ', i, ' | ', adj_mat, ' | ', count

    return comb_reliability
    # print  count


def get_reliability(edge_mat, p):
    rel = 1.0
    for col in edge_mat:
```

```
        if col == 1:
            rel = rel * p # p is the probability that the link is up
        else:
            rel = rel * (1.0 - p) # 1-p is the probability that the link is
down
    return rel


def drange(start, stop, step):
    r = start
    while r < stop:
        yield r
        r += step


def k_reliability(edge_mat, k):

    while k>0:
        i = random.randint(0,1023)
        edge_mat[i] = flip(edge_mat[i])
        k -= 1

    return calc_reliability(edge_mat, 0.9)

# toggle state of the link
def flip(edge_mat):

    for i in range(10):
        if edge_mat[i] == 1:
            edge_mat[i] = 0
        elif edge_mat[i] == 0:
            edge_mat[i] = 1
    return edge_mat


def run_exp1(edge_mat):

    print 'p¥t¥tReliability'
    listp = []
    listr = []
    for p in drange(0.0, 1.0, 0.05):
```

```
        listp.append(p)
        r = calc_reliability(edge_mat, p)
        listr.append(r)
        print p, '¥t', r
    # print listp
    # print listr
    print


def run_exp2(edge_mat):
    print 'p=0.9'
    print 'k¥treliability'

    for k in range(26):
        krel = k_reliability(edge_mat, k)
        print k, '¥t', krel


def main():
    edge_mat = make_combination()
    run_exp1(edge_mat)
    run_exp2(edge_mat)


if __name__ == '__main__':
    main()
```

File: connected.py

```
# check if the graph is connected

def connected(adj_mat):

    seen = [False for i in range(5)]

    dfs(adj_mat, seen, 0)

    for i in seen:

        if i is False:
```

```
            return False
    return True


# Performs Depth First Search


def dfs(adj_mat, seen, j):
    seen[j] = True
    for k in range(5):
        if adj_mat[j][k] == 1 and seen[k] == False:
            dfs(adj_mat, seen, k)
```

README

1. Copy paste the code to two separate files as per the given files names above
2. Download and install python 2.7
3. To run, Goto the Dir where above files are located by terminal or command prompt
4. Type commands: python reliability.py

HELPFUL LINKS

- L. E. Miller, J. J. Kelleher, and L. Wong, "Assessment of Network Reliability Calculation Methods," J. S. Lee Associates, Inc. report JC-2097-FF under contract DAAL02-92-C-0045, January 1993.
- Lecture Slides
- Python Documentation