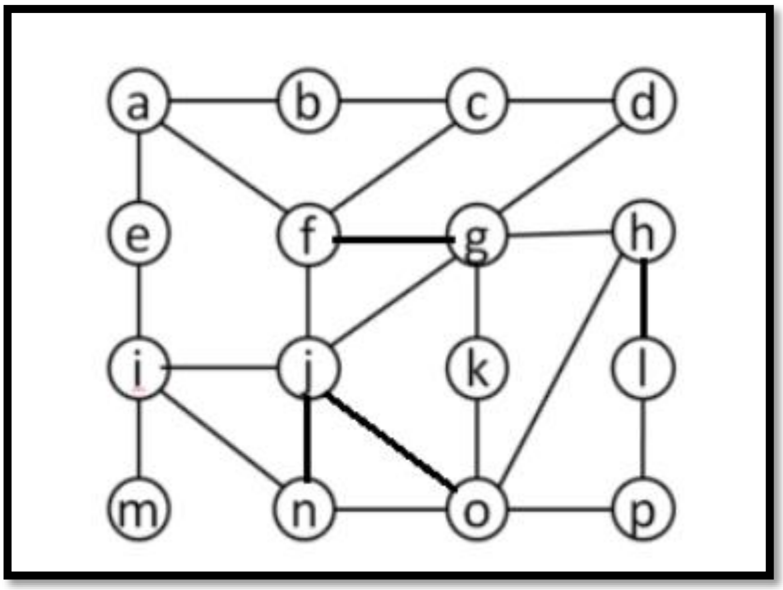
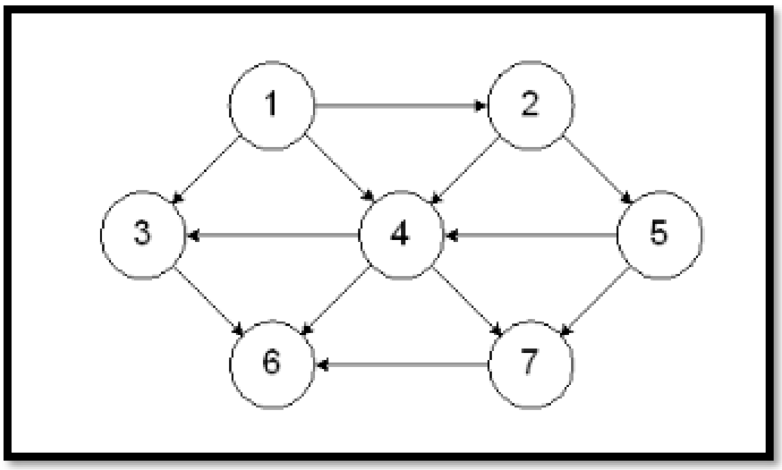
Harshit Mishra (19BCE0799)

CSE-3024 WEB MINING
LAB ASSIGNMENT 6

Aim: Write a python program to find the centrality (degree, closeness and betweenness) and prestige (degree and proximity) for the given graph:





Harshit Mishra (19BCE0799)

Centrality –

Procedure:

- Firstly, we import the networkx and matplotlib libraries.
- Then we instantiate a Graph G, using networkx's .Graph method.
- Then using .add_edge method we create our given graph.
- We then print our graph using matplotlib's .show method.
- To find the degree of each node, we use degree_centrality method of networkx.
- To find the closeness of each node, we use closeness_centrality method of netowrkx.
- To find the betweenness of each node, we use betweenness_centrality method of networkx.

Code:

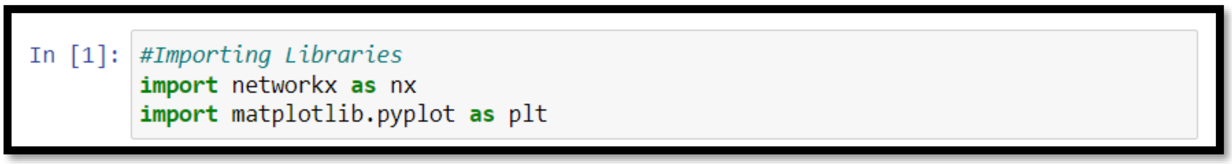
```
#Importing Libraries
import networkx as nx
import matplotlib.pyplot as plt


#Initiating and desgining graph
G = nx.Graph()
G.add_edge('a','b')
G.add_edge('a','e')
G.add_edge('a','f')
G.add_edge('b','c')
G.add_edge('c','d')
G.add_edge('c','f')
G.add_edge('d','g')
G.add_edge('e','i')
G.add_edge('f','g')
G.add_edge('f','j')
G.add_edge('g','h')
G.add_edge('g','j')
```

Harshit Mishra (19BCE0799)

```
G.add_edge('g','k')
G.add_edge('h','l')
G.add_edge('h','o')
G.add_edge('i','j')
G.add_edge('i','n')
G.add_edge('i','m')
G.add_edge('j','n')
G.add_edge('j','o')
G.add_edge('k','o')
G.add_edge('l','p')
G.add_edge('n','o')
G.add_edge('o','p')
nx.draw(G, with_labels=True)
plt.show()


#Closeness - Degree
print("Node\tDegree")
for node in G.nodes():
   print(node, "\t",  nx.degree_centrality(G)[node])


#Closeness - Centrality
print("Node\tCentrality")
for node in G.nodes():
   print(node,"\t", nx.closeness_centrality(G)[node])


#Closeness -Betweeness
print("Node\tBetweeness")
for node in G.nodes():
   print(node, "\t", nx.betweenness_centrality(G)[node])
```

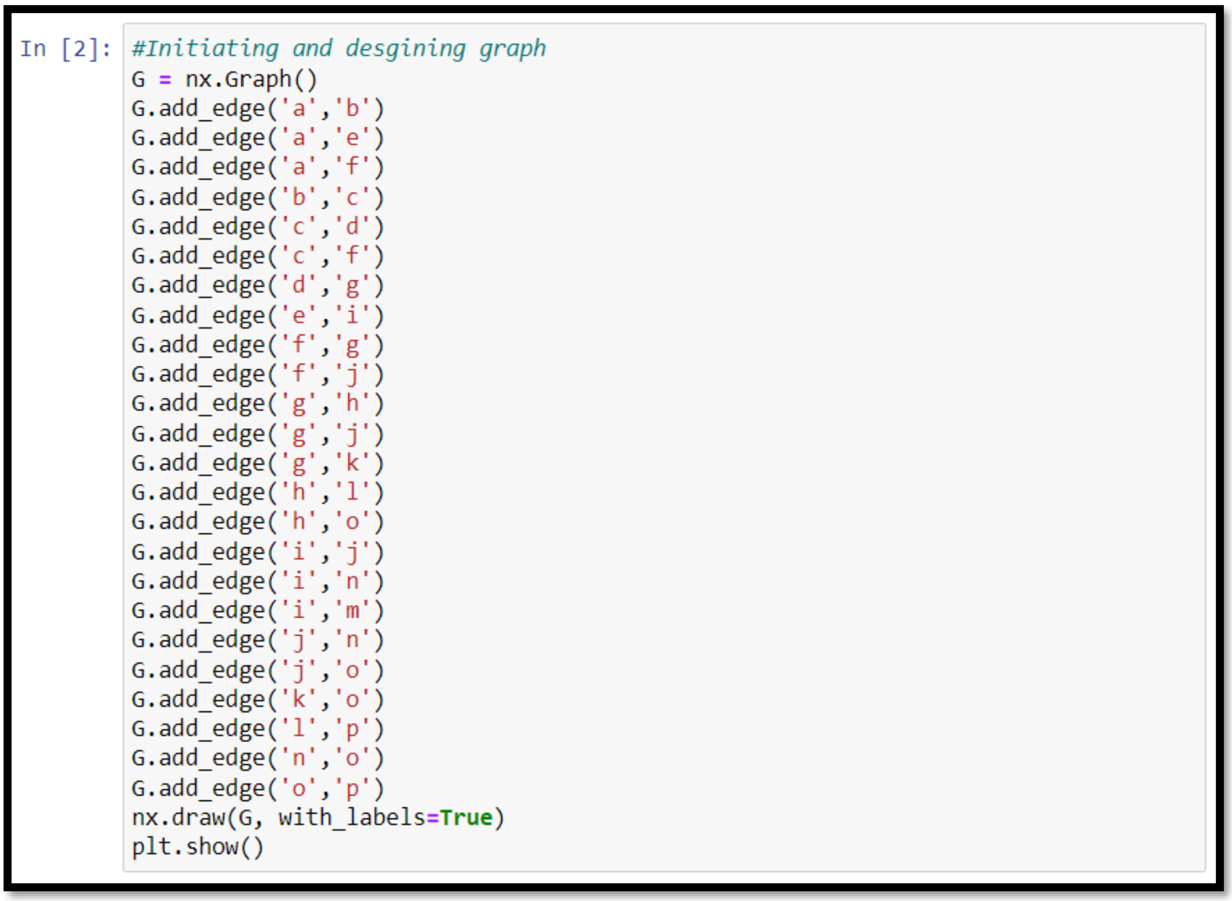
Harshit Mishra (19BCE0799)

Code Snippet and Outputs:

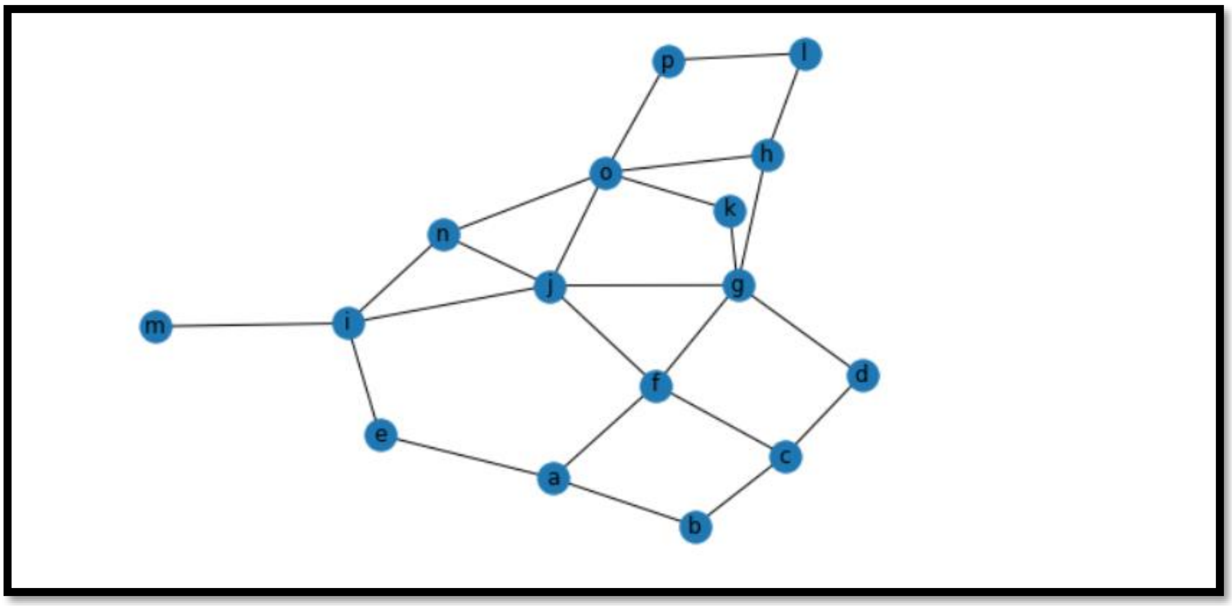


```
In [1]: #Importing Libraries
        import networkx as nx
        import matplotlib.pyplot as plt
```

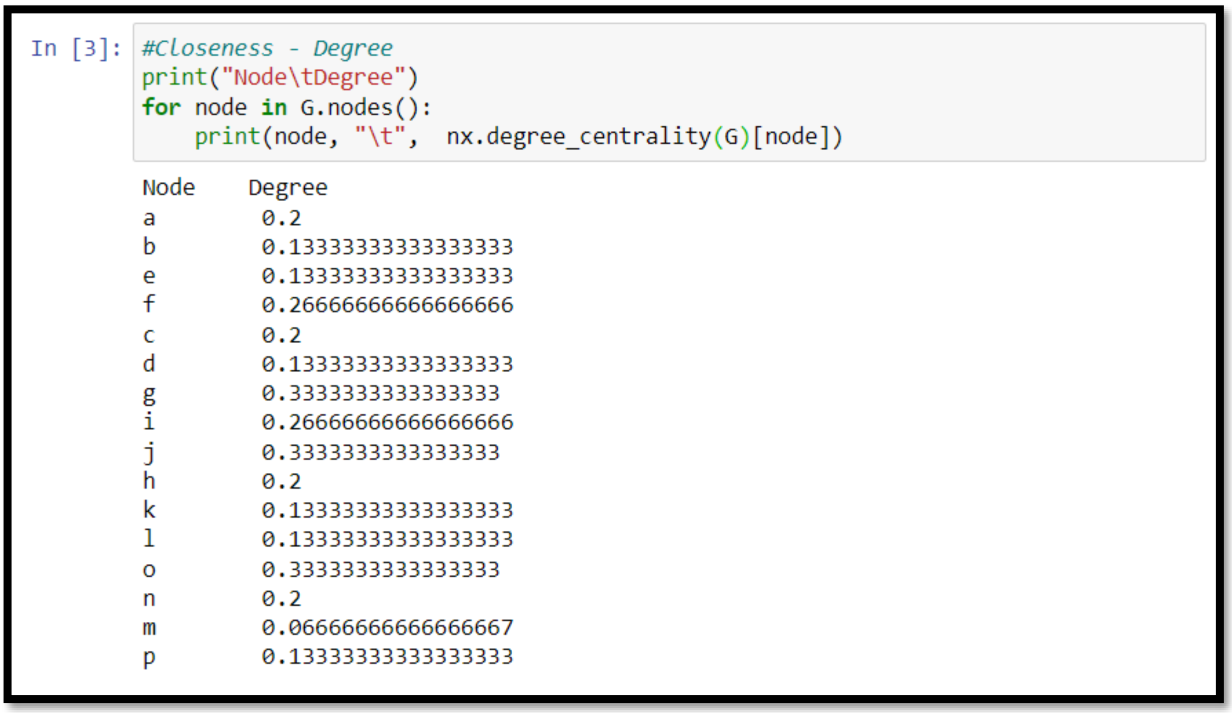
Here we are importing our libraries. We import networkx as nx and matplotlib.pyplot as plt.



```
In [2]: #Initiating and desgining graph
        G = nx.Graph()
        G.add_edge('a','b')
        G.add_edge('a','e')
        G.add_edge('a','f')
        G.add_edge('b','c')
        G.add_edge('c','d')
        G.add_edge('c','f')
        G.add_edge('d','g')
        G.add_edge('e','i')
        G.add_edge('f','g')
        G.add_edge('f','j')
        G.add_edge('g','h')
        G.add_edge('g','j')
        G.add_edge('g','k')
        G.add_edge('h','l')
        G.add_edge('h','o')
        G.add_edge('i','j')
        G.add_edge('i','n')
        G.add_edge('i','m')
        G.add_edge('j','n')
        G.add_edge('j','o')
        G.add_edge('k','o')
        G.add_edge('l','p')
        G.add_edge('n','o')
        G.add_edge('o','p')
        nx.draw(G, with_labels=True)
        plt.show()
```



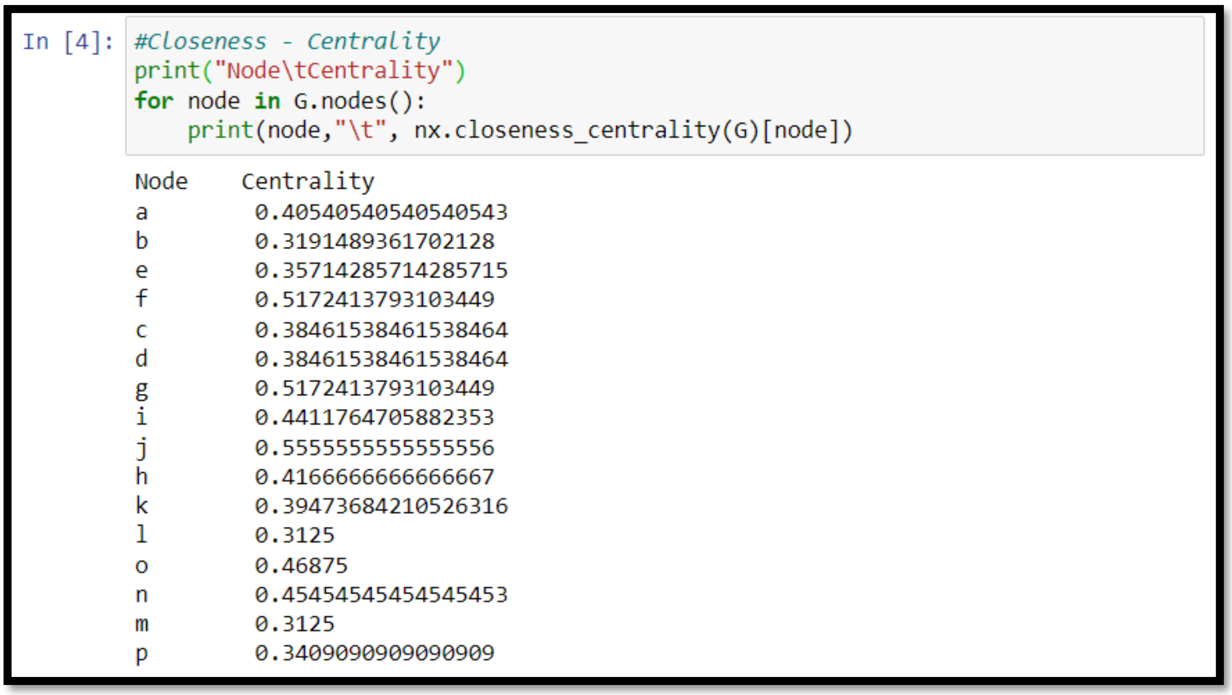
Here we are Initiating our graph using Graph method of networkx and then we make graph by adding each edge to their respective vertices.



```
In [3]: #Closeness - Degree
        print("Node\tDegree")
        for node in G.nodes():
            print(node, "\t",  nx.degree_centrality(G)[node])
```

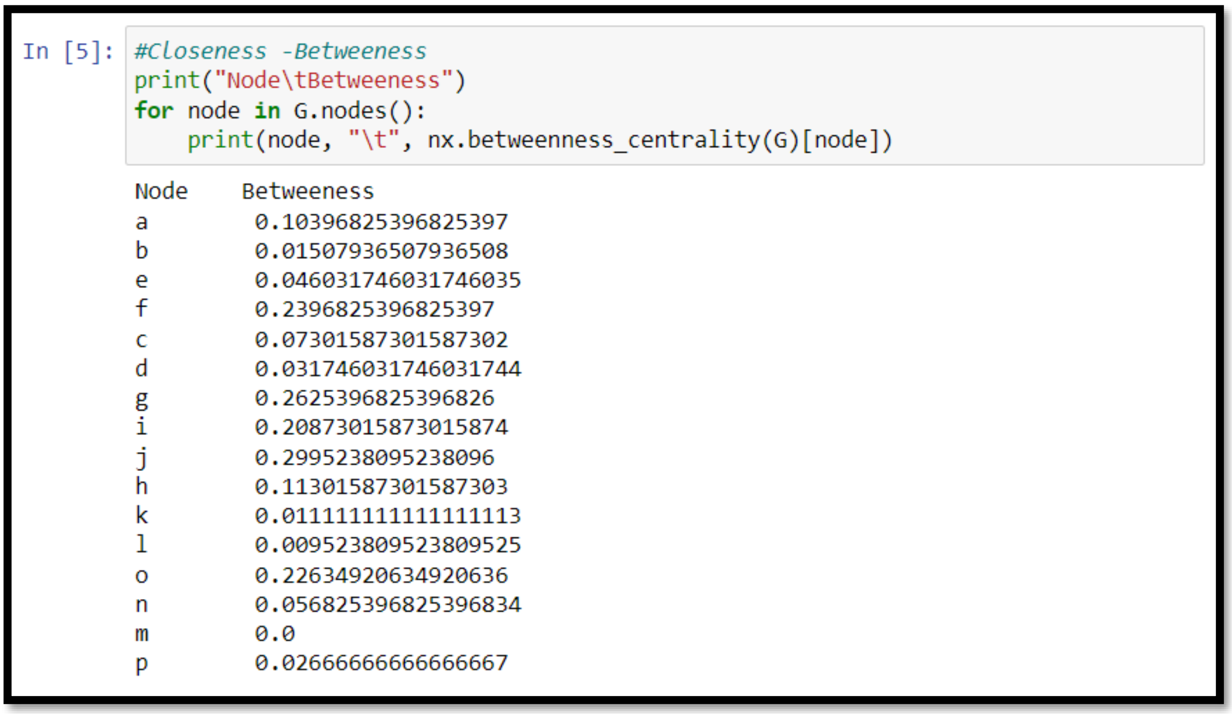
```
Node    Degree
a        0.2
b        0.13333333333333333
e        0.13333333333333333
f        0.26666666666666666
c        0.2
d        0.13333333333333333
g        0.3333333333333333
i        0.26666666666666666
j        0.3333333333333333
h        0.2
k        0.13333333333333333
l        0.13333333333333333
o        0.3333333333333333
n        0.2
m        0.06666666666666667
p        0.13333333333333333
```

Here we are printing the degree of each node using degree_centrality method of networkx.



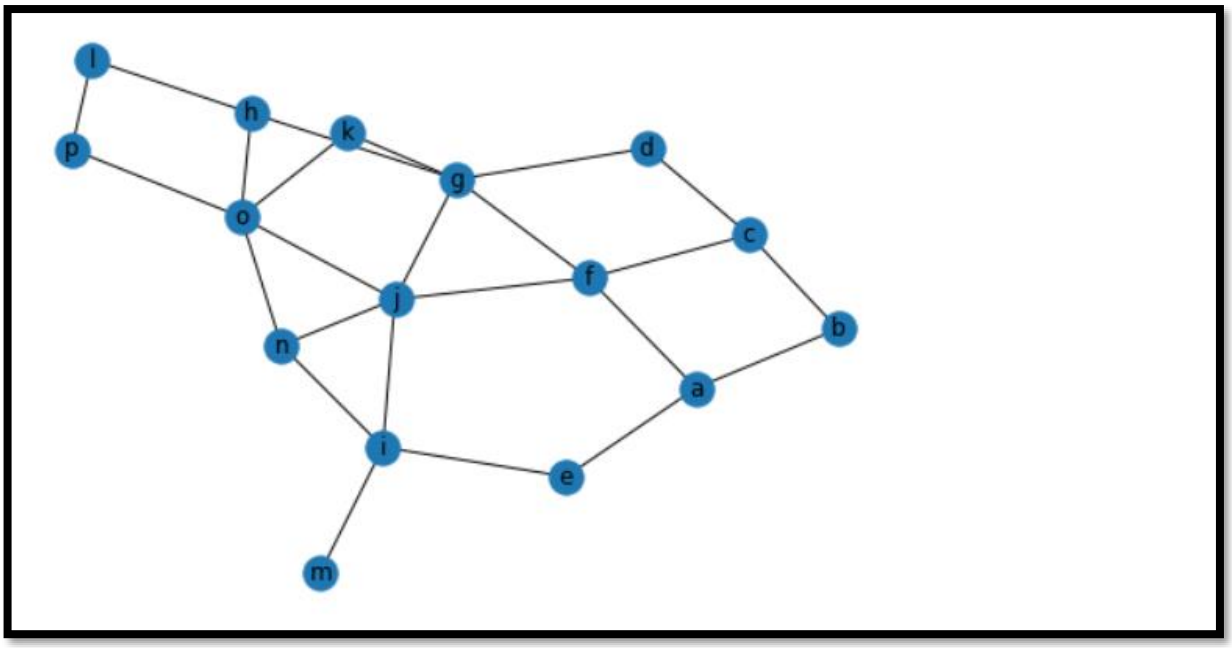
```
In [4]: #Closeness - Centrality
        print("Node\tCentrality")
        for node in G.nodes():
            print(node,"\t", nx.closeness_centrality(G)[node])

        Node    Centrality
        a        0.40540540540540543
        b        0.3191489361702128
        e        0.35714285714285715
        f        0.5172413793103449
        c        0.38461538461538464
        d        0.38461538461538464
        g        0.5172413793103449
        i        0.4411764705882353
        j        0.5555555555555556
        h        0.4166666666666667
        k        0.39473684210526316
        l        0.3125
        o        0.46875
        n        0.45454545454545453
        m        0.3125
        p        0.3409090909090909
```

Here we are printing closeness of each node using closeness_centrality method.

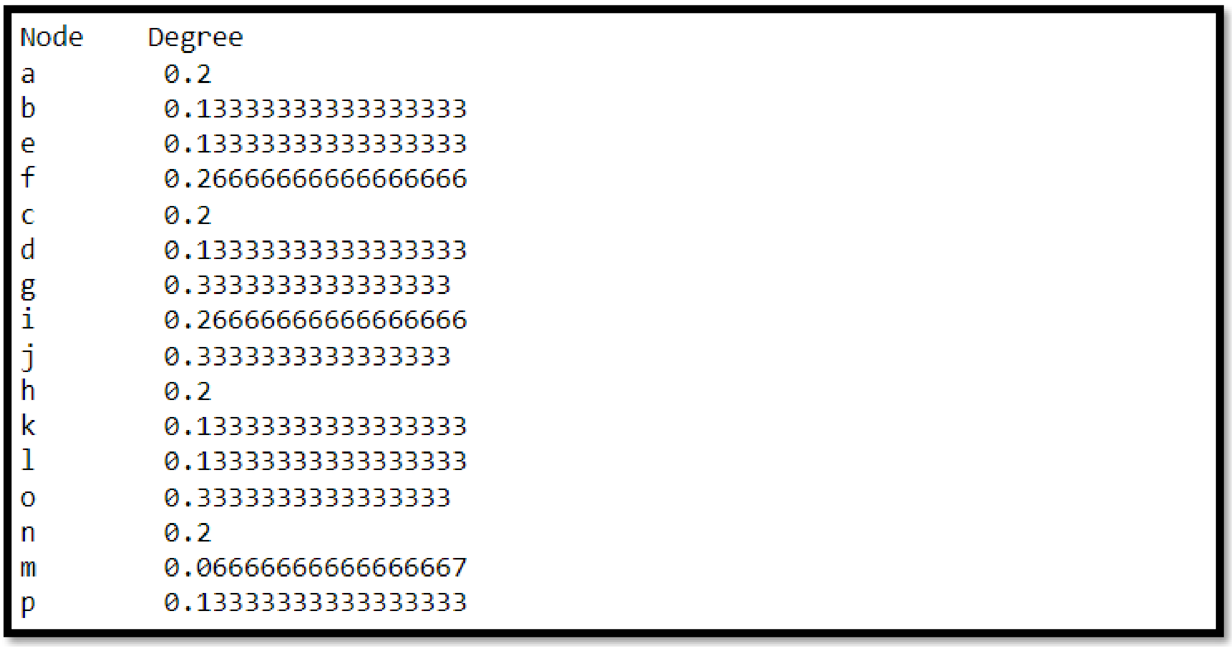


```
In [5]: #Closeness -Betweeness
        print("Node\tBetweeness")
        for node in G.nodes():
            print(node, "\t", nx.betweenness_centrality(G)[node])

        Node    Betweeness
        a        0.10396825396825397
        b        0.01507936507936508
        e        0.046031746031746035
        f        0.2396825396825397
        c        0.07301587301587302
        d        0.031746031746031744
        g        0.2625396825396826
        i        0.20873015873015874
        j        0.2995238095238096
        h        0.11301587301587303
        k        0.011111111111111113
        l        0.009523809523809525
        o        0.22634920634920636
        n        0.056825396825396834
        m        0.0
        p        0.026666666666666667
```

Results:

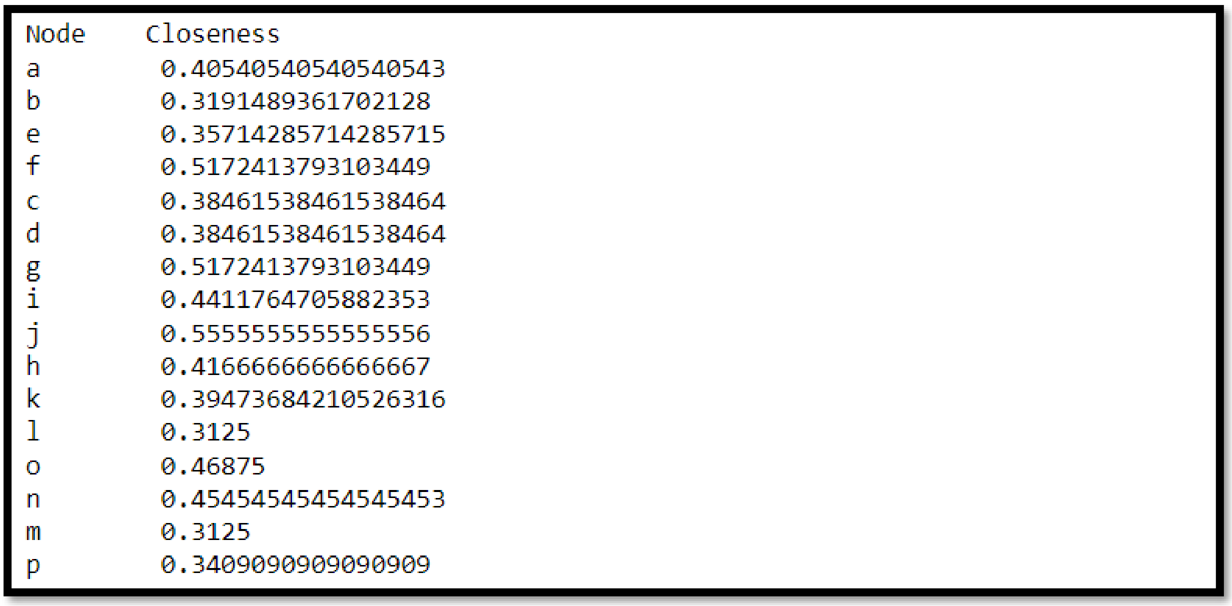


This is the given graph in question. The spatial structure seems to be changed but the edges and node relation remains the same.



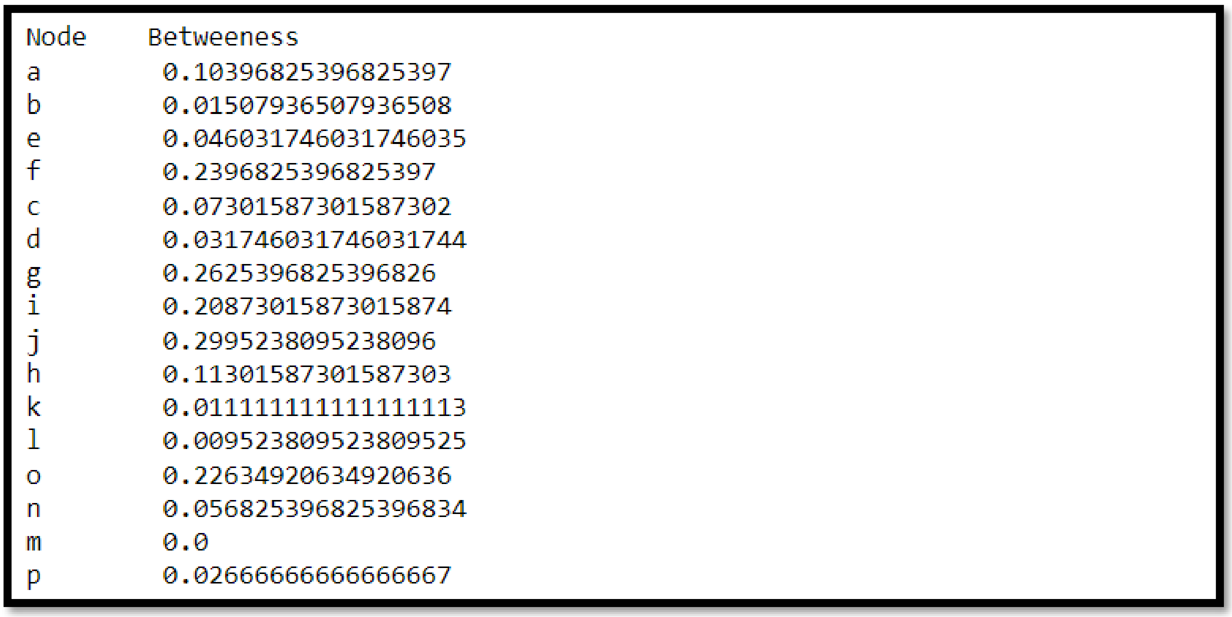
```
Node     Degree
a         0.2
b         0.13333333333333333
e         0.13333333333333333
f         0.26666666666666666
c         0.2
d         0.13333333333333333
g         0.3333333333333333
i         0.26666666666666666
j         0.3333333333333333
h         0.2
k         0.13333333333333333
l         0.13333333333333333
o         0.3333333333333333
n         0.2
m         0.06666666666666667
p         0.13333333333333333
```

Centrality Degree of each node in our graph.



```
Node    Closeness
a        0.40540540540540543
b        0.3191489361702128
e        0.35714285714285715
f        0.5172413793103449
c        0.38461538461538464
d        0.38461538461538464
g        0.5172413793103449
i        0.4411764705882353
j        0.5555555555555556
h        0.4166666666666667
k        0.39473684210526316
l        0.3125
o        0.46875
n        0.45454545454545453
m        0.3125
p        0.3409090909090909
```

Centrality Closeness of each node in our graph.



```
Node    Betweeness
a        0.10396825396825397
b        0.01507936507936508
e        0.046031746031746035
f        0.2396825396825397
c        0.07301587301587302
d        0.031746031746031744
g        0.2625396825396826
i        0.20873015873015874
j        0.2995238095238096
h        0.11301587301587303
k        0.011111111111111113
l        0.009523809523809525
o        0.22634920634920636
n        0.056825396825396834
m        0.0
p        0.02666666666666667
```

Centrality Betweenness of each node in our graph.

Prestige –

Procedure:

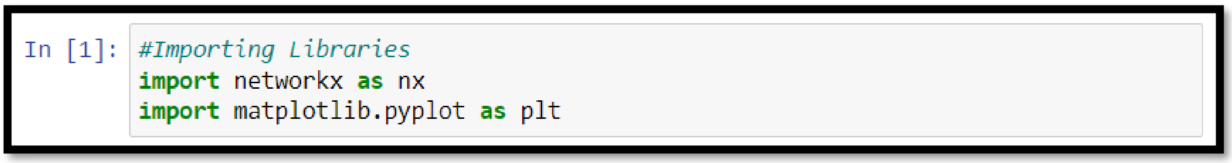
- Firstly, we import the networkx and matplotlib libraries.
- Then we instantiate a Graph G, using networkx's .Graph method.
- Then using .add_edge method we create our given graph.
- We then print our graph using matplotlib's .show method.
- To find degree of each node in our graph, we find the number of incoming edges to each node and then divide it by 6 (number of nodes – 1).
- To find Proximity of each node, we find the distance of that node to each of the other node, then we sum it and finally we divide it by the number of nodes.
- We then print our results.

Code:

```
#Importing Libraries
import networkx as nx
import matplotlib.pyplot as plt


#Initiating and designing graph
G = nx.DiGraph()
G.add_edge('1','2')
G.add_edge('1','3')
G.add_edge('1','4')
G.add_edge('2','4')
G.add_edge('2','5')
G.add_edge('3','6')
G.add_edge('4','3')
G.add_edge('4','6')
G.add_edge('4','7')
G.add_edge('5','4')
G.add_edge('5','7')
G.add_edge('7','6')
nx.draw(G, with_labels=True)
plt.show()
```

```
#Degree Calculation
n_nodes = 7
print("Node\tDegree")
for node in G.nodes():
    print(node, "\t", len(G.in_edges(node))/(n_nodes-1))


#Proximity Calculation
distance = []

temp_dis = 0
n = 0
for dest in G.nodes:
    temp_dis = 0
    n = 0
    for src in G.nodes:
        if (nx.has_path(G,src,dest) == True):
            temp_dis = temp_dis + nx.shortest_path_length(G,source = src,target = dest)
            n = n + 1
    if temp_dis == 0:
        distance.append([dest, 0])
    else:
        distance.append([dest, temp_dis/(n - 1)])

print("Node\tProximity")
for i in distance:
    print(str(i[0]) + " \t " + str(i[1]))
```

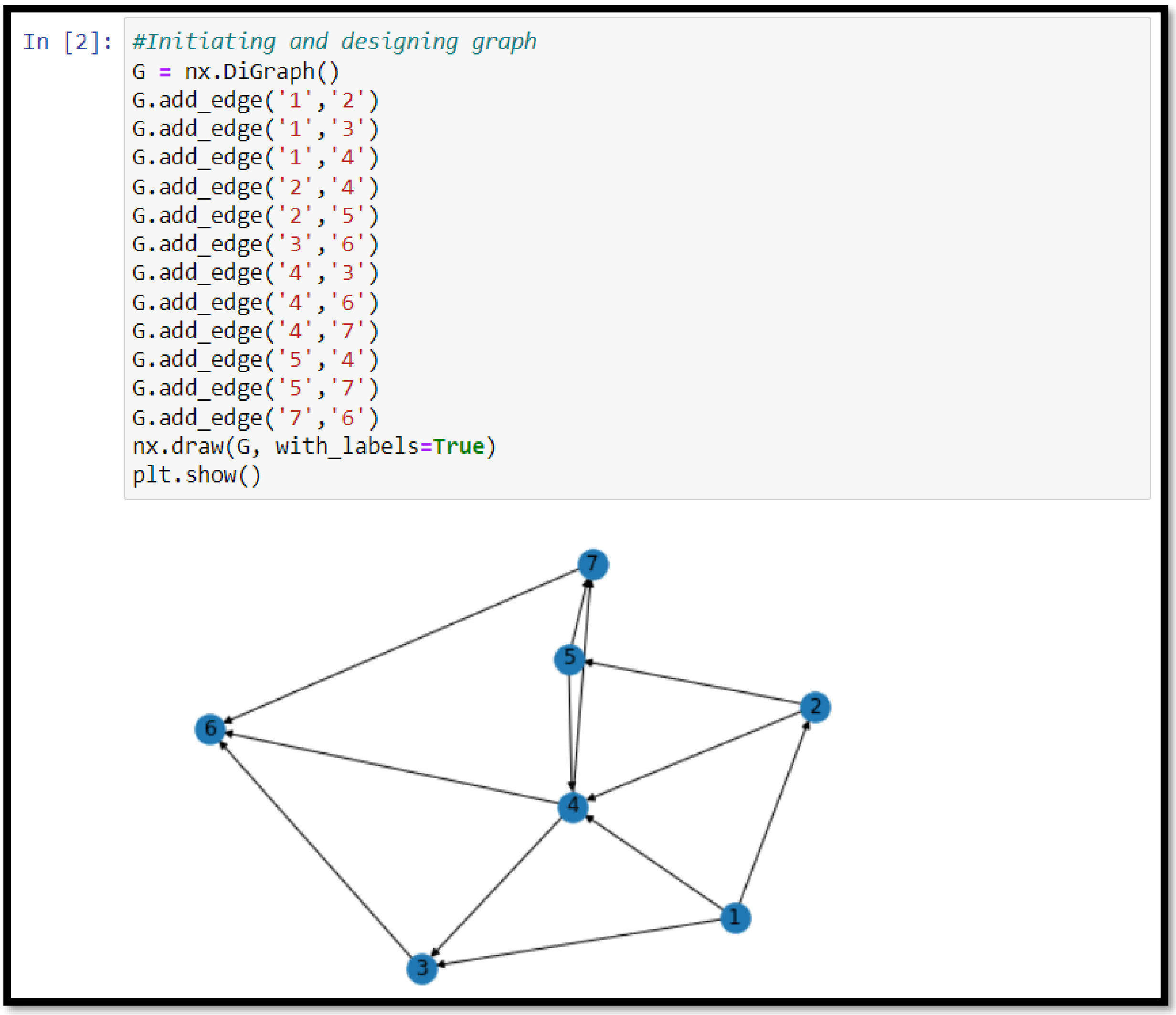
Code Snippets and Outputs:



```
In [1]: #Importing Libraries
        import networkx as nx
        import matplotlib.pyplot as plt
```

Here we are importing our libraries. We import networkx as nx and matplotlib.pyplot as plt.

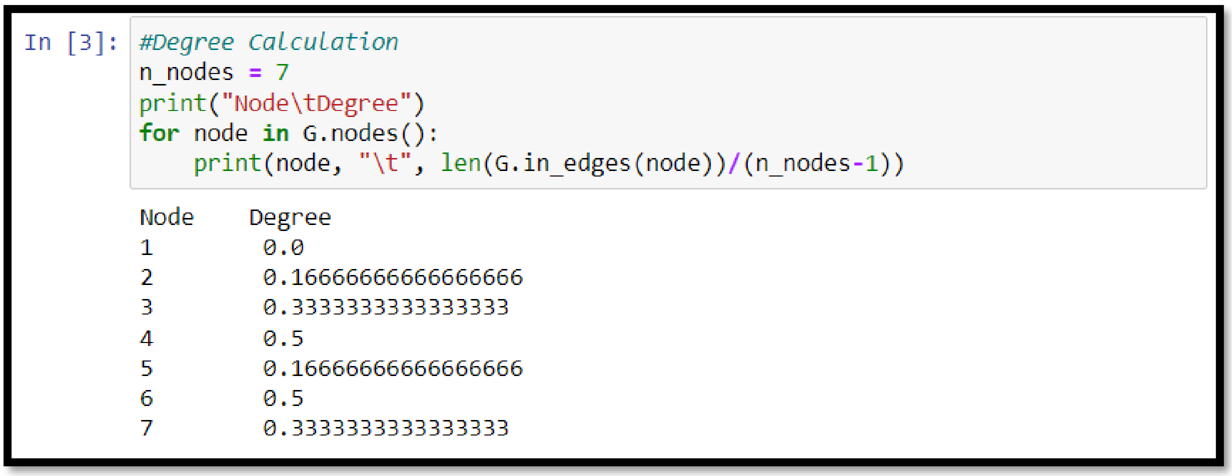
Harshit Mishra (19BCE0799)



```
In [2]: #Initiating and designing graph
        G = nx.DiGraph()
        G.add_edge('1','2')
        G.add_edge('1','3')
        G.add_edge('1','4')
        G.add_edge('2','4')
        G.add_edge('2','5')
        G.add_edge('3','6')
        G.add_edge('4','3')
        G.add_edge('4','6')
        G.add_edge('4','7')
        G.add_edge('5','4')
        G.add_edge('5','7')
        G.add_edge('7','6')
        nx.draw(G, with_labels=True)
        plt.show()
```

Here we initiate our graph using DiGraph method for directional graph. Then we add the edges of our graph using add_edge method.

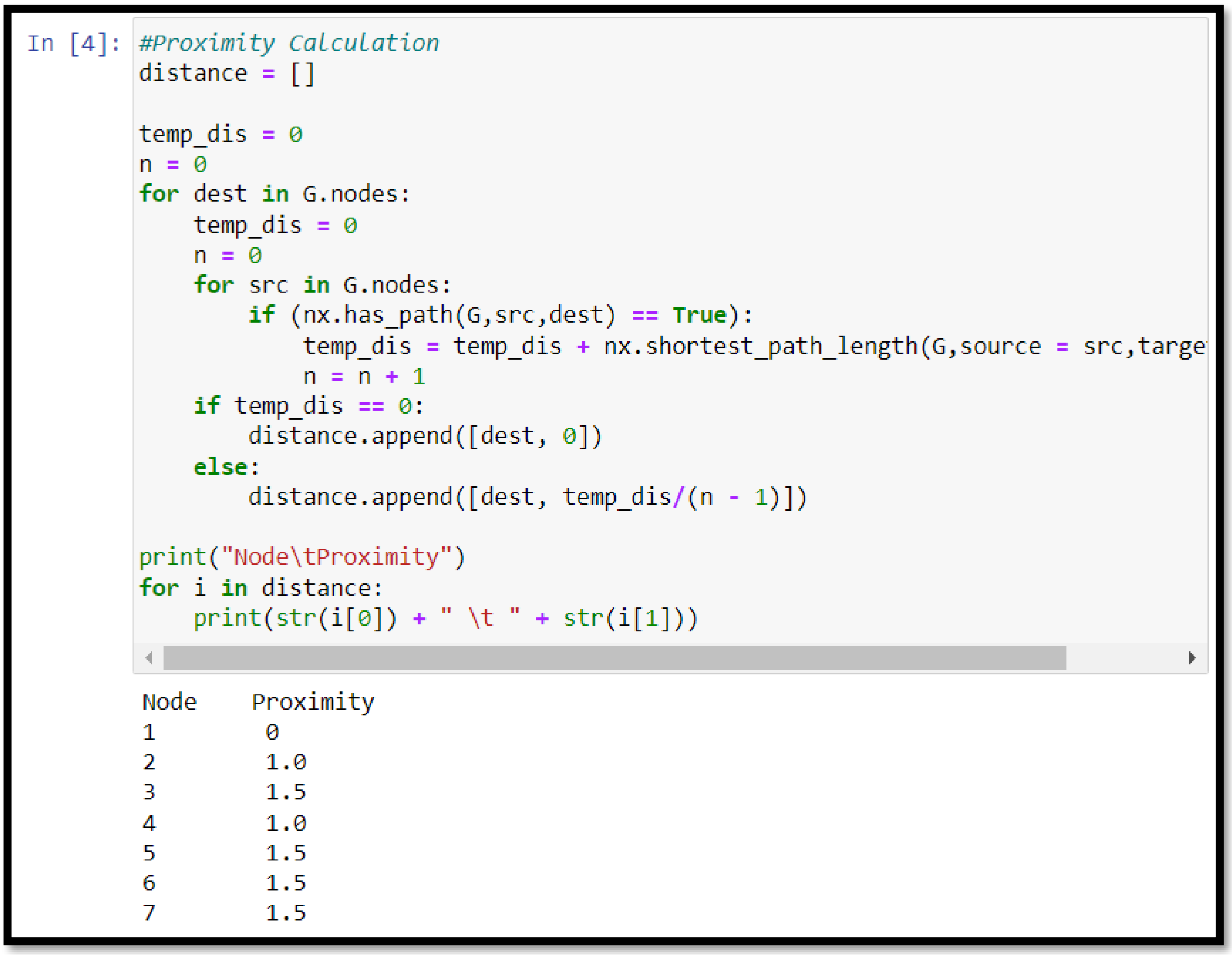
Harshit Mishra (19BCE0799)



```
In [3]: #Degree Calculation
        n_nodes = 7
        print("Node\tDegree")
        for node in G.nodes():
            print(node, "\t", len(G.in_edges(node))/(n_nodes-1))
```

```
Node     Degree
1         0.0
2         0.16666666666666666
3         0.3333333333333333
4         0.5
5         0.16666666666666666
6         0.5
7         0.3333333333333333
```

Here we have printed the Prestige degree of each node in our graph.

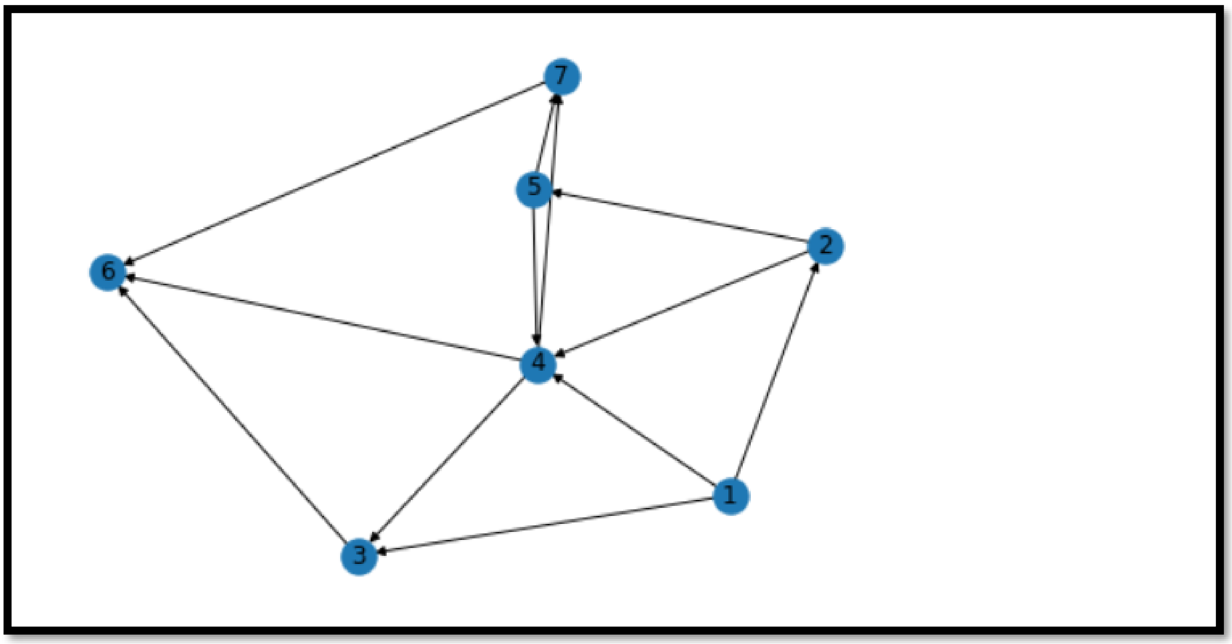


```
In [4]: #Proximity Calculation
        distance = []

        temp_dis = 0
        n = 0
        for dest in G.nodes:
            temp_dis = 0
            n = 0
            for src in G.nodes:
                if (nx.has_path(G,src,dest) == True):
                    temp_dis = temp_dis + nx.shortest_path_length(G,source = src,targe
                    n = n + 1
            if temp_dis == 0:
                distance.append([dest, 0])
            else:
                distance.append([dest, temp_dis/(n - 1)])

        print("Node\tProximity")
        for i in distance:
            print(str(i[0]) + " \t " + str(i[1]))
```

```
Node     Proximity
1         0
2         1.0
3         1.5
4         1.0
5         1.5
6         1.5
7         1.5
```

Here we have calculated the Prestige Proximity of each node in our graph.

Results:



This is our graph.

```
Node      Degree
1          0.0
2          0.16666666666666666
3          0.3333333333333333
4          0.5
5          0.16666666666666666
6          0.5
7          0.3333333333333333
```

This is the Prestige Degree of each node.

```
Node      Proximity
1          0
2          1.0
3          1.5
4          1.0
5          1.5
6          1.5
7          1.5
```

This is the Prestige Proximity of each node.