Jaydeep Dey

20BCE1419

In [ ]:

```
vertices_list = ['1','2','3','4','5','6','7','8','9','10']
```

In [ ]:

```
# !pip install 'networkx<2.7'
# !pip install 'scipy>=1.8'
```

In [ ]:

```
import networkx as nx
import matplotlib.pyplot as plt
```

In [ ]:

```
G = nx.Graph()

G.add_edge('1','2',relation="neighbour")
G.add_edge('1','3',relation="neighbour")
G.add_edge('1','5',relation="neighbour")
G.add_edge('1','6',relation="neighbour")
G.add_edge('1','7',relation="neighbour")
G.add_edge('1','10',relation="neighbour")

G.add_edge('2','1',relation="neighbour")
G.add_edge('2','3',relation="neighbour")
G.add_edge('2','4',relation="neighbour")
G.add_edge('2','8',relation="neighbour")
G.add_edge('2','9',relation="neighbour")
G.add_edge('2','10',relation="neighbour")

G.add_edge('3','1',relation="neighbour")
G.add_edge('3','2',relation="neighbour")
G.add_edge('3','8',relation="neighbour")
G.add_edge('3','9',relation="neighbour")
G.add_edge('3','10',relation="neighbour")

G.add_edge('4','2',relation="neighbour")
G.add_edge('4','7',relation="neighbour")
G.add_edge('4','8',relation="neighbour")
G.add_edge('4','10',relation="neighbour")

G.add_edge('5','1',relation="neighbour")
G.add_edge('5','6',relation="neighbour")
G.add_edge('5','9',relation="neighbour")
G.add_edge('5','10',relation="neighbour")

G.add_edge('6','1',relation="neighbour")
G.add_edge('6','5',relation="neighbour")
G.add_edge('6','8',relation="neighbour")
G.add_edge('6','9',relation="neighbour")

G.add_edge('7','1',relation="neighbour")
G.add_edge('7','4',relation="neighbour")
G.add_edge('7','9',relation="neighbour")
G.add_edge('7','10',relation="neighbour")

G.add_edge('8','2',relation="neighbour")
[illegible]ur")
G.add_edge('8','4',relation="neighbour")
```

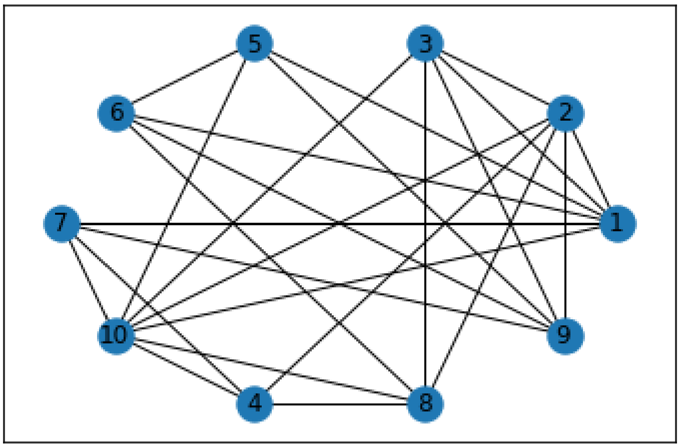
```
G.add_edge('8','6',relation="neighbour")

G.add_edge('9','2',relation="neighbour")
G.add_edge('9','3',relation="neighbour")
G.add_edge('9','5',relation="neighbour")
G.add_edge('9','6',relation="neighbour")
G.add_edge('9','7',relation="neighbour")

G.add_edge('10','1',relation="neighbour")
G.add_edge('10','3',relation="neighbour")
G.add_edge('10','5',relation="neighbour")
G.add_edge('10','8',relation="neighbour")
G.add_edge('10','4',relation="neighbour")
G.add_edge('10','2',relation="neighbour")
G.add_edge('10','7',relation="neighbour")
G.edges(data=True)
```

Out[ ]:

```
EdgeDataView([('1', '2', {'relation': 'neighbour'}), ('1', '3', {'relation': 'neighb
our'}), ('1', '5', {'relation': 'neighbour'}), ('1', '6', {'relation': 'neighbou
r'}), ('1', '7', {'relation': 'neighbour'}), ('1', '10', {'relation': 'neighbour'}),
('2', '3', {'relation': 'neighbour'}), ('2', '4', {'relation': 'neighbour'}), ('2',
'8', {'relation': 'neighbour'}), ('2', '9', {'relation': 'neighbour'}), ('2', '10',
{'relation': 'neighbour'}), ('3', '8', {'relation': 'neighbour'}), ('3', '9', {'rela
tion': 'neighbour'}), ('3', '10', {'relation': 'neighbour'}), ('5', '6', {'relatio
n': 'neighbour'}), ('5', '9', {'relation': 'neighbour'}), ('5', '10', {'relation':
'neighbour'}), ('6', '8', {'relation': 'neighbour'}), ('6', '9', {'relation': 'neigh
bour'}), ('7', '4', {'relation': 'neighbour'}), ('7', '9', {'relation': 'neighbou
r'}), ('7', '10', {'relation': 'neighbour'}), ('10', '4', {'relation': 'neighbou
r'}), ('10', '8', {'relation': 'neighbour'}), ('4', '8', {'relation': 'neighbou
r'})])
```

In [ ]:

```
nx.draw_networkx(G, pos=nx.circular_layout(G),with_labels=True)
plt.show()
```



In [ ]:

```
A = nx.adjacency_matrix(G, nodelist=vertices_list)
print(A.todense())
```

```
[[0 1 1 0 1 1 1 0 0 1]
 [1 0 1 1 0 0 0 1 1 1]
 [1 1 0 0 0 0 0 1 1 1]
 [0 1 0 0 0 0 1 1 0 1]
 [1 0 0 0 0 1 0 0 1 1]
 [1 0 0 0 1 0 0 1 1 0]
 [1 0 0 1 0 0 0 0 1 1]
 [0 1 1 1 0 1 0 0 0 1]
 [0 1 1 0 1 1 1 0 0 0]
 [1 1 1 1 1 0 1 1 0 0]]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [ ]:

```
# degree centrality

print(nx.degree_centrality(G))
```

```
{'1': 0.6666666666666666, '2': 0.6666666666666666, '3': 0.5555555555555556, '5': 0.4
444444444444444, '6': 0.4444444444444444, '7': 0.4444444444444444, '10': 0.777777777
7777777, '4': 0.4444444444444444, '8': 0.5555555555555556, '9': 0.5555555555555556}
```

In [ ]:

```
print("Number of neighbors of node 2:")
G['2']
```

```
Number of neighbors of node 2:
```

Out[ ]:

```
AtlasView({'1': {'relation': 'neighbour'}, '3': {'relation': 'neighbour'}, '4': {'re
lation': 'neighbour'}, '8': {'relation': 'neighbour'}, '9': {'relation': 'neighbou
r'}, '10': {'relation': 'neighbour'}})
```

In [ ]:

```
num_of_vertices = len(vertices_list)
```

In [ ]:

```
#  Average Degree of Graph
print("Average degree of graph is:")
2*G.number_of_edges() / float(num_of_vertices)
```

```
Average degree of graph is:
```

Out[ ]:

```
5.0
```

In [ ]:

```
# Density of graph
print("Density of graph is:")
nx.density(G)
```

```
Density of graph is:
```

Out[ ]:

```
0.5555555555555556
```

In [ ]:

```
# Closeness centrality of Node 10
print("Closeness centrality of node 10 is:");
closeness_centrality = nx.closeness_centrality(G)
closeness_centrality['10']
```

```
Closeness centrality of node 10 is:
```

Out[ ]:

```
0.8181818181818182
```

In [ ]:

```
# Possible path to reach 4 from 6, (min 5 path)
print("All paths from node 4 to 6 are:")
path=nx.all_simple_paths(G,source='4',target='6')

a = list(path)
for i in range(6):
    print(a[i])
```

```
All paths from node 4 to 6 are:
['4', '2', '1', '3', '8', '6']
['4', '2', '1', '3', '8', '10', '5', '6']
['4', '2', '1', '3', '8', '10', '5', '9', '6']
['4', '2', '1', '3', '8', '10', '7', '9', '5', '6']
['4', '2', '1', '3', '8', '10', '7', '9', '6']
['4', '2', '1', '3', '9', '5', '6']
```

In [ ]:

```
# Longest path between any two nodes
def max_length_list(input_list):
    max_length = max(len(x) for x in input_list )
    for i in input_list:
      print(i)
      if len(i) == max_length:
        return(max_length,i)
ShortestPaths=[]
print("Longest Shortest path between any two nodes:");
for i in range(num_of_vertices) :
  for j in range(num_of_vertices):
    ShortestPaths.append(nx.shortest_path(G,source=vertices_list[i],target=vertices_
print(max_length_list(ShortestPaths))
```

```
Longest Shortest path between any two nodes:
['1']
['1', '2']
['1', '3']
['1', '2', '4']
(3, ['1', '2', '4'])
```

In [ ]:

```
# Betweeness Centrality of Node 1
nx.betweenness_centrality(G)['1']
```

Out[ ]:

```
0.08564814814814813
```

In [ ]:

```
# Eigen vector centrality of all node using power Iteration method
nx.eigenvector_centrality(G)
```

Out[ ]:

```
{'1': 0.3624871504024329,
 '2': 0.38240345398629105,
 '3': 0.3396078330961079,
 '5': 0.24994025374803494,
 '6': 0.23334520757872412,
 '7': 0.25599376324062195,
 '10': 0.4186404002649102,
 '4': 0.2647436445750589,
 '8': 0.3159802808874201,
 '9': 0.2817654001541297}
```

Part - B

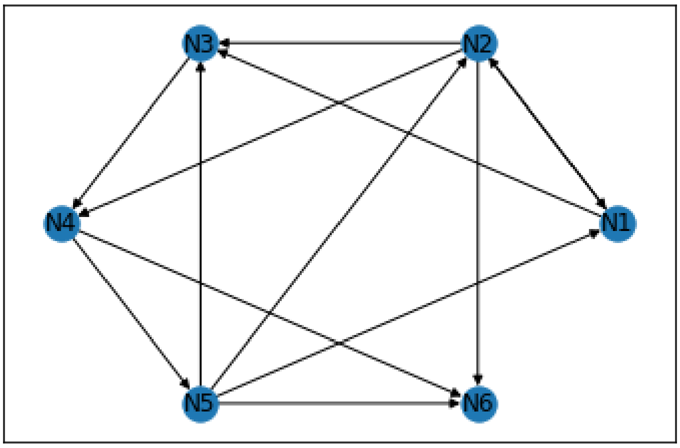
In [ ]:

```
vertices_list1 = ["N1","N2", "N3", "N4", "N5", "N6"]
G1 = nx.DiGraph()

G1.add_nodes_from(vertices_list1)
G1.add_edge("N1","N2",relation="neighbour")
G1.add_edge("N1","N3",relation="neighbour")
G1.add_edge("N2","N4",relation="neighbour")
G1.add_edge("N2","N3",relation="neighbour")
G1.add_edge("N2","N6",relation="neighbour")
G1.add_edge("N2","N4",relation="neighbour")
G1.add_edge("N2","N1",relation="neighbour")
G1.add_edge("N3","N4",relation="neighbour")
G1.add_edge("N4","N5",relation="neighbour")
G1.add_edge("N4","N6",relation="neighbour")
G1.add_edge("N2","N3",relation="neighbour")
G1.add_edge("N5","N1",relation="neighbour")
G1.add_edge("N5","N2",relation="neighbour")
```

```
G1.add_edge("N5","N3",relation="neighbour")
```

In [ ]:

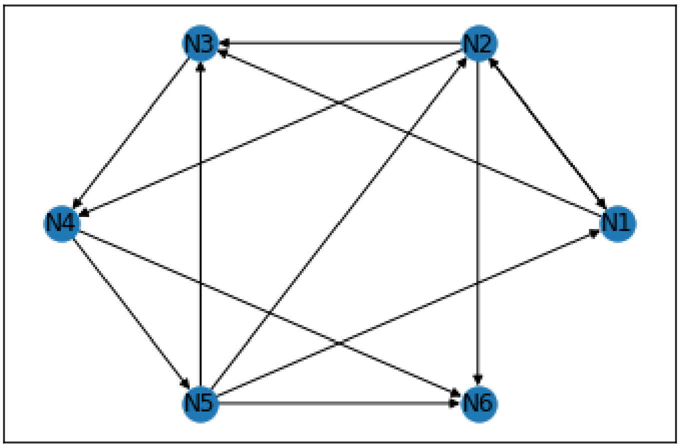
```
A=nx.adjacency_matrix(G1,nodelist=vertices_list1)
nx.draw_networkx(G1, pos=nx.circular_layout(G1), arrows=True, with_labels=True)
```



In [ ]:

```
# (I)   Build a Co-citation coupling matrix. Determine the pair(s) of vertices that
import numpy as np
A1=nx.adjacency_matrix(G1,nodelist=vertices_list1)
nx.draw_networkx(G1, pos=nx.circular_layout(G1), arrows=True, with_labels=True)
adj=np.array(A1.todense())
newrow=[0]*5
newcol=[0]*6
adj= np.vstack([adj, newcol])
zeroes_column = np.zeros((adj.shape[0], 1), dtype=int)
adj = np.hstack((adj, zeroes_column))
adj_t=adj.transpose()
res=np.dot(adj_t,adj)
adj[4][5]=1
print(res)
```

```
[[2 1 2 1 0 2 0]
 [1 2 2 0 0 1 0]
 [2 2 3 1 0 2 0]
 [1 0 1 2 0 1 0]
 [0 0 0 0 1 1 0]
 [2 1 2 1 1 3 0]
 [0 0 0 0 0 0 0]]
```



In [ ]:

```
print(adj)
#Co Citation Matrix
l=list()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
  for j in range(len(res)):
```

```
      if(i==j):
        continue
      if(res[i][j]>1):
        if(i<j):
          l.append([i,j])
      else:
        pass
print("Co Citation Matrix is :")
print(res)
for i in l:
  f=i[0]
  r=i[1]
  temp=[]
  for j in range(len(adj)):
    if(adj[j][f]==1 and adj[j][r]==1):
      temp.append(j+1)
  print(" For Vertices ({},{}) the Pair is ({},{})".format(f+1,r+1,temp[0],temp[1]))
```

```
[[0 1 1 0 0 0 0]
 [1 0 1 1 0 1 0]
 [0 0 0 1 0 0 0]
 [0 0 0 0 1 1 0]
 [1 1 1 0 0 1 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]]
Co Citation Matrix is :
[[2 1 2 1 0 2 0]
 [1 2 2 0 0 1 0]
 [2 2 3 1 0 2 0]
 [1 0 1 2 0 1 0]
 [0 0 0 0 1 1 0]
 [2 1 2 1 1 3 0]
 [0 0 0 0 0 0 0]]
 For Vertices (1,3) the Pair is (2,5)
 For Vertices (1,6) the Pair is (2,5)
 For Vertices (2,3) the Pair is (1,5)
 For Vertices (3,6) the Pair is (2,5)
```

In [ ]:

```
# (II)  Build a Bibliographic Coupling matrix. Determine the pair(s) of vertices tha

ans=np.dot(adj,adj_t)
print("BiblioGraphic Coupling Matrix is ")
print(ans)
l2=list()
for i in range(len(ans)):
  for j in range(len(ans)):
    if(i==j):
      continue
    if(ans[i][j]>1):
      #Symmetric
      if(i<j):
        l2.append([i,j])
    else:
      pass
for i in l2:
  f=i[0]
  r=i[1]
  temp=[]
  for j in range(len(adj)):
    if(adj[f][j]==1 and adj[r][j]==1):
      temp.append(j+1)
  print(" For Vertices ({},{}) the Pair is ({},{})".format(f+1,r+1,temp[0],temp[1]))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
[[2 1 0 0 2 0 0]
```

```
 [1 4 1 1 3 0 0]
 [0 1 1 0 0 0 0]
 [0 1 0 2 1 0 0]
 [2 3 0 1 4 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]]
-----------------------------------------------------
 For Vertices (1,5) the Pair is (2,3)
 For Vertices (2,5) the Pair is (1,3)
```

In [ ]: