accxxzzz

Web Mining Lab Assignment-6

Aryan Vigyat

20BCE1452

In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
!pip install 'networkx<2.7'
!pip install 'scipy>=1.8'
```

```
The system cannot find the file specified.
ERROR: Invalid requirement: "'scipy"
```

Part A

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

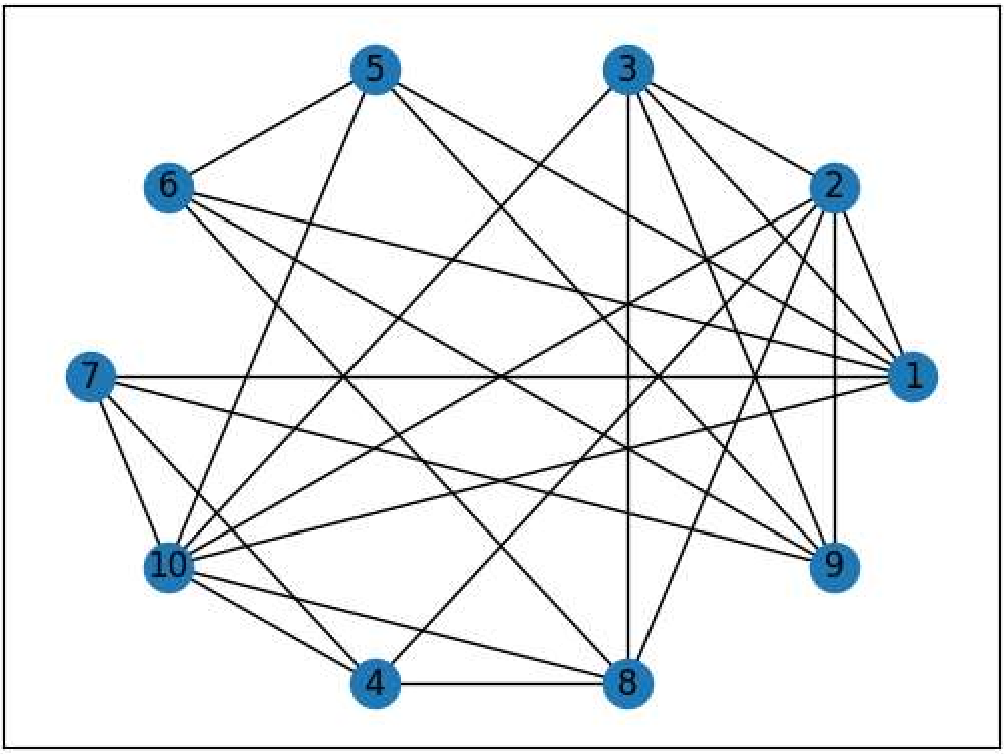
```python
G.add_edge('6','9',relation="neighbour")

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")
G.add_edge('8','3',relation="neighbour")
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': 'neig
hbour'}), ('1', '5', {'relation': 'neighbour'}), ('1', '6', {'relation': 'neighbou
r'}), ('1', '7', {'relation': 'neighbour'}), ('1', '10', {'relation': 'neighbou
r'}), ('2', '3', {'relation': 'neighbour'}), ('2', '4', {'relation': 'neighbou
r'}), ('2', '8', {'relation': 'neighbour'}), ('2', '9', {'relation': 'neighbou
r'}), ('2', '10', {'relation': 'neighbour'}), ('3', '8', {'relation': 'neighbou
r'}), ('3', '9', {'relation': 'neighbour'}), ('3', '10', {'relation': 'neighbou
r'}), ('5', '6', {'relation': 'neighbour'}), ('5', '9', {'relation': 'neighbou
r'}), ('5', '10', {'relation': 'neighbour'}), ('6', '8', {'relation': 'neighbou
r'}), ('6', '9', {'relation': 'neighbour'}), ('7', '4', {'relation': 'neighbou
r'}), ('7', '9', {'relation': 'neighbour'}), ('7', '10', {'relation': 'neighbou
r'}), ('10', '4', {'relation': 'neighbour'}), ('10', '8', {'relation': 'neighbou
r'}), ('4', '8', {'relation': 'neighbour'})])
```

In [ ]:

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



Adjancency Matrix

```
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]]
```

Degree Centrality

```
In [ ]: print("The Degree Centrality is ")
        print(nx.degree_centrality(G))
```

```
The Degree Centrality is
{'1': 0.6666666666666666, '2': 0.6666666666666666, '3': 0.5555555555555556, '5':
0.4444444444444444, '6': 0.4444444444444444, '7': 0.4444444444444444, '10': 0.7777
777777777777, '4': 0.4444444444444444, '8': 0.5555555555555556, '9': 0.55555555555
55556}
```

No of Neighbours of 2

```
In [ ]: print("Number of Neighbours of Verticle 2:")
        G['2']
```

```
Number of Neighbours of Verticle 2:
```

```
Out[ ]: AtlasView({'1': {'relation': 'neighbour'}, '3': {'relation': 'neighbour'}, '4':
{'relation': 'neighbour'}, '8': {'relation': 'neighbour'}, '9': {'relation': 'neig
hbour'}, '10': {'relation': 'neighbour'}})
```

Average Degree

In [ ]:
```
total_nodes = len(vertices_list)
#  Average Degree of Graph
print("Average degree of graph is:")
2*G.number_of_edges() / float(total_nodes)
```

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

Out[ ]:
```
5.0
```

Density of the Graph

In [ ]:
```
print("Density of graph is:")
round(nx.density(G),2)
```

```
Density of graph is:
```

Out[ ]:
```
0.56
```

Closeness Centrality of Node 10

In [ ]:
```
print("Closeness Centrality of Node 10 is:");
closeness_centrality = nx.closeness_centrality(G)
round(closeness_centrality['10'],2)
```

```
Closeness Centrality of Node 10 is:
```

Out[ ]:
```
0.82
```

Possible paths to reach 4 from 6, (min 5 path)

In [ ]:
```
print("All the paths starting from Node 4 to 6 are:")
path=nx.all_simple_paths(G,source='4',target='6')

a = list(path)
for i in range(10):
    print(("->").join(a[i]))
```

```
All the paths starting 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
4->2->1->3->9->5->10->8->6
4->2->1->3->9->6
4->2->1->3->9->7->10->5->6
4->2->1->3->9->7->10->8->6
```

Longest path between any two nodes

In [ ]:
```
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(total_nodes) :
  for j in range(total_nodes):
    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'])
```

Betweeness Centrality of Node 1

In [ ]:
```
print("The Betweenness Centrality is ")
round(nx.betweenness_centrality(G)['1'],2)
```

```
The Betweenness Centrality is
```

Out[ ]:
```
0.09
```

Eigen vector centrality of all node using power Iteration method

In [ ]:
```
print("The Eigen Vector Centrality is ")
d=nx.eigenvector_centrality(G,45)
sorted_dict = dict(sorted(d.items(), key=lambda item: item[1], reverse=True))
print(sorted_dict)
first_item = next(iter(sorted_dict.items()))
print("The Most Influential Node is ")
print(first_item)
```

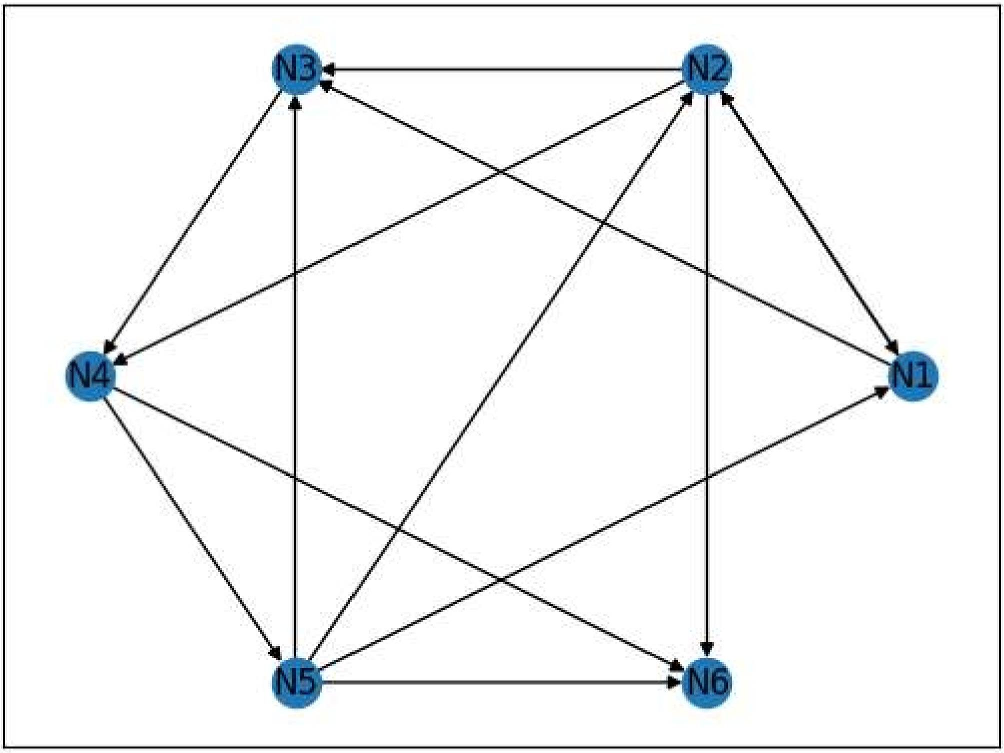
```
The Eigen Vector Centrality is
{'10': 0.4186404002649102, '2': 0.38240345398629105, '1': 0.36248715040243285,
'3': 0.3396078330961079, '8': 0.3159802808874201, '9': 0.2817654001541297, '4': 0.
264743644575059, '7': 0.255993763240622, '5': 0.24994025374803494, '6': 0.23334520
757872415}
The Most Influential Node is
('10', 0.4186404002649102)
```

PART B

In [ ]:
```
G=nx.DiGraph()
v=["N1","N2","N3","N4","N5"]
G.add_nodes_from(v)
G.add_edge("N1","N2",relation="neighbour")
G.add_edge("N1","N3",relation="neighbour")
G.add_edge("N2","N4",relation="neighbour")
G.add_edge("N2","N3",relation="neighbour")
G.add_edge("N2","N6",relation="neighbour")
G.add_edge("N2","N4",relation="neighbour")
G.add_edge("N2","N1",relation="neighbour")
G.add_edge("N3","N4",relation="neighbour")
G.add_edge("N4","N5",relation="neighbour")
G.add_edge("N4","N6",relation="neighbour")
G.add_edge("N2","N3",relation="neighbour")
G.add_edge("N5","N1",relation="neighbour")
G.add_edge("N5","N2",relation="neighbour")
G.add_edge("N5","N3",relation="neighbour")
G.add_edge("N5","N6",relation="neighbour")
A=nx.adjacency_matrix(G,nodelist=v)
nx.draw_networkx(G, pos=nx.circular_layout(G), arrows=True, with_labels=True)
adj=np.array(A.todense())
```

```python
newrow=[0]*5
newcol=[0]*6
adj= np.vstack([adj, newrow])
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 0]
 [1 2 2 0 0 0]
 [2 2 3 1 0 0]
 [1 0 1 2 0 0]
 [0 0 0 0 1 0]
 [0 0 0 0 0 0]]
```



In [ ]:
```python
print("Adjacency Matrix is ")
print(adj)
#Co Citation Matrix
l=list()
for i in range(len(res)):
  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)
print("-----------------------------------------------------")
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]
```

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

In [ ]:

```
# For BiblioGraphy Matrix
ans=np.dot(adj,adj_t)
print("BiblioGraphic Coupling Matrix is ")
print(ans)
print("-----------------------------------------------------")
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]
```

```
BiblioGraphic Coupling Matrix is
[[2 1 0 0 2 0]
 [1 3 1 0 2 0]
 [0 1 1 0 0 0]
 [0 0 0 1 0 0]
 [2 2 0 0 4 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)
```