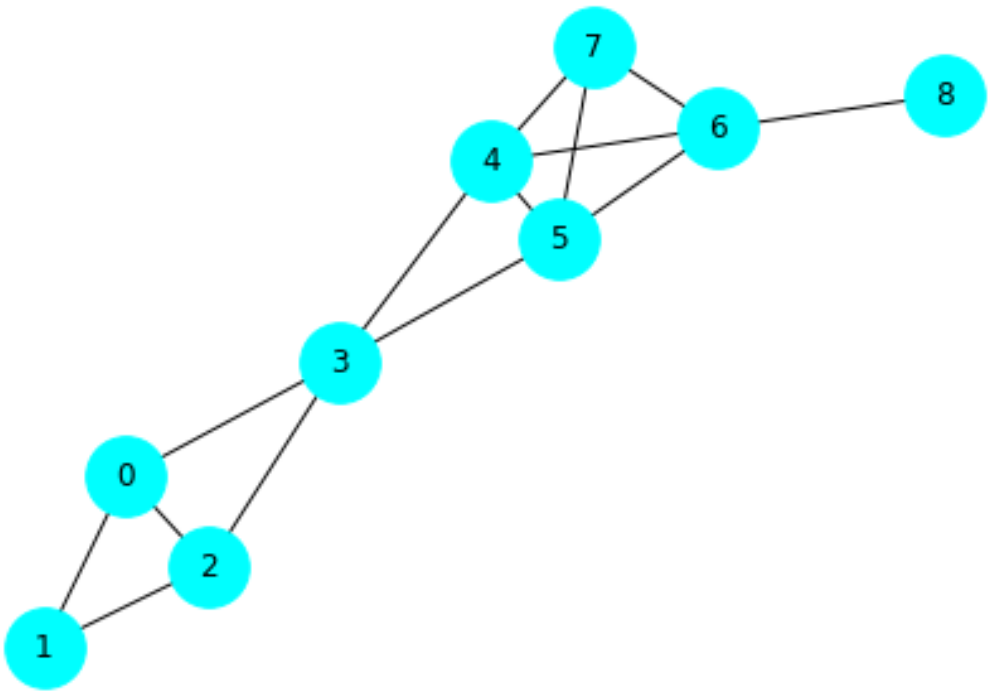
22mcb1002-lab4

March 12, 2023

[1]:
```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
```

[11]:
```
adj_mat = np.array([[0,1,1,1,0,0,0,0,0],
                    [1,0,1,0,0,0,0,0,0],
                    [1,1,0,1,0,0,0,0,0],
                    [1,0,1,0,1,1,0,0,0],
                    [0,0,0,1,0,1,1,1,0],
                    [0,0,0,1,1,0,1,1,0],
                    [0,0,0,0,1,1,0,1,1],
                    [0,0,0,0,1,1,1,0,0],
                    [0,0,0,0,0,0,1,0,0]])
G = nx.from_numpy_array(adj_mat)
nx.draw(G, with_labels = True, node_color = 'cyan', node_size = 1000)
```



1 Measures without NetworkX

A. Degree Centrality

[ ]:
```
DC = {}
for node in G:
    degree = len(G[node])
    DC[node] = degree / (len(G) - 1)

df = pd.DataFrame.from_dict(DC, orient = 'index', columns = [' Degree␣
 ↪Centrality'])
print(df)
```

```
   Degree Centrality
0              0.375
1              0.250
2              0.375
3              0.500
4              0.500
5              0.500
6              0.500
7              0.375
8              0.125
```

B. Degree Centralization

[ ]:
```
max_deg = len(G) - 1
sum_of_diff = sum(max_deg - len(G[node]) for node in G)
degree_centralization = sum_of_diff / (max_deg * (len(G) - 2))

print("Degree Centralization: ",round(degree_centralization, 4))
```

```
Degree Centralization:  0.7857
```

C. Average Degree

[ ]:
```
total_deg = sum(len(G[node]) for node in G)
avg_deg = total_deg / len(G)

print("Average Degree: ", round(avg_deg, 3))
```

```
Average Degree:  3.111
```

D. Betweeness centrality of vertex

[ ]:
```
def betweenness_centrality(graph):
    betweenness = dict.fromkeys(graph, 0.0)
    for node in graph:
        stack = []
        pred = {w: [] for w in graph}
        sigma = dict.fromkeys(graph, 0)
        sigma[node] = 1
        dist = dict.fromkeys(graph, -1)
        dist[node] = 0
        queue = [node]
        while queue:
            v = queue.pop(0)
            stack.append(v)
            for w in graph[v]:
                if dist[w] < 0:
                    queue.append(w)
                    dist[w] = dist[v] + 1
                if dist[w] == dist[v] + 1:
                    sigma[w] += sigma[v]
                    pred[w].append(v)
        delta = dict.fromkeys(graph, 0)
        while stack:
            w = stack.pop()
            for v in pred[w]:
                delta[v] += (sigma[v] / sigma[w]) * (1 + delta[w])
            if w != node:
                betweenness[w] += delta[w]
    return betweenness

BC = betweenness_centrality(G)
df1 = pd.DataFrame.from_dict(BC, orient = 'index', columns = [' Betweeness␣
 ↪Centrality'])
print(df1)
```

```
   Betweeness Centrality
0                    6.0
1                    0.0
2                    6.0
3                   30.0
4                   12.0
5                   12.0
6                   14.0
7                    0.0
8                    0.0
```

E. Edge betweeness

[ ]:
```
def edge_betweenness(graph):
    betweenness = dict.fromkeys(graph.edges(), 0.0)
    for node in graph:
        stack = []
        pred = {w: [] for w in graph}
        sigma = dict.fromkeys(graph, 0)
        sigma[node] = 1
        dist = dict.fromkeys(graph, -1)
        dist[node] = 0
        queue = [node]
        while queue:
            v = queue.pop(0)
            stack.append(v)
            for w in graph[v]:
                if dist[w] < 0:
                    queue.append(w)
                    dist[w] = dist[v] + 1
                if dist[w] == dist[v] + 1:
                    sigma[w] += sigma[v]
                    pred[w].append(v)
        delta = dict.fromkeys(graph, 0)
        while stack:
            w = stack.pop()
            for v in pred[w]:
                c = (sigma[v] / sigma[w]) * (1 + delta[w])
                betweenness[(min(v, w), max(v, w))] += c
                delta[v] += c
    return betweenness
EBC = edge_betweenness(G)
df2 = pd.DataFrame.from_dict(EBC, orient = 'index', columns = [' Edge␣
 ↪Betweeness Centrality'])
print(df2)
```

```
        Edge Betweeness Centrality
(0, 1)                         8.0
(0, 2)                         2.0
(0, 3)                        18.0
(1, 2)                         8.0
(2, 3)                        18.0
(3, 4)                        20.0
(3, 5)                        20.0
(4, 5)                         2.0
(4, 6)                        12.0
(4, 7)                         6.0
(5, 6)                        12.0
(5, 7)                         6.0
(6, 7)                         4.0
```

```
(6, 8)                          16.0
```

F. Closeness centrality

[ ]:
```
def closeness_centrality(graph, node):
    dist_sum = 0
    visited = set()
    queue = [(node, 0)]
    while queue:
        v, d = queue.pop(0)
        if v not in visited:
            visited.add(v)
            dist_sum += d
            for w in graph[v]:
                if w not in visited:
                    queue.append((w, d+1))
    n = len(graph) - 1
    if dist_sum == 0:
        return 0
    else:
        return n / dist_sum

print("node\tCloseness Centrality")
for i in G:
  print(i,"\t", round(closeness_centrality(G,i), 3))
```

```
node    Closeness Centrality
0        0.471
1        0.348
2        0.471
3        0.615
4        0.615
5        0.615
6        0.5
7        0.471
8        0.348
```

G. Eigen Vector Centrality

[ ]:
```
G1 = adj_mat.reshape((9,9))
eigenvalues, eigenvectors = np.linalg.eig(G1)
max_index = np.argmax(np.abs(eigenvalues))
eigenvector = eigenvectors[:, max_index]
eigenvector = eigenvector / np.sum(eigenvector)
eigen_vector_centrality = {i+1: round(eigenvector[i], 3) for i in␣
 ↪range(len(eigenvector))}

df3 = pd.DataFrame.from_dict(eigen_vector_centrality, orient = 'index', columns␣
 ↪= ['Eigen Vector Centrality'])
```

```
print(df3)
```

```
   Eigen Vector Centrality
1                    0.072
2                    0.041
3                    0.072
4                    0.139
5                    0.172
6                    0.172
7                    0.150
8                    0.141
9                    0.043
```

2 Measures with NetworkX

A. Degree Centrality

[ ]:
```
DC = nx.degree_centrality(G)
df = pd.DataFrame.from_dict(DC, orient='index', columns=[' Degree Centrality'])
print(df)
```

```
    Degree Centrality
0               0.375
1               0.250
2               0.375
3               0.500
4               0.500
5               0.500
6               0.500
7               0.375
8               0.125
```

B. Average Degree

[ ]:
```
avg_deg = sum(dict(G.degree()).values())/len(G)
print(round(avg_deg, 3))
```

```
3.111
```

C. Betweeness centrality

[ ]:
```
bet_cen = nx.betweenness_centrality(G)

df1 = pd.DataFrame.from_dict(bet_cen, orient = 'index', columns = [' Betweeness␣
 ↪Centrality'])
print(round(df1, 3))
```

```
    Betweeness Centrality
0                   0.107
```

```
1                   0.000
2                   0.107
3                   0.536
4                   0.214
5                   0.214
6                   0.250
7                   0.000
8                   0.000
```

D. Edge Betweenness

[ ]:
```
edge_bet_cen = nx.edge_betweenness_centrality(G)

df2 = pd.DataFrame.from_dict(edge_bet_cen, orient = 'index', columns = ['␣
 ↪Betweeness Centrality'])
print(round(df2, 3))
```

```
         Betweeness Centrality
(0, 1)                   0.111
(0, 2)                   0.028
(0, 3)                   0.250
(1, 2)                   0.111
(2, 3)                   0.250
(3, 4)                   0.278
(3, 5)                   0.278
(4, 5)                   0.028
(4, 6)                   0.167
(4, 7)                   0.083
(5, 6)                   0.167
(5, 7)                   0.083
(6, 7)                   0.056
(6, 8)                   0.222
```

E. Closeness Centrality

[ ]:
```
CC = nx.closeness_centrality(G)

df3 = pd.DataFrame.from_dict(CC, orient = 'index', columns = [' Closeness␣
 ↪Centrality'])
print(round(df3,3))
```

```
    Closeness Centrality
0                  0.471
1                  0.348
2                  0.471
3                  0.615
4                  0.615
5                  0.615
6                  0.500
```

```
7                   0.471
8                   0.348
```

F. Eigen Vector Centrality

[ ]:
```
EC = nx.eigenvector_centrality(G)

df4 = pd.DataFrame.from_dict(EC, orient = 'index', columns = [' Eigen Vector␣
 ↪Centrality'])
print(round(df4,3))
```

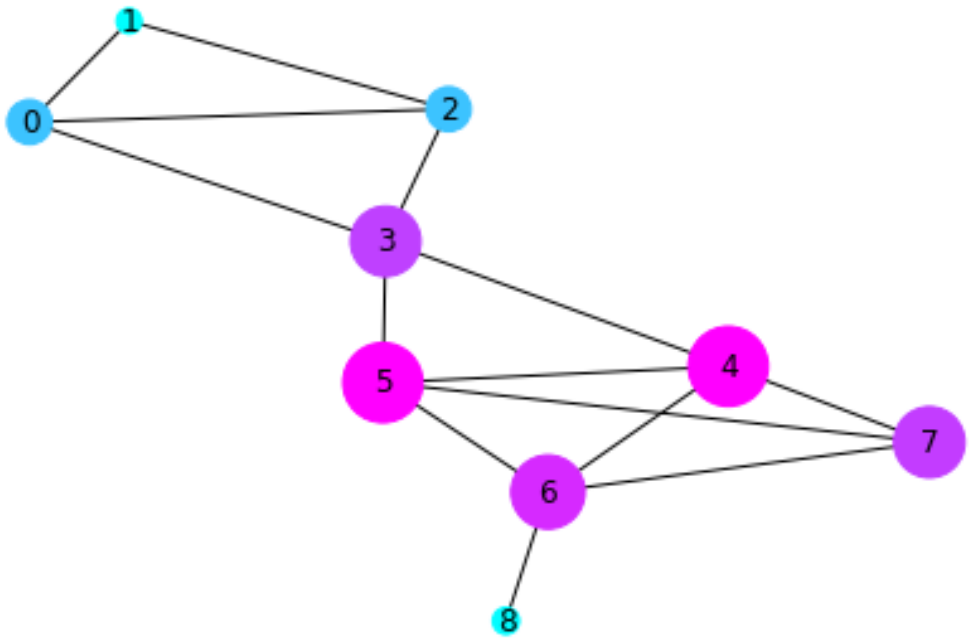
```
   Eigen Vector Centrality
0                    0.196
1                    0.112
2                    0.196
3                    0.379
4                    0.468
5                    0.468
6                    0.410
7                    0.384
8                    0.117
```

3 Function for graph plot and eigen vector

[ ]:
```
def plot_eigen_graph(adjacency_matrix):
    G = nx.from_numpy_array(adjacency_matrix)

    EC = nx.eigenvector_centrality(G)
    dframe = pd.DataFrame.from_dict(EC, orient = 'index', columns = [' Eigen␣
 ↪Vector Centrality'])
    print(round(dframe,3))

    min_ec = min(EC.values())
    max_ec = max(EC.values())
    scaled_ec = {k: ((v - min_ec) / (max_ec - min_ec)) * 0.9 + 0.1 for k, v in␣
 ↪EC.items()}

    node_size = [scaled_ec[node] * 1000 for node in G.nodes()]
    node_color = [scaled_ec[node] for node in G.nodes()]
    cmap = plt.cm.get_cmap('cool')
    nx.draw(G, node_size = node_size, node_color = node_color, cmap = cmap,␣
 ↪with_labels = True)

    plt.show()

adj_mat = np.array([[0,1,1,1,0,0,0,0,0],
                    [1,0,1,0,0,0,0,0,0],
```

```
                      [1,1,0,1,0,0,0,0,0],
                      [1,0,1,0,1,1,0,0,0],
                      [0,0,0,1,0,1,1,1,0],
                      [0,0,0,1,1,0,1,1,0],
                      [0,0,0,0,1,1,0,1,1],
                      [0,0,0,0,1,1,1,0,0],
                      [0,0,0,0,0,0,1,0,0]])
plot_eigen_graph(adj_mat)
```

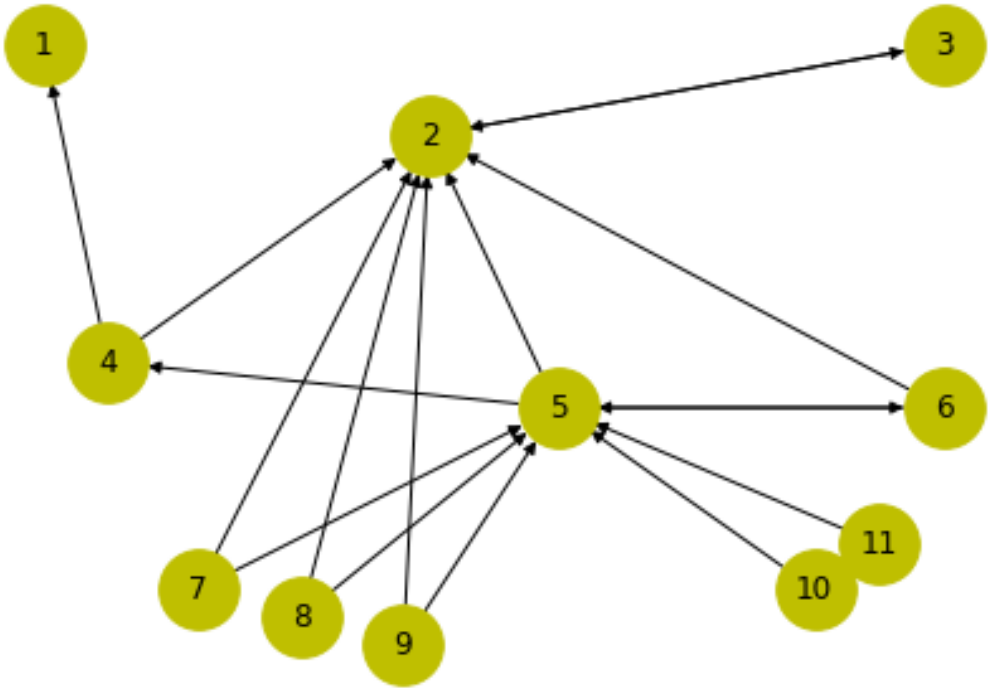
	Eigen Vector Centrality
0	0.196
1	0.112
2	0.196
3	0.379
4	0.468
5	0.468
6	0.410
7	0.384
8	0.117



4 Katz Centrality

[3]:
```
DiG = nx.DiGraph()
DiG.add_edges_from([(2, 3), (3, 2), (4, 1), (4, 2), (5, 2), (5, 4),
                    (5, 6), (6, 2), (6, 5), (7, 2), (7, 5), (8, 2),
                    (8, 5), (9, 2), (9, 5), (10, 5), (11, 5)])
dpos = {1: [0.1, 0.9], 2: [0.4, 0.8], 3: [0.8, 0.9], 4: [0.15, 0.55],
        5: [0.5,  0.5], 6: [0.8,  0.5], 7: [0.22, 0.3], 8: [0.30, 0.27],
        9: [0.38, 0.24], 10: [0.7,  0.3], 11: [0.75, 0.35]}
nx.draw(DiG, dpos, with_labels = 'True', node_color = 'y', node_size = 1000)

KC = nx.katz_centrality(DiG)
dframe = pd.DataFrame.from_dict(KC, orient = 'index', columns = [' Katz␣
 ↪Centrality'])
print(round(dframe, 3))
```

```
    Katz Centrality
2             0.450
3             0.293
4             0.288
1             0.277
5             0.401
6             0.288
7             0.248
8             0.248
9             0.248
10            0.248
11            0.248
```


1
3
2
4
5
6
11
7
10
8
9