Data Mining and Web algorithm

Lab Assignment 8:

[16-21May, 2022]

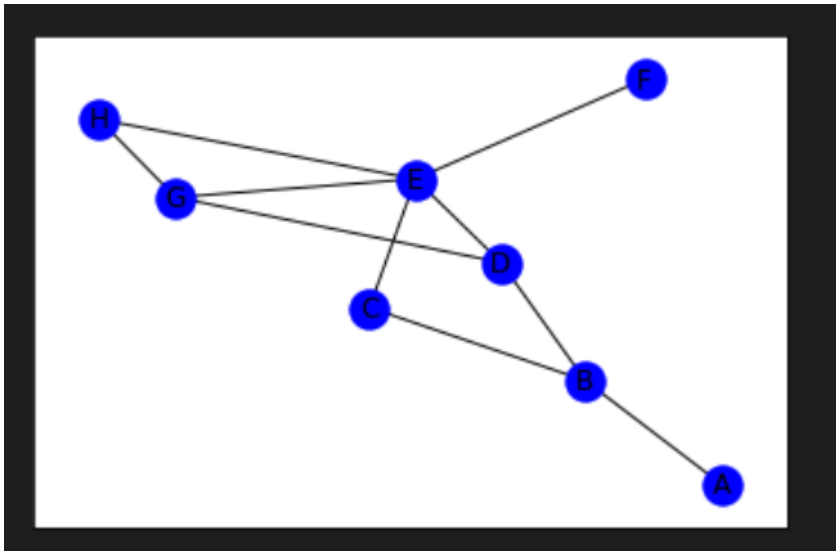
Patil Amit Gurusidhappa
19104004
B11

Answer the following questions about this graph.

```
import networkx as nx
```

```
# Create Graph
g = nx.Graph()
g.add_nodes_from(['A','B','C','D','E','F','G','H'])
g.add_edges_from([('A',
'B'),('B','C'),('B','D'),('C','E'),('D','G'),('D','E'),('E','F'),('H','G')
,('E','G'),('E','H')])
nx.draw_networkx(g, node_color ='blue')
```

a) Draw the given graph G, using networkx library



b) Count the number of nodes & edges in the network?

```
total_node=len(g2.nodes)
given_edges=len(g2.edges)
```

```
print("Nodes:",total_node," Edges:",given_edges)
```

```
Nodes: 4  Edges: 6
```

c) Create an adjacency list for this graph.

```
adj_list={}
for node in g2.nodes():
    adj_list[node]=[]
    for nei in g2.neighbors(node):
        adj_list[node].append(nei)
print(adj_list)
```

```
{'Page A': ['Page B'], 'Page B': ['Page A', 'Page C'], 'Page C':
['Page A', 'Page D'], 'Page D': ['Page A']}
```

d) Create an adjacency matrix for this graph.

```
print (nx.adjacency_matrix(g2))
```

```
  (0, 1)        1
  (1, 0)        1
  (1, 2)        1
  (2, 0)        1
  (2, 3)        1
  (3, 0)        1
```

e) Compute density of a graph.

```
# Density
total_node=len(g.nodes)
total_edges=total_node*(total_node-1)
print(len(g.edges)/total_edges)
```

```
0.12222222222222222
```

f) Compute the degree of each node

```
# Degree
for node in g.nodes():
    print(node, g.degree(node))
```

```
A 1
B 3
C 2
D 3
E 5
F 1
```

g) Remove the node with the lowest degree.

```
# Remove
g_copy=g.copy()
```

```
g_copy.remove_node('A')
print(g_copy.nodes())
```

2. Consider the 4 web graphs and Implement the Lary and Brin PageRank algorithm with damping as a function w = PageRank(B,d) that takes as input a damping factor, d=0.7, and returns the PageRank of each page till the solution converges.

```
import numpy as np
import numpy.linalg as la
```

```
def pageRank(linkMatrix, d) :
    n = linkMatrix.shape[0]
    # M = d * linkMatrix + (1-d)/n * np.ones([n, n])
    M = d * linkMatrix
    r = 100 * np.ones(n) / n # Sets up this vector (n entries of 1/n × 100
each)
    last = r
    r = M @ r
    for i in range(18):
        last = r
        r = M @ r
    return r
```

```
L = np.array([[0,   1/2, 1/3, 0, 0,   0 ],
              [1/3, 0,   0,   0, 1/2, 0 ],
              [1/3, 1/2, 0,   1, 0,   1/2 ],
              [1/3, 0,   1/3, 0, 1/2, 1/2 ],
              [0,   0,   0,   0, 0,   0 ],
              [0,   0,   1/3, 0, 0,   0 ]])
```

```
pageRank(L, 1)
```

```
array([16.00149917,  5.33252025, 39.99916911, 25.3324738 ,  0.
,
       13.33433767])
```

3. Implement HITS Algorithm and find out the authority score and hub scores of all pages in the
below given graphs. Repeat the iterations until the solution converges.

```
import networkx as nx
```

```
def hits(graph, iterations, tolarance=1.0e-8):
    hubs = dict.fromkeys(graph, 1.0 / graph.number_of_nodes())   #{'A': 0.25, 'B': 0.25, 'C': 0.25, 'D': 0.25}
    authorities = {}
    # power iteration, which stops after given iterations or reaching tolerance
    for _ in range(iterations):
        last_hubs = hubs
        hubs = dict.fromkeys(last_hubs.keys(), 0) # {'A': 0, 'B': 0, 'C': 0, 'D': 0}
        authorities = dict.fromkeys(last_hubs.keys(), 0) # {'A': 0, 'B': 0, 'C': 0, 'D': 0}
        for node in hubs:
            for neighbor in graph[node]:
                authorities[neighbor] += last_hubs[node] * graph[node][neighbor].get('weight', 1)
        for node in hubs:
            for neighbor in graph[node]:
                hubs[node] += authorities[neighbor] * graph[node][neighbor].get('weight', 1)
        scaling = 1.0 / max(hubs.values())
        for node in hubs:
            hubs[node] *= scaling
        scaling = 1.0 / max(authorities.values())
        for node in authorities:
            authorities[node] *= scaling
        err = sum([abs(hubs[node] - last_hubs[node]) for node in hubs])
        if err < tolarance:
            break
    return hubs, authorities


G1 = nx.DiGraph()
```

```
G1.add_edges_from([('A', 'B'),
                   ('A', 'C'),
                   ('D', 'C'),
                   ('D', 'C'),
                   ('C', 'A'),
                   ('B', 'C')
                   ])

print(hits(G1, 5))
# ({'A': 1.0, 'B': 0.7071428571428572, 'C': 0.0017857142857142852, 'D':
0.7071428571428572}, {'A': 0.0025252525252525246, 'B': 0.4141414141414141,
'C': 1.0, 'D': 0.0})
```

```
({'A': 1.0, 'B': 0.7071428571428572, 'C': 0.0017857142857142852,
'D': 0.7071428571428572}, {'A': 0.0025252525252525246, 'B':
0.4141414141414141, 'C': 1.0, 'D': 0.0})
```

```
hubs, authorities = nx.hits(G1, max_iter = 5, normalized = True)
print(hubs)
print(authorities)
```

```
{'A': 0.4142135623730951, 'B': 0.2928932188134525, 'C':
-5.7145246952375 84e-17, 'D': 0.2928932188134525}
{'A': -1.3796083021758554e-16, 'B': 0.2928932188134525, 'C':
0.7071067811865476, 'D': 0.0}
```