Tutorial 7

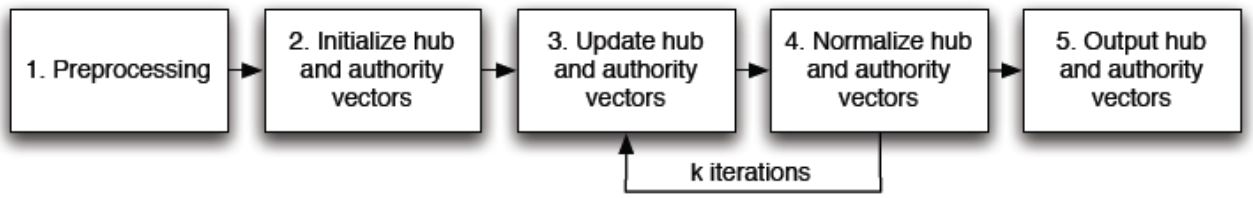
Nur Farahin Binti Al Latiff (17172226/1)

1) What is the difference between Hits algorithm and PageRank algorithm?

	Hits algorithm	PageRank algorithm
Scoring Criteria	Good authorities are pointed to by good hubs and good hubs point to good authorities.	A webpage is important if it is pointed to by other important pages.
Number of scores Dual Rankings	a) one with the most authoritative documents related to the query b) other with the most "hubby" documents.	Page rank only presents one score.
Query independence	HITS score is calculated after getting the neighbourhood graph according to the query	PageRank score is query independent

2) Where and how HITS is used?
HITS is used to the web link-structures to discover and rank the webpages relevant for a particular search

How HITS used:



3) What are the two core concepts in the HITS algorithm?
Authority value and hub value

4) What are HUBs in the HITS algorithm? What purpose do they serve?

Hubs are pages which are not very relevant but point to a page in the Root. Hub purpose to estimate the value links to other pages.

5) What are Authorities in the HITS algorithm? What purpose do they serve?

- Authority value is computed as the sum of the scaled hub values that point to that page.
- Authority values will estimate the value of the content of the page.

6) A web-page containing links to Top 1000 Universities in the world — is this a HUB or Authority in HITS?

- Authority

7) Replicate the source(https://www.youtube.com/watch?v=2rvN6tXdUkE&t=1520s&pbjreload=101) in to your own Jupyter Notebook. Run each link prediction algorithms that was shown in this source using the dataset(dolphin_edges.csv)

1. Load the network

In [2]:
```
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
A = pd.read_csv('dolphin_edges.csv')
```

In [4]:
```
A
```

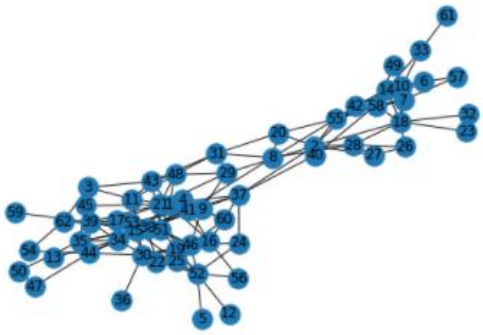
Out[4]:

	From	To
0	4	9
1	6	10
2	7	10
3	1	11
4	3	11
...	...	...
154	46	60
155	33	61
156	3	62
157	38	62
158	54	62

159 rows × 2 columns

In [6]:
```
G = nx.from_pandas_edgelist(A, source='From', target='To')
```

In [7]:
```
nx.draw(G, with_labels=True)
```



2. Common Neighbors

In [8]:
```
# Targets are pair of nodes that are not connected directly.
# Links to be predicted
targets = nx.non_edges(G)
common_neighbors = [(e[0], e[1],
                     len(list(nx.common_neighbors(G, e[0], e[1]))))
                    for e in targets]
sorted(common_neighbors, key=lambda x: x[2], reverse=True)
```

Out[8]:
```
[(6, 7, 4),
 (7, 42, 4),
 (10, 55, 4),
 (16, 52, 4),
 (17, 44, 4),
 (18, 42, 4),
 (18, 55, 4),
 (22, 25, 4),
 (38, 39, 4),
 (38, 51, 4),
 (39, 51, 4),
 (2, 8, 3),
 (2, 14, 3),
 (2, 26, 3),
 (2, 58, 3),
 (6, 18, 3),
 (6, 42, 3),
 (9, 37, 3),
 (15, 16, 3),
 (15, 21, 3),
```

3. Jaccard Coefficients

In [9]:
```
jaccard_coefficient = list(nx.jaccard_coefficient(G))
sorted(jaccard_coefficient, key=lambda x: x[2], reverse=True)
```

Out[9]:
```
[(5, 12, 1.0),
 (23, 32, 1.0),
 (6, 7, 0.6666666666666666),
 (7, 42, 0.5714285714285714),
 (5, 56, 0.5),
 (6, 42, 0.5),
 (12, 56, 0.5),
 (22, 25, 0.5),
 (40, 49, 0.5),
 (17, 44, 0.4444444444444444),
 (6, 33, 0.4),
 (10, 55, 0.4),
 (18, 42, 0.4),
 (2, 26, 0.375),
 (17, 35, 0.375),
 (44, 53, 0.375),
 (39, 51, 0.36363636363636365),
 (5, 24, 0.3333333333333333),
 (12, 24, 0.3333333333333333),
 (18, 27, 0.3333333333333333),
```

4. Resource Allocation

In [10]:
```
resource_allocation_index = list(nx.resource_allocation_index(G))
sorted(resource_allocation_index, key=lambda x: x[2], reverse=True)
```

Out[10]:
```
[(16, 52, 0.9004329004329005),
 (6, 7, 0.878968253968254),
 (18, 27, 0.6583333333333333),
 (2, 26, 0.6444444444444444),
 (37, 46, 0.6242424242424243),
 (10, 55, 0.6027777777777777),
 (2, 8, 0.5928571428571429),
 (44, 62, 0.5909090909090909),
 (15, 30, 0.5595238095238095),
 (18, 55, 0.5277777777777777),
 (7, 42, 0.5218253968253967),
 (18, 42, 0.503968253968254),
 (35, 47, 0.5),
 (37, 58, 0.5),
 (44, 50, 0.5),
 (38, 39, 0.4928571428571428),
 (21, 38, 0.47619047619047616),
 (38, 54, 0.47619047619047616),
 (39, 51, 0.46111111111111114),
 (15, 16, 0.4583333333333333),
```

5. Adamic-Adar Index

```
In [11]: adamic_adar_index = list(nx.adamic_adar_index(G))
         sorted(adamic_adar_index, key=lambda x: x[2], reverse=True)
```

```
Out[11]: [(16, 52, 2.9317364012342075),
          (6, 7, 2.8926113435351204),
          (10, 55, 2.1154635213872655),
          (18, 27, 2.012472508149437),
          (2, 26, 1.986693774499868),
          (18, 55, 1.9750269337906414),
          (7, 42, 1.963814645015908),
          (37, 46, 1.9486065526106957),
          (18, 42, 1.930814649609145),
          (38, 39, 1.9087330552060944),
          (44, 62, 1.8597274323132096),
          (2, 8, 1.8565807973738442),
          (39, 51, 1.8499543261497624),
          (22, 25, 1.8203448290106674),
          (38, 51, 1.8118671042605898),
          (15, 30, 1.7933564893654796),
          (17, 44, 1.7346548246723308),
          (21, 38, 1.6301195954722452),
          (15, 16, 1.5971196000654824),
          (2, 14, 1.5903528902427813),
```