INFORMATION RETRIEVAL ASSIGNMENT - 3

Group No. 78
Nellore Bhavana (MT21131)
Prachi Gupta (MT21135)

Question 1 - Link Analysis

- **Represent the network in terms of its 'adjacency matrix' as well as 'edge list'**

Adjacency matrix:

```
#Taking approximate rows and columns as 8900
rows, cols = (8900, 8900)
adj = [[0]*cols]*rows
adj = [[0 for i in range(8900)] for i in range(8900)]
#Form adjacency matrix
for ed in edges:
  adj[ed[0]][ed[1]] = 1
#Network in terms of its 'adjacency matrix'
adj
```

```
[[0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
```

Edge List:

```
#Network in terms of 'edge list'
edges
```

```
[[30, 1412],
 [30, 3352],
 [30, 5254],
 [30, 5543],
 [30, 7478],
 [3, 28],
 [3, 30],
 [3, 39],
 [3, 54],
 [3, 108],
 [3, 152],
 [3, 178],
 [3, 182],
 [3, 214],
 [3, 271],
 [3, 286],
```

- **Briefly describe the dataset chosen and report the following:**
  Dataset Used:
  The dataset used can be found in the following link [1],
  https://snap.stanford.edu/data/wiki-Vote.html

Wikipedia is a free encyclopedia written collaboratively by volunteers around the world. A small part of Wikipedia contributors are administrators, who are users with access to additional technical features that aid in maintenance. In order for a user to become an administrator, a Request for adminship (RfA) is issued and the Wikipedia community via a public discussion or a vote decides who to promote to adminship. Using the latest complete dump of Wikipedia page edit history (from January 3, 2008) we extracted all administrator elections and voting history data. This gave us 2,794 elections with 103,663 total votes and 7,066 users participating in the elections (either casting a vote or being voted on). Out of these 1,235 elections resulted in a successful promotion, while 1,559 elections did not result in the promotion. About half of the votes in the dataset are from existing admins, while the other half come from ordinary Wikipedia users.

The network contains all the Wikipedia voting data from its inception of Wikipedia till January 2008. Nodes in the network represent Wikipedia users and a directed edge from node *i* to node *j* represents that user *i* voted on user *j*.

Dataset statistics	
Nodes	7115
Edges	103689
Nodes in largest WCC	7066 (0.993)
Edges in largest WCC	103663 (1.000)
Nodes in largest SCC	1300 (0.183)
Edges in largest SCC	39456 (0.381)
Average clustering coefficient	0.1409
Number of triangles	608389
Fraction of closed triangles	0.04564
Diameter (longest shortest path)	7
90-percentile effective diameter	3.8

1. **Number of Nodes**
2. **Number of Edges**

```
[5]  #Number of nodes
     print(len(nodes))
     #Number of edges
     print(len(edges))
```

```
7115
103689
```

3. **Avg In-degree**

```
[8]  #taking indegree as a dictionary
     indegree = {}
     #Initializing indegree for every node as 0
     for nd in nodes:
       indegree[nd] = 0
     for ed in edges:
       indegree[ed[1]] = indegree[ed[1]] + 1
     #average in-degree = divide the summation of all nodes' degree by the total number of nodes
     sum = 0
     for nd in nodes:
       sum = sum + indegree[nd]
     nl = len(nodes)
     el = len(edges)
     print(f"Avg In-degree: {sum/nl}")
```

```
Avg In-degree: 14.573295853829936
```

4. **Avg. Out-Degree**

```
[9]  #taking outdegree as a dictionary
     outdegree = {}
     #Initializing outdegree for every node as 0
     for nd in nodes:
       outdegree[nd] = 0
     for ed in edges:
       outdegree[ed[0]] = outdegree[ed[0]] + 1
     #average out-degree = divide the summation of all nodes' degree by the total number of nodes
     sum = 0
     for nd in nodes:
       sum = sum + outdegree[nd]
     nl = len(nodes)
     print(f"Avg Out-degree: {sum/nl}")
```

```
Avg Out-degree: 14.573295853829936
```

5. Node with Max In-degree

```
[10] #Initialize maximum indegree and node with maximum indegree with 0
     max_indegree = 0
     nodes_max = 0
     for nd in nodes:
       if(indegree[nd] > max_indegree):
         nodes_max = nd
         max_indegree = indegree[nd]
       else:
         nodes_max = nodes_max
     print(f"Node {nodes_max} with {max_indegree} maximum indegree")
```

```
Node 4037 with 457 maximum indegree
```

6. Node with Max out-degree

```
[11] #Initialize maximum outdegree and node with maximum outdegree with 0
     max_outdegree = 0
     nodes_maxo = 0
     for nd in nodes:
       if(outdegree[nd] > max_outdegree):
         nodes_maxo = nd
         max_outdegree = outdegree[nd]
       else:
         nodes_maxo = nodes_maxo
     print(f"Node {nodes_maxo} with {max_outdegree} maximum outdegree")
```

```
Node 2565 with 893 maximum outdegree
```

7. The density of the network:

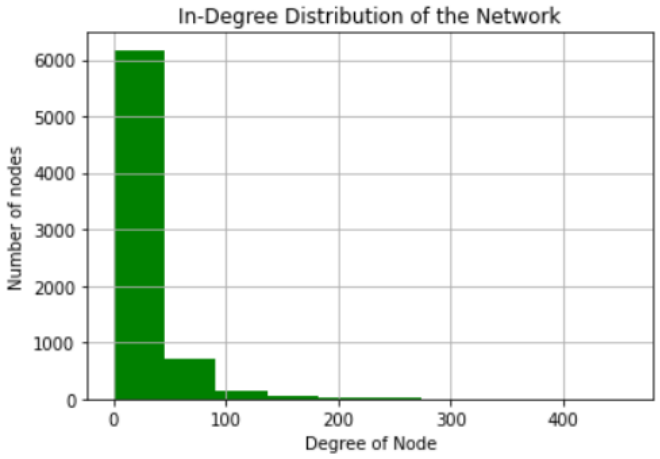
```
nl = len(nodes)
el = len(edges)
#Density = The ratio of observed edges to the number of possible edges for a given network
density = el/(nl*(nl-1))
print(f"The density of the network: {density}")
```

```
The density of the network: 0.0020485375110809584
```

- **Plot degree distribution of the network**

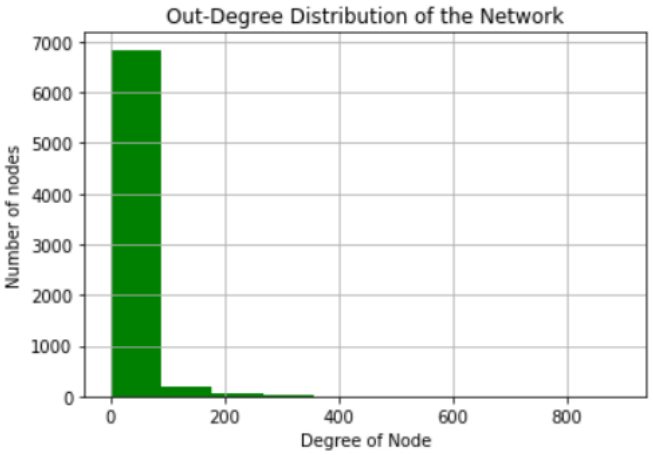
1. **In-degree:**

```
[13] #Store unique indegree values for plotting
     distinct_ind = list(indegree.values())
     plt.hist(distinct_ind, color = "green")
     plt.xlabel("Degree of Node")
     plt.ylabel("Number of nodes")
     plt.title("In-Degree Distribution of the Network")
     plt.grid()
     plt.show()
```



2. **Out-degree:**

```
#Store unique outdegree values for plotting
distinct_out = list(outdegree.values())
plt.hist(distinct_out, color = "green")
plt.xlabel("Degree of Node")
plt.ylabel("Number of nodes")
plt.title("Out-Degree Distribution of the Network")
plt.grid()
plt.show()
```



Question 2 - PageRank, Hubs, and Authority

```
[28] #Create an empty graph structure (a “null graph”) with no nodes and no edges
     G = nx.DiGraph()
     #Add nodes and edges from container "nodes" and "edges"
     G.add_nodes_from(nodes)
     G.add_edges_from(edges)
```

```
[29] #Checking number of nodes and edges in the Graph
     print(G.number_of_nodes())
     print(G.number_of_edges())
```

```
7115
103689
```

- **PageRank score for each node**

```
#Calculating page rank score for each node
page_rank = nx.pagerank(G)
page_rank
```

```
{3: 0.00020539498232448016,
 4: 5.048782345863015e-05,
 5: 5.048782345863015e-05,
 6: 0.00031183250978437455,
 7: 5.048782345863015e-05,
 8: 0.00032663557615950425,
 9: 5.048782345863015e-05,
 10: 0.0004213996615598798,
 11: 5.048782345863015e-05,
 12: 5.048782345863015e-05,
 13: 5.048782345863015e-05,
 14: 5.048782345863015e-05,
 15: 0.00368122072952927,
 16: 5.048782345863015e-05,
 17: 5.048782345863015e-05,
 18: 5.048782345863015e-05,
 19: 0.00013112179292607272,
 20: 5.048782345863015e-05,
```

Sorting the values of page rank score

```
#Sort the page rank score for each node
page_rank_sorted = dict(sorted(page_rank.items(), key=lambda x: x[1], reverse = True))
page_rank_sorted
```

```
{4037: 0.004612715891167541,
 15: 0.00368122072952927,
 6634: 0.003524813657640256,
 2625: 0.003286374369230901,
 2398: 0.0026053331717250175,
 2470: 0.002530105328384948,
 2237: 0.0025047038004839886,
 4191: 0.0022662633042363433,
 7553: 0.0021701850491959575,
 5254: 0.0021500675059293213,
 1186: 0.002043893687602912,
 2328: 0.002041628886088916,
 1297: 0.0019518608216122287,
 4335: 0.001935301447578486,
 7620: 0.001930119395754876,
 5412: 0.0019167080775239892,
 7632: 0.0019037739909136607,
 4875: 0.0018675748225119072,
 3352: 0.0017851250122027202,
 2654: 0.0017693207143482403,
 6832: 0.0017646895191923734,
```

- **Authority and Hub score for each node**

Hub Score:

```
#Authority and Hub score for each node
Hub, Authority = nx.hits(G)
Hub
```

```
{3: 4.0210316397776376e-05,
 4: 7.319607685824178e-05,
 5: 3.501788474433638e-05,
 6: 0.0010539872861763616,
 7: 8.200618013274943e-05,
 8: 0.0003200123333087148,
 9: 0.00023181963355171437,
 10: 0.00018207047608178445,
 11: 0.004921182063808104,
 12: 0.0002882286245176331,
 13: 3.1200539556819314e-05,
 14: 0.0004975871202612586,
 15: 0.0004157326033809383,
 16: 8.409812451935359e-05,
 17: 0.00010084393686112596,
 18: 3.617008656019803e-05,
 19: 7.217473720148202e-05,
 20: 0.0008868292175658342,
 21: 0.00029286346222089055,
```

Sorting Hub score,

```
#Hub values in sorted order
hub_sorted = dict(sorted(Hub.items(), key=lambda x: x[1], reverse = True))
hub_sorted
```

```
{2565: 0.00794049270814314,
 766: 0.007574335297501249,
 2688: 0.006440248991029863,
 457: 0.006416870490261071,
 1166: 0.006010567902411201,
 1549: 0.005720754058269245,
 11: 0.004921182063808104,
 1151: 0.0045720407017564085,
 1374: 0.004467888792711107,
 1133: 0.003918881732057349,
 2485: 0.0037844608130803746,
 2972: 0.003517673976814718,
 3449: 0.003503558110460446,
 3453: 0.003449414861112209,
 4967: 0.0034433407418341284,
 3352: 0.003381423106344999,
 2871: 0.0032390167017277093,
```

Authority score:

```
#Values of authority
Authority
```

```
{3: 9.50117185846073e-05,
 4: -0.0,
 5: -0.0,
 6: 6.398065594290155e-05,
 7: -0.0,
 8: 0.000187766801949214,
 9: -0.0,
 10: 6.848818550186205e-05,
 11: -0.0,
 12: -0.0,
 13: -0.0,
 14: -0.0,
 15: 0.002201543492565582,
 16: -0.0,
 17: -0.0,
 18: -0.0,
 19: 0.00012068234426719106,
```

Sorting authority values,

```
#Authority values in sorted order
auth_sorted = dict(sorted(Authority.items(), key=lambda x: x[1], reverse = True))
auth_sorted
```

```
{2398: 0.0025801471780088738,
 4037: 0.0025732411242297927,
 3352: 0.002328415091497684,
 1549: 0.0023037314804571795,
 762: 0.0022558748562871394,
 3089: 0.0022534066884511636,
 1297: 0.002250144636662723,
 2565: 0.00222356410395361 3,
 15: 0.00220154349256558 2,
 2625: 0.002197896803403074,
 2328: 0.0021723715453407406,
 2066: 0.0021070409396099755,
 4191: 0.0020811941305289875,
 3456: 0.002050435521510778,
```

- **Compare the results obtained from both the algorithms in parts 1 and 2 based on the node scores**

- The importance and relevance of web page can be understood using the page rank score and authority score based on the incoming links.
- On cross verifying we observed that different nodes are marked relevant based on both the score metrics.
- The reason for this variance is that page rank score takes node with maximum in-degree (node 4037 as calculated in above parts) while in authority score, the higher values are for those nodes that have incoming links from maximum number of hub nodes that may not contain all the in-degree linked nodes from the graphical network.
- For Hub score, we referred the node based on the outgoing edge. So, the node with maximum number of outgoing links/ out-degree that is calculated as 2565 node in the above part.

Results Comparison for PageRank and HITS –

	Page Rank	HITS
By	Larry Page and Sergey Brin	Jon Kleinberg in 1998
Year	1998	1998
Working	PageRank computes a ranking of nodes in the graph based on the structure of the incoming links.	HITS algorithm computes the authority score for a node based on the incoming links and computes the hub score based on outgoing links.
Calculating Scores	As PageRank calculates scores based on incoming links, higher the incoming links from the good nodes.	As hub score is based on outgoing links, the higher the scores means it points to many good nodes.
Usage Plan	PageRank is used for ranking all the nodes of the complete graph and then applying a search.	HITS is applied on a subgraph after a search is done on the complete graph.
Structure Idea	PageRank is based on the 'random surfer' idea and the web is seen as a Markov Chain.	HITS defines hubs and authorities recursively.
Known Issues	PR is distributed from a node to their outlinks, therefore there is an issue when the "surfer" reaches the dangling nodes in a graph.	Links to a large number of separate topics may receive a high hub rank which is not relevant to the given query.
Historical Day Uses	Rank web pages in google's search engine results.	HITS has been used to analyse co-citation and co-reference in the field of citation analysis and bibliometrics.
Modern Day Uses	VisualRank – Google's application of PageRank to image-search	In recent years, researchers have been using the HITS algorithm to solve problems such as spam detection.

Fig. A Review of PageRank and HITS Algorithms [2]

References:

[1] https://snap.stanford.edu/data/wiki-Vote.html

[2] Patel, Punit & Patel, Kanu. (2015). A Review of PageRank and HITS Algorithms. International Journal of Advance Research in Engineering, Science & Technology. 2. 2394-2444.