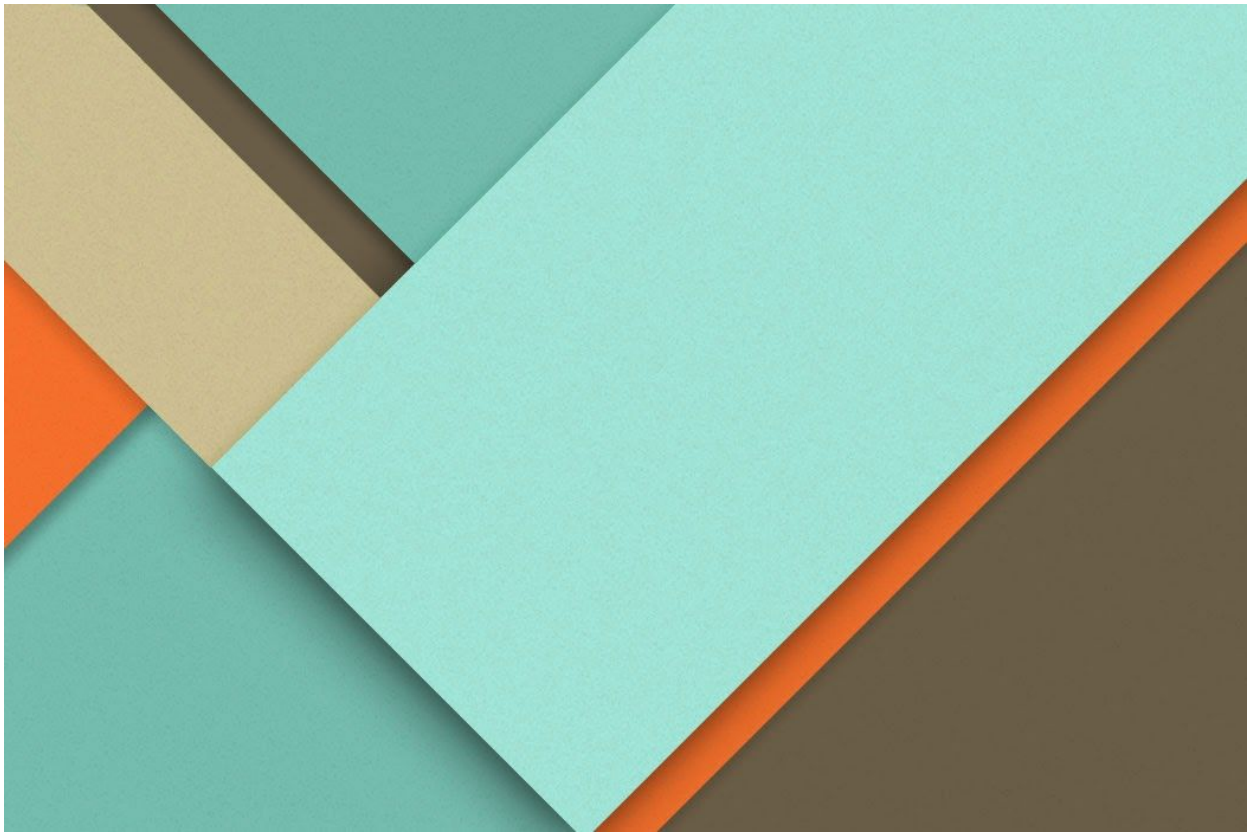


SNA Project Round 1

14.03.2021

Team FALCONS:

Mehak Singhal (18ucc112)

Raghav Lakhotia (18ucs058)

Deepanshu Garg (18ucs068)

Supervisor:

Dr. Sakti Balan

INDEX

Overview

The Project has two datasets (Facebook social media and wikipedia vote network), implementation of different centralities along with clustering coefficient, reciprocity, and transitivity has been made. Code is written in python using a network library for manipulation graph dataset.

Goals

1. Learning about the Network package of python.
2. Implementing different centralities for the actual dataset.

Dataset Specifications

Following are two Dataset; one is the social circle of Facebook, which is an undirected graph because in Facebook if we accept the friendship, it is a two-way network. Another Dataset is the Wikipedia Vote network. It is a Directed type of graph because it's not necessary if a webpage has a backlink to another website, vice versa may not be true.

1. Social Circle : Facebook (UNDIRECTED EDGES)

This dataset consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using Facebook App. The dataset includes node features (profiles), circles, and ego networks.

Facebook data has been anonymized by replacing the Facebook-internal ids for each user with a new value. Also, while feature vectors from this dataset have been provided, the interpretation of those features has been obscured.

Network Specification:

Dataset statistics	
Nodes	4039
Edges	88234
Nodes in largest WCC	4039 (1.000)
Edges in largest WCC	88234 (1.000)
Nodes in largest SCC	4039 (1.000)
Edges in largest SCC	88234 (1.000)
Average clustering coefficient	0.6055
Number of triangles	1612010
Fraction of closed triangles	0.2647
Diameter (longest shortest path)	8
90-percentile effective diameter	4.7

Dataset Link: https://snap.stanford.edu/data/ego-Facebook.html

2. Wikipedia Vote Network (DIRECTED EDGES)

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 dataset's votes are by existing admins, while the other half comes from ordinary Wikipedia users.

Network Specification:

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

Dataset Link: https://snap.stanford.edu/data/wiki-Vote.html

Playing with Graph:

1. Importing Libraries

```
import networkx as nx
import matplotlib.pyplot as plt
```

2. Importing Dataset

```
G1 =nx.read_edgelist("social_facebook_dataset.txt", create_using = nx.Graph(), nodetype=int)
G2 =nx.read_edgelist("Wiki-Vote.txt", create_using = nx.Graph(), nodetype=int)
```

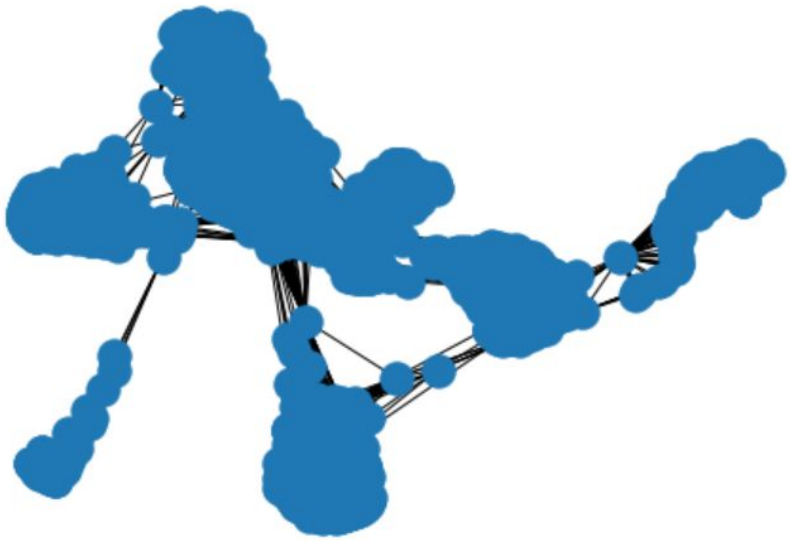
3. Graph Information

```
# Infomatio About the Graph
print(nx.info(G1))
print(nx.info(G2))
```

```
Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88234
Average degree:  43.6910
Name:
Type: Graph
Number of nodes: 7115
Number of edges: 100762
Average degree:  28.3238
```

4. Plotting Graph

```
#Plotting Graph : Social Facebook Network
nx.draw(G1)
```



```
#Plotting Graph : Wikipedia Network
nx.draw(G2)
```

Problem Statement 1:

Find all centrality measures, clustering coefficients (both local and global) and reciprocity and transitivity. We have studied in the class using appropriate algorithms (you may use specific packages for this or write your own algorithm for the same).

1. All Centrality Measure:

- **Degree Centrality**

Degree centrality is a simple count of the total number of connections linked to a vertex. Degree is the measure of the total number of edges connected to a particular vertex. For our data set, the user rating a large number of books and a book with a large number of ratings will be more central according to this metric.

There are two types of degree centrality for a directed graph:

1. In Degree : no of Incoming edges

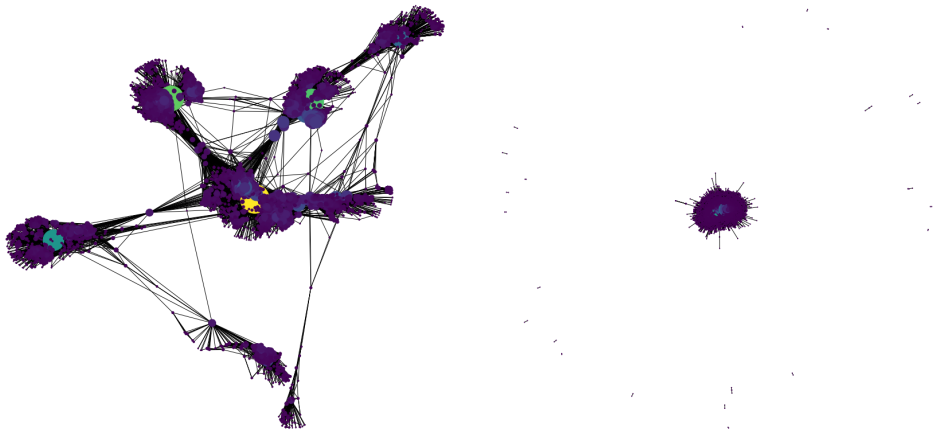
2. Out Degree : no of Outgoing edges

The degree distribution obtained for our sample was:

```python
def degreeCentrality(G):
    pos = nx.spring_layout(G)
    degCent = nx.degree_centrality(G)
    node_color = [20000.0 * G.degree(v) for v in G]
    node_size =  [v * 10000 for v in degCent.values()]
    plt.figure(figsize=(15,15))
    nx.draw_networkx(G, pos=pos, with_labels=False,
                     node_color=node_color,
                     node_size=node_size )
    plt.axis('off')
    #Printing Top 10 Nodes as per Degree Centrality
    print("Printing Top 10 Nodes as per Degree Centrality")
    sorted_degree=sorted(degCent, key=degCent.get, reverse=True)[:10]
    for d in sorted_degree[:10]:
        print("Node Label: ",d,"=>",G.degree[d])

#Facebook Network Centality
print("Facebook Network Degree Centality:\n")
degreeCentrality(G1)
# Wikipedia Network Degree Centality
print("\nWikipedia Network Degree Centality:\n")
degreeCentrality(G2)
```

```
Facebook Network Degree Centality:

Printing Top 10 Nodes as per Degree Centrality
Node Label:  107 => 1045
Node Label:  1684 => 792
Node Label:  1912 => 755
Node Label:  3437 => 547
Node Label:  0 => 347
Node Label:  2543 => 294
Node Label:  2347 => 291
Node Label:  1888 => 254
Node Label:  1800 => 245
Node Label:  1663 => 235
```

```
Wikipedia Network Degree Centality:

Printing Top 10 Nodes as per Degree Centrality
Node Label:  2565 => 1065
Node Label:  766 => 773
Node Label:  11 => 743
Node Label:  1549 => 740
Node Label:  457 => 732
Node Label:  1166 => 688
Node Label:  2688 => 618
Node Label:  1374 => 533
Node Label:  1151 => 517
Node Label:  5524 => 495
```



- **Eigenvector Centrality**

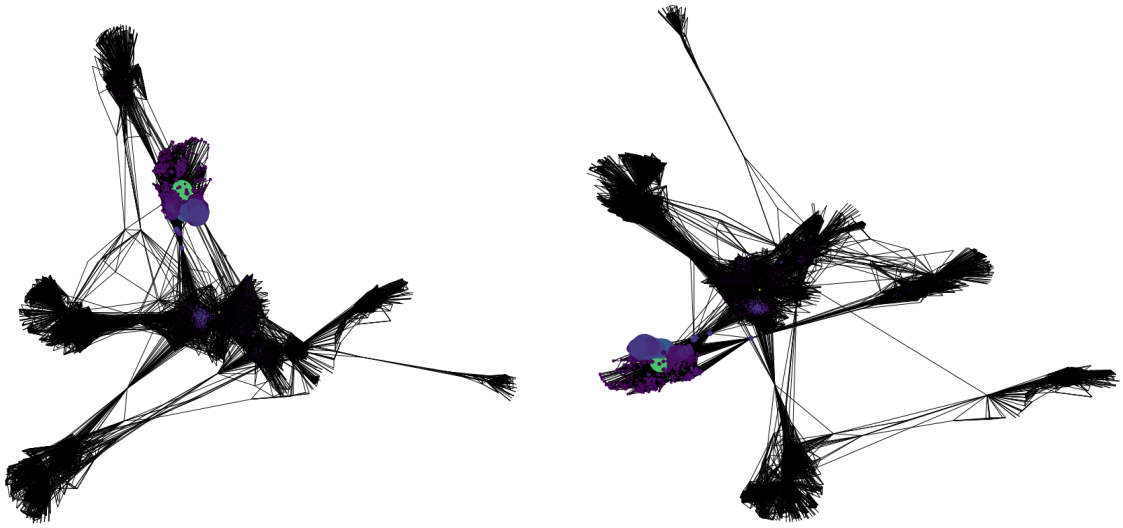
Eigenvector centrality measures a node's importance while considering the importance of its neighbors. It is sometimes used to measure a node's influence in the network.

It is determined by performing a matrix calculation to determine what is called the principal eigenvector using the adjacency matrix.

```
def eigenVectorCentrality(G):
    pos = nx.spring_layout(G1)
    eigCent = nx.eigenvector_centrality(G1)
    node_color = [20000.0 * G1.degree(v) for v in G1]
    node_size =  [v * 10000 for v in eigCent.values()]
    plt.figure(figsize=(15,15))
    nx.draw_networkx(G1, pos=pos, with_labels=False,
                     node_color=node_color,
                     node_size=node_size )
    plt.axis('off')
    print("Printing Top 10 Nodes as per Eigenvector Centrality")
    sorted_degree=sorted(eigCent, key=eigCent.get, reverse=True)[:10]
    for d in sorted_degree[:10]:
        print("Node Label: ",d,"=>",G1.degree[d])

#Facebook Network Eigenvector Centality
print("Facebook Network Eigenvector Centality:\n")
eigenVectorCentrality(G1)
# Wikipedia Network Eigenvector Centality
print("\nWikipedia Network Eigenvector Centality:\n")
eigenVectorCentrality(G2)
```

```
Facebook Network Eigenvector Centality:

Printing Top 10 Nodes as per Degree Centrality
Node Label:  107 => 1045
Node Label:  1684 => 792
Node Label:  1912 => 755
Node Label:  3437 => 547
Node Label:  0 => 347
Node Label:  2543 => 294
Node Label:  2347 => 291
Node Label:  1888 => 254
Node Label:  1800 => 245
Node Label:  1663 => 235
```

```
Wikipedia Network Eigenvector Centality:

Printing Top 10 Nodes as per Eigenvector Centrality
Node Label:  1912 => 755
Node Label:  2266 => 234
Node Label:  2206 => 210
Node Label:  2233 => 222
Node Label:  2464 => 202
Node Label:  2142 => 221
Node Label:  2218 => 205
Node Label:  2078 => 204
Node Label:  2123 => 203
Node Label:  1993 => 203
```



- **KatzCentrality**

Eigenvector centrality is not ideal for directed graphs because eigenvector centrality would not take zero in-degree nodes into account in directed graphs. Katz centrality computes the relative influence of a node within a network by measuring the number of the immediate neighbors (first degree nodes) and also all other nodes in the network that connect to the node under consideration through these immediate neighbors.

```
def katzCentrality(G):
    pos = nx.spring_layout(G)
    katzCent = nx.katz_centrality(G)
    node_color = [20000.0 * G.degree(v) for v in G]
    node_size =  [v * 10000 for v in katzCent.values()]
    plt.figure(figsize=(15,15))
    nx.draw_networkx(G, pos=pos, with_labels=False,
                     node_color=node_color,
                     node_size=node_size )
    plt.axis('off')
    #Printing Top 10 Nodes as per Eigenvector Centrality
    sorted_degree=sorted(katzCent, key=katzCent.get, reverse=True)[:10]
    for d in sorted_degree[:10]:
        print("Node Label: ",d,"=>",G.degree[d])
#Facebook Network Katz Centality
print("Facebook Network Katz Centality:\n")
katzCentrality(G1)
# Wikipedia Network Katz Centality
print("\nWikipedia Network Katz Centality:\n")
katzCentrality(G2)
```

- ## Pagerank

Page Rank can be considered as an extension of Katz centrality . The websites on the web can be represented as a directed graph, where hypermedia links between websites determine the edges.

Let's consider a popular web directory website with high Katz centrality value which has millions of links to other websites. It would contribute to every single website significantly, nevertheless not all of them are important.

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more

important websites are likely to receive more links from other websites.

```
def pagerankCentrality(G):
    pos = nx.spring_layout(G)
    pr = nx.pagerank(G, alpha = 0.8)
    node_color = [20000.0 * G.degree(v) for v in G]
    node_size =  [v * 10000 for v in pr.values()]
    plt.figure(figsize=(15,15))
    nx.draw_networkx(G, pos=pos, with_labels=False,
                     node_color=node_color,
                     node_size=node_size )
    plt.axis('off')
    #Printing Top 10 Nodes as per Eigenvector Centrality
    sorted_degree=sorted(pr, key=pr.get, reverse=True)[:10]
    for d in sorted_degree[:10]:
        print("Node Label: ",d,"=> Degree: ",G.degree[d])
#Facebook Network Pagerank Centality
print("Facebook Network Pagerank Centality:\n")
pagerankCentrality(G1)
# Wikipedia Network Pagerank Centality
print("\nWikipedia Network Pagerank Centality:\n")
pagerankCentrality(G2)
```

Facebook Network Pagerank Centality:

```
Node Label:  3437 => Degree:  547
Node Label:  107 => Degree:  1045
Node Label:  1684 => Degree:  792
Node Label:  0 => Degree:  347
Node Label:  1912 => Degree:  755
Node Label:  348 => Degree:  229
Node Label:  686 => Degree:  170
Node Label:  3980 => Degree:  59
Node Label:  414 => Degree:  159
Node Label:  698 => Degree:  68
```

Wikipedia Network Pagerank Centality:

```
Node Label:  2565 => Degree:  1065
Node Label:  4037 => Degree:  467
Node Label:  11 => Degree:  743
Node Label:  457 => Degree:  732
Node Label:  766 => Degree:  773
Node Label:  1549 => Degree:  740
Node Label:  1166 => Degree:  688
Node Label:  2688 => Degree:  618
Node Label:  15 => Degree:  403
Node Label:  2237 => Degree:  387
```

• Betweenness Centrality

Betweenness centrality measures how important a node is to the shortest paths through the network.

To compute betweenness for a node N, we select a pair of nodes and find all the shortest paths between those nodes. Then we compute the fraction of those shortest paths that include node N.

We repeat this process for every pair of nodes in the network. We then add up the fractions we computed, and this is the betweenness centrality for node N

```
def betweenessCentrality(G):
    pos = nx.spring_layout(G)
    betCent = nx.betweenness_centrality(G)
    node_color = [20000.0 * G.degree(v) for v in G]
    node_size =  [v * 10000 for v in betCent.values()]
    plt.figure(figsize=(20,20))
    nx.draw_networkx(G, pos=pos, with_labels=False,
                     node_color=node_color,
                     node_size=node_size )
    plt.axis('off')
    #Printing Top 10 Nodes as per Betweeness Centrality
    sorted_degree=sorted(betCent, key=betCent.get, reverse=True)[:10]
    for d in sorted_degree[:10]:
        print("Node Label: ",d,"=> Degree: ",G.degree[d])
#Facebook Network Betweeness Centality
print("Facebook Network Betweeness Centality:\n")
betweenessCentrality(G1)
# Wikipedia Network Betweeness Centality
print("\nWikipedia Network Betweeness Centality:\n")
betweenessCentrality(G2)
```

```
Facebook Network Betweeness Centality:

Node Label:  107 => Degree:  1045
Node Label:  1684 => Degree:  792
Node Label:  3437 => Degree:  547
Node Label:  1912 => Degree:  755
Node Label:  1085 => Degree:  66
Node Label:  0 => Degree:  347
Node Label:  698 => Degree:  68
Node Label:  567 => Degree:  63
Node Label:  58 => Degree:  12
Node Label:  428 => Degree:  115
```

Wikipedia Network Betweeness Centality:

```
Node Label:  2565 => Degree:  1065
Node Label:  11 => Degree:  743
Node Label:  457 => Degree:  732
Node Label:  4037 => Degree:  467
Node Label:  1549 => Degree:  740
Node Label:  766 => Degree:  773
Node Label:  1166 => Degree:  688
Node Label:  15 => Degree:  403
Node Label:  1374 => Degree:  533
Node Label:  2237 => Degree:  387
```

• Closeness Centrality

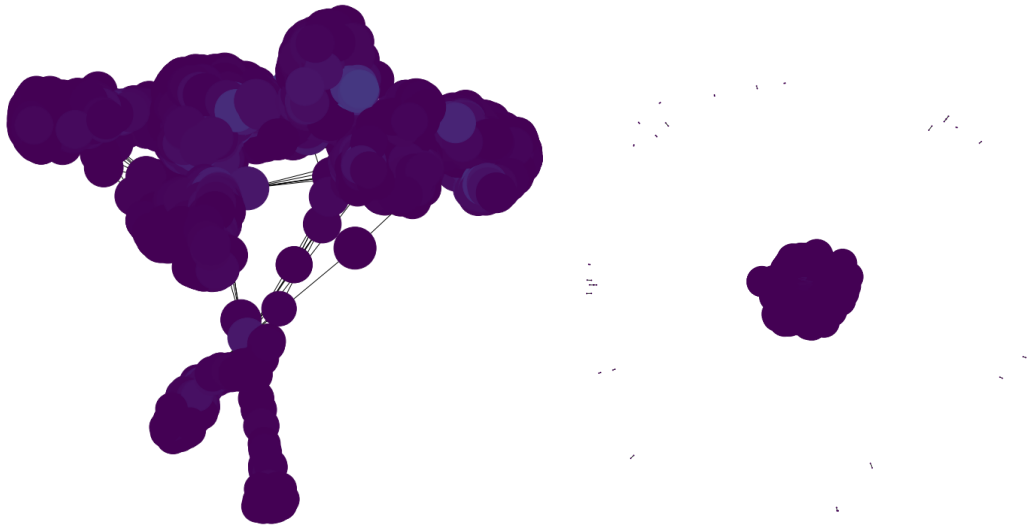
For each node, the Closeness Centrality algorithm calculates the sum of its distances to all other nodes,based on calculating the shortest paths between all pairs of nodes.

The closeness centrality for our sample came out to be:

```
def closenessCentrality(G):
    pos = nx.spring_layout(G)
    cloCent = nx.closeness_centrality(G)
    node_color = [20000.0 * G.degree(v) for v in G]
    node_size =  [v * 10000 for v in cloCent.values()]
    plt.figure(figsize=(13,13))
    nx.draw_networkx(G, pos=pos, with_labels=False,
                     node_color=node_color,
                     node_size=node_size )
    plt.axis('off')
    #Printing Top 10 Nodes as per Closeness Centrality
    sorted_degree=sorted(cloCent, key=cloCent.get, reverse=True)[:10]
    for d in sorted_degree[:10]:
        print("Node Label: ",d,"=> Degree: ",G.degree[d])
#Facebook Network Closeness Centality
print("Facebook Network Closeness Centality:\n")
closenessCentrality(G1)
# Wikipedia Network Closeness Centality
print("\nWikipedia Network Closeness Centality:\n")
closenessCentrality(G2)
```

```
Facebook Network Closeness Centality:

Node Label:  107 => Degree:  1045
Node Label:  58 => Degree:  12
Node Label:  428 => Degree:  115
Node Label:  563 => Degree:  91
Node Label:  1684 => Degree:  792
Node Label:  171 => Degree:  22
Node Label:  348 => Degree:  229
Node Label:  483 => Degree:  231
Node Label:  414 => Degree:  159
Node Label:  376 => Degree:  133
```

```
Wikipedia Network Closeness Centality:

Node Label:  2565 => Degree:  1065
Node Label:  766 => Degree:  773
Node Label:  457 => Degree:  732
Node Label:  1549 => Degree:  740
Node Label:  1166 => Degree:  688
Node Label:  1374 => Degree:  533
Node Label:  11 => Degree:  743
Node Label:  1151 => Degree:  517
Node Label:  2688 => Degree:  618
Node Label:  2485 => Degree:  435
```



- Group Betweenness Centrality

```
def groupBetweenCentrality(G):
    pos = nx.spring_layout(G)
    gbCent = nx.group_betweenness_centrality(G)
    node_color = [20000.0 * G.degree(v) for v in G]
    node_size =  [v * 10000 for v in gbCent.values()]
    plt.figure(figsize=(13,13))
    nx.draw_networkx(G1, pos=pos, with_labels=False,
                     node_color=node_color,
                     node_size=node_size )
    plt.axis('off')
    #Printing Top 10 Nodes as per Closeness Centrality
    sorted_degree=sorted(gbCent, key=gbCent.get, reverse=True)[:10]
    for d in sorted_degree[:10]:
        print("Node Label: ",d,"=> Degree: ",G.degree[d])

#Facebook Network Group Betweeeness Centality
print("Facebook Network Group Betweeeness Centality:\n")
groupBetweenCentrality(G1)
# Wikipedia Network Group Betweeeness Centality
print("\nWikipedia Network Group Betweeeness Centality:\n")
groupBetweenCentrality(G2)
```

2. Clustering Coefficient

The clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together.

In most real-world networks, and in particular social networks, nodes tend to create tightly knit groups characterized by a relatively high density of ties, this likelihood tends to be greater than the average probability of a tie randomly established between two nodes.

Two versions of this measure exist: the global and the local. The global version was designed to give an overall indication of the clustering in the network, whereas the local gives an indication of the embeddedness of single nodes.

- Local and Global Clustering Coefficient|

```
all_nodes = list(G1.nodes())
for node in all_nodes:
    local_clustering_coefficient=nx.clustering(G1,node)
    global_clustering_coefficient=(3*nx.triangles(G1,node))/nx.transitivity(G1)
    print("Local Clustering Coefficient of ",node," = ",local_clustering_coefficient)
    print("Global Clustering Coefficient of ",node," = ",global_clustering_coefficient)
```

- Average Clustering Coefficient

```
print("No. Of Nodes of Facebook Social Graph:",G1.number_of_nodes())
print("No. Of Edges of Facebook Social Graph",G1.number_of_edges())
print("No. Of Nodes of Wikipedia Network Graph:",G2.number_of_nodes())
print("No. Of Edges of Wikipedia Network Graph:",G2.number_of_edges())

# Average Clustering Coefficient
average_clustering_coefficient_G1=nx.average_clustering(G1)
average_clustering_coefficient_G2=nx.average_clustering(G2)
print("Average Clustering Coefficient of Facebook Social Graph: ", average_clustering_coefficient_G1)
print("Average Clustering Coefficient of Wikipedia Network Graph: ", average_clustering_coefficient_G2)
```

```
No. Of Nodes of Facebook Social Graph: 4039
No. Of Edges of Facebook Social Graph 88234
No. Of Nodes of Wikipedia Network Graph: 7115
No. Of Edges of Wikipedia Network Graph: 100762
Average Clustering Coefficient of Facebook Social Graph:  0.6055467186200876
Average Clustering Coefficient of Wikipedia Network Graph:  0.14089784589308738
```

3. Reciprocity:

Reciprocity is a measure of the likelihood of vertices in a directed network to be mutually linked. Like the clustering coefficient, scale-free degree distribution, or community structure, reciprocity is a quantitative measure used to study complex networks.

1.3 Reciprocity

```
print("Reciprocity of Facebook Social Graph is: ",nx.overall_reciprocity(G1))
print("Reciprocity of Wikipedia Vote Network Graph is: ",nx.overall_reciprocity(G2))
```

```
Reciprocity of Facebook Social Graph is:  0.0
Reciprocity of Wikipedia Vote Network Graph is:  0.0
```

4. Transitivity:

Transitivity refers to the extent to which the relation that relates two nodes in a network that are connected by an edge is transitive. Perfect transitivity implies that if x is connected to y , and y is connected to z , then x is connected to z as well.

1.4 Transitivity

```
print("Transitivity of Facebook Social Graph is: ",nx.transitivity(G1))
print("Transitivity of Wikipedia Vote Network Graph is: ",nx.transitivity(G2))
```

```
Transitivity of Facebook Social Graph is:  0.5191742775433075
Transitivity of Wikipedia Vote Network Graph is:  0.12547914899233995
```

Problem Statement 2:

Try to get an algorithm package in Python to find the maximum connected component (called a giant component in the class) in a given graph G. Let us denote the number of nodes in the giant component of a graph G as N_G. Vary ⟨k⟩ from 0 to 5 with increment of 0.1. For each value of ⟨k⟩ find the ratio N_G/N where N is the number of nodes in the graph. Plot this ratio with respect to ⟨k⟩. Take ⟨k⟩ as x-axis and ratio N_G/N as y-axis.

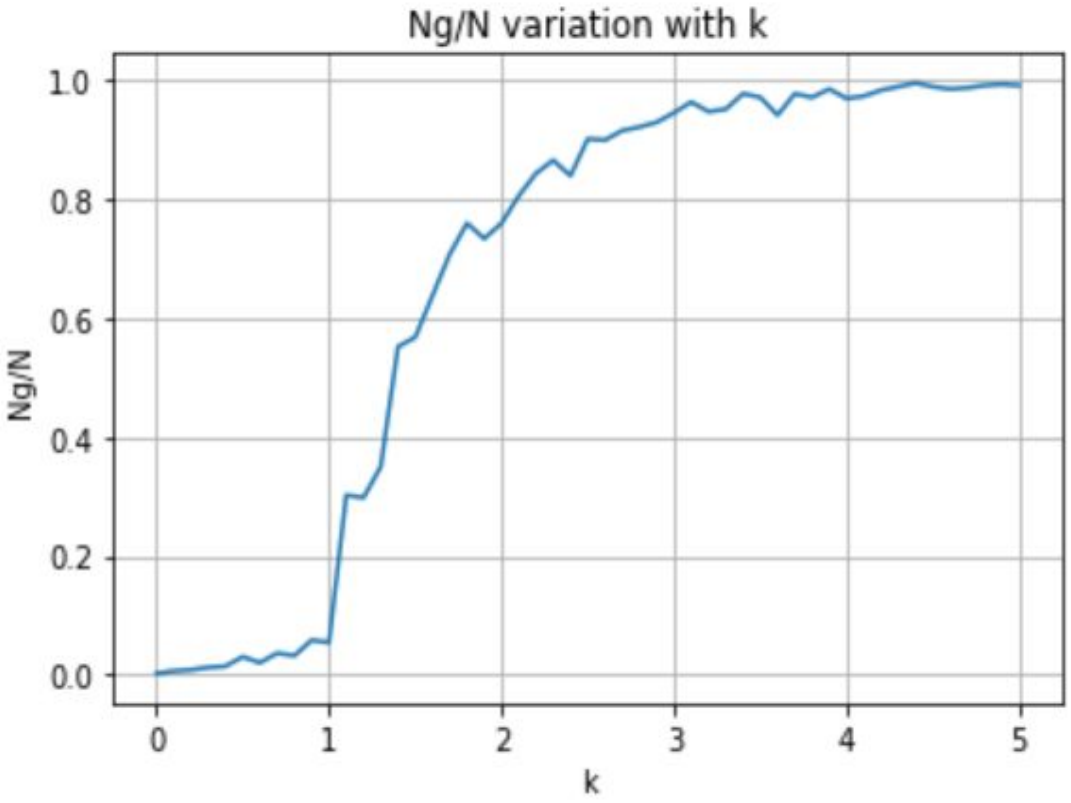
Problem Statement 2: Giant component variation

```
giant_facebook = len(max(nx.connected_components(G1), key=len))
print("Number of node in the Giant Componenet of G ="+str(giant_facebook)+'\n')
giant_wiki = len(max(nx.connected_components(G2), key=len))
print("Number of node in the Giant Componenet of G ="+str(giant_wiki)+'\n')
```

```
Number of node in the Giant Componenet of G =4039

Number of node in the Giant Componenet of G =7066
```

```
# Importing required libraries
import networkx as nx
import matplotlib.pyplot as plt
import pylab as plt

n = 500
arr = {} # Dictionary to store k and corresponding Ng/N value
for i in range(0,51):
    k = float(i/10) # k varies by 0.1
    p = k/(n-1)
    g = nx.erdos_renyi_graph(n, p)
    # finding Giant component and ng
    giant = max(nx.connected_components(g), key=len)
#     ng = giant_components(giant)
    ng = len(giant)
    # adding the ratio to dictionary
    arr[k] = float(ng/n)
# since dictionary is unsorted collection, sorting wrt k for plotting
temp = sorted(arr.items())
# unzipping keys and values of dictionary into x and y
# x takes the value of k , y takes the ratio in sorted order
x,y = zip(*temp)
```

```
# plotting the histogram
plt.figure()
plt.grid(True)
plt.plot(x,y)
plt.title('Ng/N variation with k')
plt.xlabel( 'k' )
plt.ylabel( 'Ng/N' )
plt.show()
plt.close()
```



Conclusion:

Erdős-Rényi Network model of random graphs was taken with n=500 nodes. The average degree varied from 0 to 5 with incrementation of 0.1 for each iteration and a giant component was observed in each graph iteration.The plot increased from 0 and as we kept increasing the average degree, the ratio kept getting closer and closer to 1. This observation is in accordance with our preliminary analysis, where we said that as the probability of edge formation p increases, (p = k/n-1, which implies p is directly proportional to average degree k), the size of the giant component increases.

Github Repository:

https://github.com/Raghavlakhotia/social_media_analysis_

References:

- https://networkx.org/documentation/stable/reference/algorithms/centrality.html
- https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.components.strongly_connected_components.html#networkx.algorithms.components.strongly_connected_components