Graph Theory

Semester Project

Topic:

Social Network Analysis

Submitted To:

Dr. Irfan Yousaf

Submitted By:

Ahmad Zayab (2020-CS-629)

Department of Computer Science

University of Engineering and Technology, New Campus, Lahore

Question(1)

Different Measures:

The Social-circles Facebook graph contains information about the relationships between individuals, represented by friendship links or edges connecting different people. This information is stored in text files in the form of an edge list, which contains tuples representing the starting and ending nodes of each edge. The starting node is the point of origin for the edge, while the ending node is the destination point where the edge is incident. These text files provide a comprehensive view of the graph's structure and the connections between nodes. In summary, the edge list files capture the interconnectivity of the Social-circles Facebook graph, allowing for analysis and visualization of the relationships between individuals in the network.

Code:

```
import networkx as nx


# Read the graph from the text file with
open('Facebook.txt', 'r') as f:
    edges = [tuple(map(int, line.strip().split())) for line in f.readlines()]
# Create a NetworkX graph object and add the edges
G = nx.Graph()
G.add_edges_from(edges)


# Compute the average degree of the nodes in the graph avg_degree
= sum(dict(G.degree()).values()) / float(len(G))


# Compute the average clustering coefficient of the nodes in the graph avg_clustering
= nx.average_clustering(G)


# Compute the average path length of the graph avg_path_length
= nx.average_shortest_path_length(G)


# Compute the diameter of the graph diameter
= nx.diameter(G)


# Find the node with the highest degree in the graph
highest_degree_node = max(dict(G.degree()).items(), key=lambda x: x[1])
# Find the node with the lowest degree in the graph lowest_degree_node
= min(dict(G.degree()).items(), key=lambda x: x[1])


# Print the results print("Average_degree:",
avg_degree) print("Average_clustering
coefficient:", avg_clustering)
print("Average_path length:",
avg_path_length) print("Diameter:",
```

```
diameter) print("Node with the highest
degree:", highest_degree_node) print("Node
with the lowest degree:",
lowest_degree_node)
```

Values:

Question(2)

Cumulative Frequency Distributions:

Graph Facebook:

Code:

import networkx as nx import

matplotlib.pyplot as plt

Load graph from file

G = nx.read_edgelist("Graph_A.txt")

Compute degree distribution degree_sequence = sorted([d for n,

d in G.degree()], reverse=True) degree_count = {}

for d in degree_sequence:

if d in degree_count:

degree_count[d] += 1

else:

degree_count[d] = 1 degree_dist = [(k, v / len(degree_sequence)) for k, v

in degree_count.items()]

Compute clustering coefficient distribution cc_sequence =

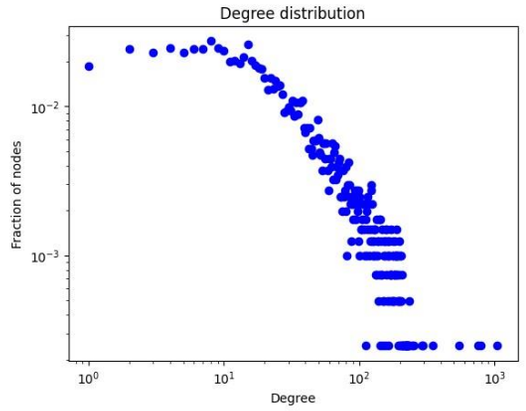
sorted(nx.clustering(G).values(), reverse=True) cc_count = {}

for cc in cc_sequence: if cc in cc_count: cc_count[cc]

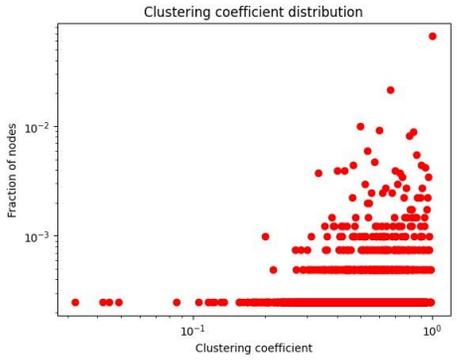
+= 1 else:

```
        cc_count[cc] = 1 cc_dist = [(k, v / len(cc_sequence)) for k,
v in cc_count.items()]


# Compute path length distribution
path_lengths = [] for node in
G.nodes():
    path_lengths.extend(nx.single_source_shortest_path_length(G, node).values())
pl_sequence = sorted(path_lengths) pl_count = {} for pl in pl_sequence:     if pl in
pl_count:         pl_count[pl] += 1     else:
        pl_count[pl] = 1 pl_dist = [(k, v / len(pl_sequence)) for k,
v in pl_count.items()]


# Plot degree distribution plt.figure() plt.plot([d for d, _ in
degree_dist], [p for _, p in degree_dist], 'bo') plt.xscale('log')
plt.yscale('log')
plt.xlabel('Degree') plt.ylabel('Fraction
of nodes') plt.title('Degree distribution')
plt.show()


# Plot clustering coefficient distribution plt.figure()
plt.plot([cc for cc, _ in cc_dist], [p for _, p in cc_dist], 'ro')
plt.xscale('log') plt.yscale('log') plt.xlabel('Clustering
coefficient') plt.ylabel('Fraction of nodes')
plt.title('Clustering coefficient distribution') plt.show()


# Plot path length distribution plt.figure() plt.plot([pl for
pl, _ in pl_dist], [p for _, p in pl_dist], 'go')
plt.xscale('log') plt.yscale('log') plt.xlabel('Path length')
plt.ylabel('Fraction of nodes') plt.title('Path length
distribution') plt.show()
```

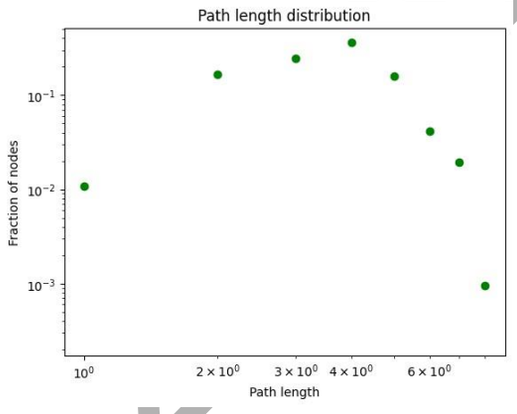
Graphs:



2:



3:



Explanation:

The graph represents the degree distribution of Group A, indicating the probability distribution of node degrees throughout the network. The majority of nodes exhibit small degrees, while a few have extremely high degrees. This distribution pattern suggests that the network may be a scale-free network. The hub node is evident from the highest count at the degree of 7.6, indicating its significance in the network. Conversely, the lowest count degree of 10 suggests that there are very few nodes with very high degrees in the network.

Clustering coefficient: Explain: The graph displays the clustering coefficient distribution of Graph A in a CDF format, with cumulative values ranging from 0 to 1. The range of 0.0 to 0.2 on the xaxis indicates that the cumulative probability value is 0.0, indicating a low clustering coefficient. The probability then gradually increases, indicating that the network has a few highly connected nodes that form tightly clustered communities while others do not. Overall, the graph suggests that the network exhibits a mix of high and low clustering coefficients.

Question(3)

Degree Centrality:

Top 10 nodes with highest degree centrality:
Node 107: 0.259
Node 1684: 0.196
Node 1912: 0.187
Node 3437: 0.135
Node 0: 0.086
Node 2543: 0.073
Node 2347: 0.072
Node 1888: 0.063
Node 1800: 0.061
Node 1663: 0.058

Bottom 10 nodes with lowest degree centrality:
Node 3856: 0.000
Node 3935: 0.000
Node 3974: 0.000
Node 3984: 0.000
Node 4008: 0.000
Node 4010: 0.000
Node 4015: 0.000
Node 4022: 0.000
Node 4024: 0.000
Node 4035: 0.000

Closeness Centrality:

Top 10 nodes based on Closeness centrality:
Node 107: 0.460 Node 58: 0.397
Node 428: 0.395
Node 563: 0.394
Node 1684: 0.394
Node 171: 0.370
Node 348: 0.370
Node 483: 0.370
Node 414: 0.370
Node 376: 0.367

Bottom 10 nodes based on Closeness centrality:
Node 785: 0.178
Node 854: 0.178
Node 699: 0.178
Node 744: 0.178
Node 749: 0.178
Node 750: 0.178
Node 775: 0.178
Node 841: 0.178
Node 692: 0.178
Node 801: 0.178

Betweenness Centrality:

Top 10 nodes with highest betweenness centrality:

107: 0.481
1684: 0.338
3437: 0.236
1912: 0.229
1085: 0.149
0: 0.146
698: 0.115
567: 0.096 58:
0.084
428: 0.064

Bottom 10 nodes with lowest betweenness centrality:

11: 0.000
12: 0.000
15: 0.000
18: 0.000
32: 0.000
33: 0.000
35: 0.000
37: 0.000
42: 0.000 43:
0.000

Eigen Centrality:

Top 10 nodes by Eigen Vector centrality:

1912: 0.095
2266: 0.087
2206: 0.086
2233: 0.085
2464: 0.084
2142: 0.084
2218: 0.084
2078: 0.084
2123: 0.084
1993: 0.084

Bottom 10 nodes by Eigen Vector centrality:

802: 0.000
750: 0.000
743: 0.000
788: 0.000
699: 0.000
841: 0.000
749: 0.000
775: 0.000
692: 0.000
801: 0.000

Pager Rank Centrality:

Top 10 nodes by Pagerank centrality:

Node 3437: 0.008
Node 107: 0.007
Node 1684: 0.006
Node 0: 0.006
Node 1912: 0.004
Node 348: 0.002
Node 686: 0.002
Node 3980: 0.002
Node 414: 0.002
Node 698: 0.001

Bottom 10 nodes by Pagerank centrality:

```
Node 1560: 0.000
Node 1581: 0.000
Node 1834: 0.000
Node 2079: 0.000
Node 2195: 0.000
Node 2269: 0.000
Node 2457: 0.000
Node 2470: 0.000
Node 2569: 0.000
Node 2596: 0.000
```

Code:

```
------------1.1----------------------------------

Degree Centrality: import

networkx as nx



# Load the graph from the file

G = nx.read_edgelist("Graph_A.txt")



# Calculate the degree centrality for all nodes deg_centrality

= nx.degree_centrality(G)



# Sort the nodes by degree centrality in descending order sorted_nodes =

sorted(deg_centrality.items(), key=lambda x: x[1], reverse=True)



# Print the top 10 nodes with highest degree centrality

print("Top 10 nodes with highest degree centrality:")

for node, centrality in sorted_nodes[:10]:

print(f"Node {node}: {centrality:.3f}")



# Print the bottom 10 nodes with lowest degree centrality

print("\nBottom 10 nodes with lowest degree centrality:")

for node, centrality in sorted_nodes[-10:]:

print(f"Node {node}: {centrality:.3f}")

------------1.2----------------------------------

Closeness Centrality: import

networkx as nx



# Load the graph from the text file G

= nx.read_edgelist("Graph_A.txt")



# Find the Closeness centrality for all nodes
```

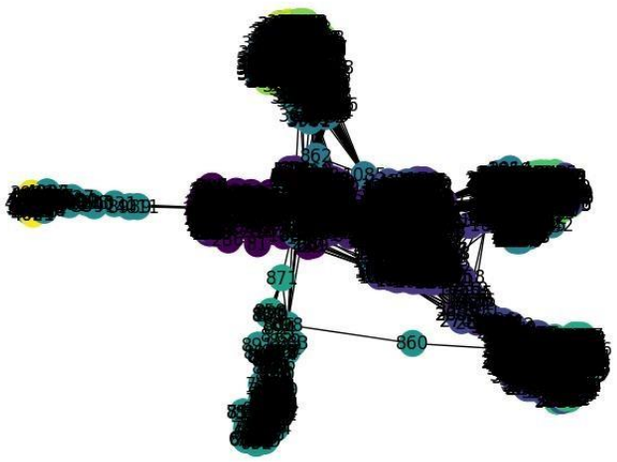
```
closeness = nx.closeness_centrality(G)

# Sort the nodes based on their Closeness centrality in descending order sorted_closeness
= sorted(closeness.items(), key=lambda x: x[1], reverse=True)

# Print the top 10 nodes based on Closeness centrality
print("Top 10 nodes based on Closeness centrality:") for
node, centrality in sorted_closeness[:10]:
print(f"Node {node}: {centrality:.3f}")

# Print the bottom 10 nodes based on Closeness centrality
print("\nBottom 10 nodes based on Closeness centrality:")
for node, centrality in sorted_closeness[-10:]:
print(f"Node {node}: {centrality:.3f}")
--------------1.3------------------------------
Betweeness Centrality: import
networkx as nx

# Load the graph from file
G = nx.read_edgelist("Graph_A.txt")

# Calculate betweenness centrality betweenness
= nx.betweenness_centrality(G)

# Get top 10 and bottom 10 nodes with highest betweenness centrality top10
= sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:10]
bottom10 = sorted(betweenness.items(), key=lambda x: x[1])[:10]

# Print the results print("Top 10 nodes with highest
betweenness centrality:") for node, centrality in top10:
    print(f"{node}: {centrality:.3f}")
 print("\nBottom 10 nodes with lowest betweenness
centrality:") for node, centrality in bottom10:
    print(f"{node}: {centrality:.3f}")
----------------------1.4----------------
Eigen Vector Centrality: import
networkx as nx
```

```
# Load the graph
G = nx.read_edgelist('Graph_A.txt')

# Calculate the eigen vector centrality eigen_cen
= nx.eigenvector_centrality(G)

# Sort the dictionary by values eigen_cen_sorted = sorted(eigen_cen.items(),
key=lambda x: x[1], reverse=True)

# Get the top 10 nodes and bottom 10 nodes
top10 = eigen_cen_sorted[:10] bottom10 =
eigen_cen_sorted[-10:]

# Print the results print("Top 10 nodes by Eigen
Vector centrality:") for node, score in top10:
    print(f"{node}: {score:.3f}")
     print("\nBottom 10 nodes by Eigen Vector
centrality:") for node, score in bottom10:
print(f"{node}: {score:.3f}") ===========1.5----------
---------
Pagerrank Centrality: import
networkx as nx

# create the graph
Graph_A = nx.read_edgelist('Graph_A.txt', nodetype=int)

# calculate pagerank centrality pagerank
= nx.pagerank(Graph_A) # sort the nodes
by their pagerank values in descending
order sorted_nodes =
sorted(pagerank.items(), key=lambda x:
x[1], reverse=True)

# print the top 10 nodes with their pagerank centrality values
print("Top 10 nodes by Pagerank centrality:") for node, pr in
sorted_nodes[:10]:     print(f"Node {node}: {pr:.3f}")
```

```
# print the bottom 10 nodes with their pagerank centrality values
print("\nBottom 10 nodes by Pagerank centrality:") for node, pr
in sorted_nodes[-10:]:     print(f"Node {node}: {pr:.3f}")
```

Question (4)

Communities in graph:



The graph shows the nodes and edges of the input graph, where nodes are colored according to their community membership. The communities were found using the asynchronous label propagation algorithm. Different colors indicate different communities.