Social Network Analysis for Computer Scientists
Fall 2024 — Assignment 2

Jiameng Ma(s4255445)

October 2024

Exercise 1: Diameter computation

1. **Initialization**

   - For each node v, initialize:
     - $\epsilon_L(v) = -\infty$ (lower bound, initially unknown),
     - $\epsilon_U(v) = +\infty$ (upper bound, initially unknown).
   - Initialize the global bounds for the diameter:
     - $\Delta_L = -\infty$ (lower bound for the diameter),
     - $\Delta_U = +\infty$ (upper bound for the diameter).

2. **First Node Selection (Node with highest degree)**

   We start by selecting **node F**, which has the highest degree (6).

   - **BFS from node F** Perform BFS from node F to compute its eccentricity (longest shortest path from F to any other node).

Node	**Distance from F**	**Path from F**
A	2	via C
B	2	via C
C	1	-
D	1	-
E	1	-
G	2	via J or E
H	1	-
I	2	via J
J	1	-
K	2	via L
L	1	-
M	2	via L
N	2	via L
P	2	via L
Q	3	via L, P
R	3	via L, P
S	4	via L, P, Q
T	4	via L, P, Q
U	5	via L, P, Q, S

Table 1: Shortest paths from node F to all other nodes in the graph

Thus, the eccentricity of F is $\epsilon(F) = 5$ (longest distance from F to U).

- **Update global bounds**

$$\Delta_L = 5, \quad \Delta_U = 2 \times 5 = 10$$

Update bounds for other nodes based on their distance from F:

Node	ϵ_L (Lower Bound)	ϵ_U (Upper Bound)
A	3	7
B	3	7
C	4	6
D	4	6
E	4	6
F	5	5
G	3	7
H	4	6
I	3	7
J	4	6
K	3	7
L	4	6
M	3	7
N	3	7
P	3	7
Q	3	8
R	3	8
S	4	9
T	4	9
U	5	10

Table 2: Updated bounds for all nodes after BFS from node F.

From now, we can remove node F since its lower and upper bounds are equal.

3. **Second Node Selection (Node with largest upper bound)**

   Next, select **node U**, which has the largest upper bound $\epsilon_U(U) = 10$.

   - **BFS from node U**

Node	**Distance from U**	**Path from U**
A	7	via S, Q, P, L, F, C
B	7	via S, Q, P, L, F, C
C	6	via S, Q, P, L, F
D	6	via S, Q, P, L, F
E	6	via S, Q, P, L, F
G	7	via S, Q, P, L, F, E
H	6	via S, Q, P, L, F
I	7	via S, Q, P, L, F, J
J	6	via S, Q, P, L, F
K	5	via S, Q, P, L
L	4	via S, Q, P
M	5	via S, Q, P, L
N	4	via S, Q, P
P	3	via S, Q
Q	2	via S
R	3	via S, Q
S	1	-
T	3	via S, Q
U	0	- (Starting Node)

Table 3: Shortest paths from node U to all other nodes in the graph

Thus, the eccentricity of U is $\epsilon(U) = 7$.

- **Update global bounds**

$$\Delta_L = \max(\Delta_L, \epsilon(U)) = \max(5, 7) = 7$$
$$\Delta_U = \min(\Delta_U, 2 \times \epsilon(U)) = \min(10, 2 \times 7) = \min(10, 14) = 10$$

Thus,
$$\Delta_L = 7, \quad \Delta_U = 10.$$

Update bounds for other nodes based on their distance from U:

Node	$\epsilon_L[w]$ (Lower Bound)	$\epsilon_U[w]$ (Upper Bound)
A	7	7
B	7	7
C	6	6
D	6	6
E	6	6
G	7	7
H	6	6
I	7	7
J	6	6
K	5	7
L	4	6
M	5	7
N	4	7
P	4	7
Q	5	8
R	4	8
S	6	8
T	4	9
U	7	7

Table 4: Updated bounds for all nodes after BFS from node U

From now, we can remove nodes A, B, C, D, E, G, H, I, J, U since their lower and upper bounds are equal and remove K, M since their bounds are sufficiently tight relative to the global bounds($\epsilon_U[w] \leq \Delta_L$and $\epsilon_L[w] \geq \Delta_U/2$).

4. **Third Node Selection (smallest lower bound)**

   Now select **node L**, which has a smallest lower bound.

- **BFS from node L**

Node	**Distance from L**	**Path from L**
N	1	-
P	1	-
Q	1	via P
R	1	via P
S	3	via Q, P
T	3	via Q, P
U	4	via S, Q, P

Table 5: Shortest paths from node L to all other nodes in the graph

Perform BFS from node L. The eccentricity of L is:

$$\epsilon(T) = 4.$$

- **Update global bounds**

$$\Delta_L = \max(\Delta_L, \epsilon(T)) = \max(7, 4) = 7$$

$$\Delta_U = \min(\Delta_U, 2 \times \epsilon(T)) = \min(10, 2 \times 4) = \min(10, 8) = 8$$

Thus,

$$\Delta_L = 7, \quad \Delta_U = 8.$$

Node	$\epsilon_L[w]$ (Lower Bound)	$\epsilon_U[w]$ (Upper Bound)
L	4	4
N	4	5
P	4	5
Q	5	6
R	4	6
S	6	7
T	4	7

Table 6: Updated bounds for all nodes after BFS from node L.

From now, all the nodes's bounds are sufficiently tight relative to the global bounds, so the algorithm stops since the bounds for all nodes have converged. After 3 iterations, the lower bound converges to the exact diameter of **7**.

The naive method (All-Pairs Shortest Path) would have required calculating the shortest path between every pair of nodes, with a time complexity of $O(n^3)$. In contrast, the BoundingDiameters algorithm required only 3 iterations, making it much more efficient.

The exact diameter of the graph is **7**, and the BoundingDiameters algorithm took only 3 iterations to compute this, compared to the much higher computational cost of the naive APSP method.

Exercise 2: Museum as a Network

Question 2.1: The corresponding graph is an undirected graph when modeling a two-dimensional physical layout of a building. That because the connections (edges) between rooms are bidirectional, we can use nodes represent rooms in museum and use edges represent connections between them.

Question 2.2: My graph has 51 nodes and 69 edges.

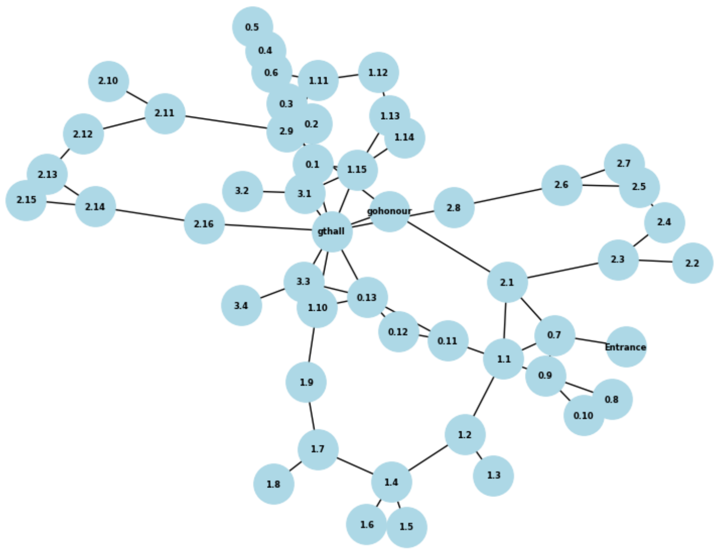

Figure 1: museum layout

Question 2.3:

- Question 2.3a: The clustering coefficient measures the degree to which nodes in a graph tend to cluster together. It is defined as the number of closed triples divided by the total number of triples involving that node. But to an infinite regular graph with degree 4, while each node has neighbors, there are no triangles formed among those neighbors. A node connected to four others does not necessarily mean those neighbors are connected to each other Thus, the clustering coefficient of such a regular graph is 0.

- Question 2.3b: Not a regular or empty graph but clustering coefficient behaves in the same way(close to 0) is a sparse random graph. When the number of edges is low compared to the number of nodes, most pairs of nodes would not be connected, leading to very few triangles. Therefore, the overall structure can lead to a low clustering coefficient.

Question 2.4:

- Question 2.4a: I aim to compute an open Traveling Salesperson Problem (Open TSP) path. It directly addresses the key objectives and constraints of navigating a museum efficiently. Open TSP is specifically designed to find the shortest path covering each point without needing to return to the starting point. And I also want this path goes from 'entrance' considering the reality.

Algorithm 1 Traverse Graph to Construct TSP Path

```
Input: Graph $G$ with weighted edges
Output: List of nodes representing the TSP path
FMainconstruct_tsp_path FnFunction:
Step 1: Initialize variables entrance_node ← 'Entrance   current_node ←
entrance_node  visited_nodes ← set()  tsp_path ← []
Step 2: Traverse until all nodes are visited
• while length(visited_nodes)<length(G.nodes) do
tsp_path.append(current_node)  visited_nodes.add(current_node)
Step 3: Get unvisited neighbors with weights neighbors ← [(neighbor,
G[current_node][neighbor].get('weight', 1))
•   for neighbor in G.neighbors(current_node)
•     if neighbor $\notin$ visited_nodes then do]
•       if neighbors is not empty then next_node ←
min(neighbors, key=lambda x: x[1])[0]
•         else unvisited_nodes ← [node for node in G.nodes
•           if node $\notin$ visited_nodes then]
•             if unvisited_nodes is empty then break
Step 4: Calculate shortest distances to unvisited nodes distances ←
{node: nx.shortest_path_length(G, current_node, node, weight='weight')
 for node in unvisited_nodes}
next_node ← min(distances, key=distances.get)
tsp_path.extend(nx.shortest_path(G, current_node, next_node,
weight='weight')[:-1])
current_node ← next_node
```

Path for my constructed network: ['Entrance', '0.7', '1.1', '1.2', '1.3', '1.2', '1.4', '1.5', '1.4', '1.6', '1.4', '1.7', '1.8', '1.7', '1.9', '1.10', '0.13', '0.11', '0.9', '0.8', '0.9', '0.10', '0.9', '0.7', '2.1', 'gohonour', 'gthall', '0.1', '0.2', '0.3', '0.4', '0.5', '0.6', '1.11', '2.9', '2.11', '2.10', '2.11', '2.12', '2.13', '2.14', '2.15', '2.14', '2.16', 'gthall', '1.15', '1.13', '1.12', '1.13', '1.14', '1.15', '3.1', '3.2', '3.1', 'gthall', '3.3', '3.4', '3.3', '0.13', '0.12', '0.13', 'gthall', '2.8', '2.6', '2.5', '2.4', '2.3', '2.2', '2.3', '2.4', '2.5', '2.7']

- Question 2.4b: The path generally progresses through different rooms in a structured manner, but there are still several instances of repeated nodes. These repetitions could indicate inefficient backtracking and may create some unnecessary movement between already-visited rooms. In a real museum, visitors usually prefer a continuous path without unnecessary loops or revisits. While this path might cover all rooms, the experience may feel repetitive, and visitors might prefer a more streamlined route with a clear, linear progression.

  Suggestion:

  – **Weight Adjustments:** Adding or adjusting weights based on the real distance between rooms could help prioritize closer rooms and potentially reduce unnecessary revisits.

  – **Add More Direct Connections:** Creating additional edges between frequently visited nodes (e.g., 1.4 and 1.7, 2.2 and 2.5) could make alternative routes possible, reducing backtracking.

Question 2.5:
The transformation involves the following steps:

1. **Identify Critical Nodes**: Doorways and stairs are the nodes of interest. Let a function $D(x)$ denote whether a node x represents a doorway (including entrances, hallways, and rooms ending in digits), and let $S(x)$ denote whether x is a stair node.
2. **Filter Nodes**: Only include nodes that satisfy $D(x) = \text{True}$ or $S(x) = \text{True}$.
3. **Filter Edges**: Retain only those edges where both endpoints are part of the critical node set from step 2.

Algorithm 2 Constructed Network Transformation

```
FMaintransform_network FnFunction: edges
Step 1: Initialize new sets for nodes and edges NEW_NODES ← ∅ NEW_EDGES ← ∅
Step 2: Define functions to identify doorways and stairs is_doorway(node) return node contains "Entrance" or node contains "gthall" or node ends with a digit
is_stair(node) return node contains "stair" *[r]Modify based on stair identifiers
Step 3: Identify relevant nodes $(u, v) \in$ edges
if is_doorway($u$) or is_stair($u$) then add $u$ to NEW_NODES
  if is_doorway($v$) or is_stair($v$) then add $v$ to NEW_NODES
Step 4: Retain relevant edges $(u, v) \in$ edges
if $u \in$ NEW_NODES and $v \in$ NEW_NODES then add $(u, v)$ to NEW_EDGES
Step 5: Construct the new graph NEW_GRAPH ← (NEW_NODES, NEW_EDGES)
return NEW_GRAPH
```

Question 2.6:
To identify the most suitable room for placing the first-aid kit, we need to locate the "central" room(s) within the museum layout graph, where the distance to any other room is minimized. This is a classic graph center problem. The center of a graph is the node that minimizes the maximum distance to all other nodes.

Thus, I calculate the eccentricity of each room which is the greatest distance (in terms of rooms traversed) from that node to any other node in the network. The center of the graph is the node with the smallest eccentricity. Then I sort the nodes by eccentricity and choose the top three with the smallest values.

The three most suitable rooms are **1.10, 2.1** and **Great hall**.

Exercise 3: Twitter network analysis

Question 3.1:

- **Parsing the TSV File:** The input file is in TSV (tab-separated values) format, with each line representing a tweet. For each line, we identified three main components: Timestamp, Username and Tweet text. The script reads each line as a list of values separated by tabs and skipped rows with fewer than three elements since these rows would not contain the necessary information to extract a valid mention.
- **Extracting Mentions:** Mentions were identified in tweets by searching for `@username` patterns using the regular expression `@(\w+)`, which captures valid Twitter usernames (alphanumeric and underscores only). This regex avoids capturing parts of URLs or hashtags but may require adjustments for more complex username patterns.
- **Building the Adjacency List:** An adjacency list was created using a dictionary where each user (source) links to the users they mention (targets), with each mention incrementing a weight count. Duplicate mentions within the same tweet were counted only once.
- **Outputting a Weighted Edge List:** The adjacency list was saved as a TSV with columns source, target, and weight, representing directed edges in the mention graph.

Algorithm 3 Extract Mention Graph from Twitter Data

```
Input: Twitter data in a TSV file twitter-smaller.tsv
Output: Weighted edge list in a TSV file mention_graph.tsv
FMainextract_mention_graph FnFunction:
Step 1: Initialize the adjacency list mentions_graph ← defaultdict(lambda: defaultdict(int))
Step 2: Define regex pattern for extracting mentions mention_pattern ← re.compile(r"@(+)")
Step 3: Parse the input TSV file
for each line in twitter-smaller.tsv do split line by tab
    if length(row) <3 then continue Skip if row doesn't have the expected format
user ← row[1]  User who posted the tweet tweet ← row[2]  Content of the tweet
Step 4: Find all mentions in the tweet mentions ← mention_pattern.findall(tweet)
Step 5: Update mentions count for each mentioned user
        for each mention in mentions do mentions_graph[user][mention] += 1
Step 6: Output the adjacency list as a weighted edge list Open file mention_graph.tsv for writing
write header: ['source', 'target', 'weight']
    for each user, targets in mentions_graph.items() do
        for each target, weight in targets.items() do
write [user, target, weight] to file
```

Question 3.2:

Statistic	**Value**
Number of nodes	94862
Number of edges	155574
Number of strongly connected components	93117
Size of largest SCC	1385
Number of weakly connected components	3513
Size of largest WCC	81670
Density	1.7288489192832098e-05
Average clustering coefficient	0.0438155559528315
Average distance in the giant component	6.5407

Table 7: Statistics of the Mention Graph(small)

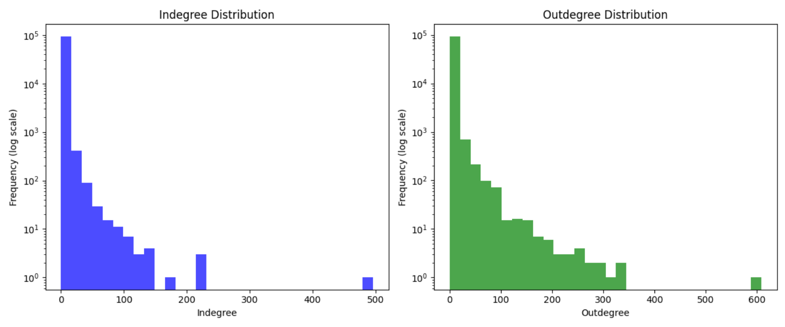

Figure 2: degree distribution(small)

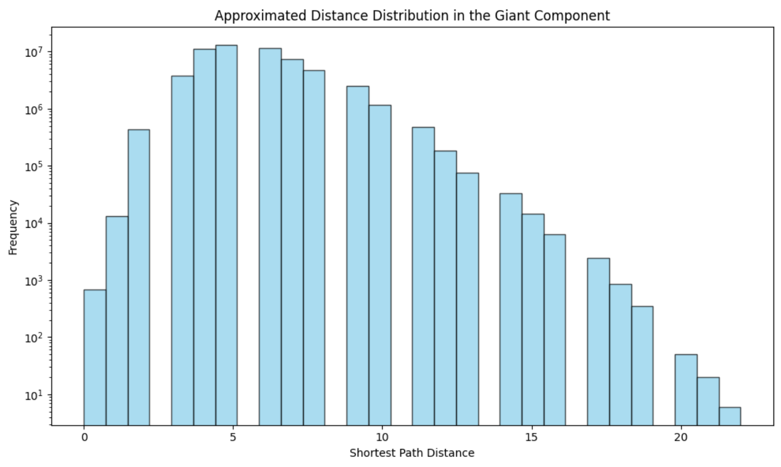

Figure 3: distance distribution of the giant component(small)

Question 3.3:
How to handle directionality:

- **Degree centrality** will be calculated for both in-degree and out-degree to reflect the number of times a user is mentioned and the number of times they mention others.
- **Betweenness centrality** can be calculated in a directed manner (taking directionality into account) using NetworkX.
- **Closeness centrality** can be calculated using either the directed graph or the undirected version. I suggest using the directed version unless the network is very fragmented.

For the results:

- **Degree centrality:** Users with high in-degree are likely popular or important, as they receive many mentions while users with high out-degree are active, frequently mentioning others.
- **Betweenness centrality:** Users with high betweenness centrality are crucial intermediaries in the network, controlling the flow of information between different parts of the graph.
- **Closeness centrality:** Users with high closeness centrality are "central" in the sense that they can quickly reach all other nodes in the graph.

I choose to use Kendall's Tau. This method compares the similarity between two ranked lists. For three centrality measures, I compare pairs of rankings.
Kendall's Tau measures the similarity of rankings:

- A value of 1 means perfect agreement between two rankings.
- A value of -1 means perfect disagreement.
- A value close to 0 indicates no significant relationvihip between the rankings.

Discussion of the results I get:

- **Degree vs. Betweenness (Tau = 0.22):** his positive Tau value indicates a moderate alignment between the rankings by degree centrality and betweenness centrality. This is expected because users with high connectivity (degree) may have higher chances of being positioned on many shortest paths, increasing their betweenness centrality. However, the moderate value suggests that other factors (such as specific network positioning) also influence betweenness.
- **Degree vs. Closeness (Tau = -0.06):** The slightly negative Tau suggests very low, potentially inverse correlation between degree and closeness centrality. This could mean that users with high degrees do not necessarily have shorter paths to all other nodes, as closeness depends more on the average path length to all nodes than on immediate connections.
- **Betweenness vs. Closeness (Tau = -0.02):** The near-zero Tau here suggests minimal to no alignment between betweenness and closeness rankings. This indicates that nodes important for connecting paths (high betweenness) do not inherently have shorter paths to all nodes, suggesting that these centralities capture different aspects of node influence.

Question 3.4:

I applying the **Louvain community detection algorithm** on the giant component of graph, yielding a modularity score of approximately **0.6407**.

Modularity values typically range from -1 to 1, where values greater than 0.3 often suggest meaningful community structure. A score of 0.6407 is relatively high and suggests that the communities are well-defined.

The high modularity score suggests that users in your this graph are forming distinct groups or communities where interactions are denser within the groups than between them. This can reflect social dynamics where certain users or groups have stronger ties based on shared interests, activities, or communication styles.

Algorithm 4 Community Detection on the Giant Component of a Mention Graph

```
Input: Directed Mention Graph $G$
Output: Community structure in the giant component of $G$
1. Extract the Giant Component:
    (a) Find the largest weakly connected component (WCC) in $G$.
    (b) Create an undirected subgraph $G'$ containing only the nodes and edges
        of the largest WCC.
2. Apply Community Detection (e.g., Louvain Algorithm):
    (a) Initialize a dictionary, partition, to store community assignments.
    (b) For each node $v$ in $G'$:
            (i) Assign $v$ to a community label using a modularity optimization
                technique (e.g., Louvain method) to maximize modularity.
    (c) Save the community label of each node in partition.
3. Visualize and Interpret Communities:
    (a) Visualize $G'$ with nodes colored by their community labels in
        partition.
    (b) Calculate the modularity score of the partitioned graph to evaluate
        community separation.
4. Analyze Community Structure:
    (a) Identify the number and size of each community.
    (b) For each community:
            (i) Analyze central nodes within the community (using metrics like
                degree or betweenness centrality).
            (ii) Identify nodes that connect different communities and consider
                 their roles as potential bridges.
5. Output Results:
    (a) Return partition (node-to-community assignments) and modularity
        score for interpretation.
    (b) Summarize key findings about the community structure in terms of
        major communities, key nodes, and inter-community connections.
=0
```

Question 3.5: Box plot is a proper choice for presenting the weight distribution in this graph because it effectively summarizes the data, identifies outliers, allows for comparison across groups, utilizes space efficiently, and communicates findings clearly.

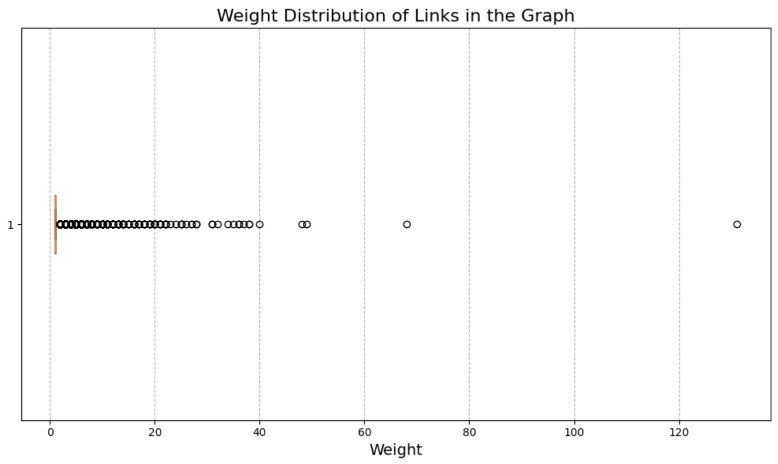

Figure 4: weight distribution of links in the graph

Question 3.6:

Statistic	**Value**
Number of nodes	546585
Number of edges	1327858
Number of strongly connected components	519906
Size of largest SCC	24568
Number of weakly connected components	13675
Size of largest WCC	505803
Density	4.444644301663516e-06
Average clustering coefficient	0.06613759864844608
Average distance in the giant component	5.6065

Table 8: Statistics of the Mention Graph(large)

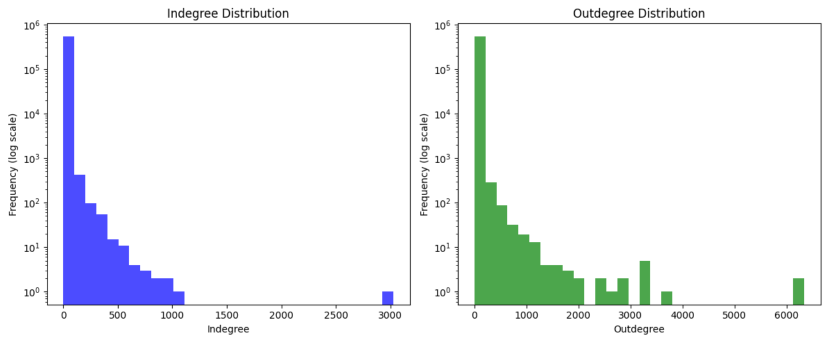

Figure 5: degree distribution (large)

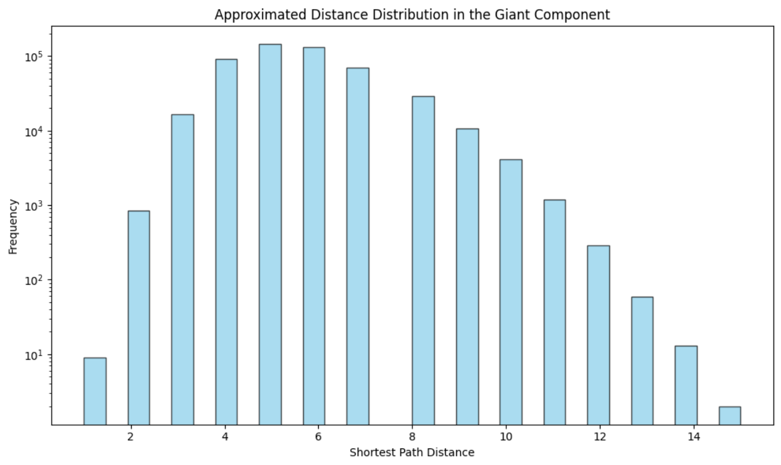

Figure 6: distance distribution of the giant component(large)