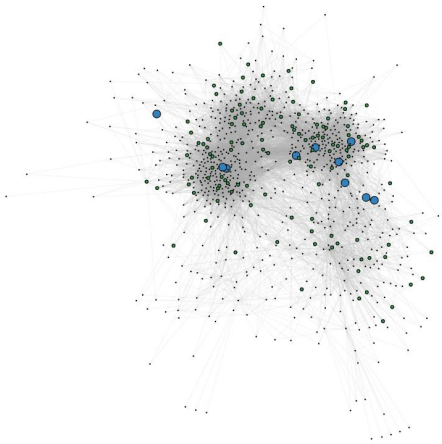
Report of Q7
Social Media Network Analysis

50015627 JIANG Zhuoyang

1. Question Description:

Different users can become friends on social networks, forming a vast social network. In this question, I received user social network data from a certain platform. I was required to conduct various graph analyses and provide insights into social relationships:

- Load the Social Media Network as a Graph
- Customized Graph Visualization Method
- Graph Analysis on Social Media Network



2. Load the Social Media Network as a Graph:

In this task, I've chosen igraph as the primary tool for graph analysis and processing. However, I'll also utilize some unique analysis methods from NetworkX. This decision primarily stems from igraph's significantly faster speed when handling large-scale networks compared to NetworkX.

I've imported edge-list data using igraph and displayed the node range as follows:

```
Minimum node ID: 1
Maximum node ID: 1157827
```

Figure 1. Node Range In Edge Table

Later, following the requirements, I utilized igraph to import an edge-list, creating a graph with 1,157,827 nodes. Here's the display of the total number of nodes in this graph:

```
Number of nodes in the graph: 1157827
```

Figure 2. Total Number of Nodes

However, the total number of nodes in the edge-list doesn't match this count, indicating the presence of numerous isolated nodes that do not appear within the edge-list.

3. Customized Graph Visualization Method:

After completing the import of edge-list data into a graph, I designed a visualization function to be used for subsequent analysis. Since this network is a large and complex one, displaying all nodes might not be ideal. Hence, I designed a function to visualize a subset of representative nodes in the graph, providing visually appealing and analyzable visualizations for a higher degree of credibility in visual analysis. The logic of my visualization method can be outlined as follows:

Build and Visulize a subgraph containing:

- The 10 most influential nodes (represented as large blue nodes),
- The 20 most influential neighbors of these 10 nodes (displayed as medium-sized green nodes),
- The 30 most influential neighbors of these 20 nodes (depicted as small grey nodes).

The algorithm is implemented as follows:

```python
def plot_subgraph(g, input_node_list=None):
    '''
    Build and Visulize a subgraph containing:
    1.the 10 most influential nodes,
    2.their 20 most influential neighbors,
    3.their 30 most influential neighbors' neighbors
    '''
    if input_node_list is not None:
        # Visualization for a node list
        most_influential_nodes_degrees = {n: g.degree(n) for n in input_node_list}
        most_influential_nodes = sorted(most_influential_nodes_degrees, key=most_influential_nodes_degrees.get, reverse=True)[:10]
    else:
        # Visualization for a graph
        degree_centrality = g.degree()
        most_influential_nodes = sorted(range(len(degree_centrality)), key=lambda x: degree_centrality[x], reverse=True)[:10]

    inner_nodes = most_influential_nodes.copy()
    midle_nodes = inner_nodes.copy()
    outer_nodes = midle_nodes.copy()

    for inner_node in inner_nodes:
        neighbors_of_inner = g.neighbors(inner_node)
        neighbor_of_inner_degrees = {n: g.degree(n) for n in neighbors_of_inner}

        top_neighbors_of_inner = sorted(neighbor_of_inner_degrees, key=neighbor_of_inner_degrees.get, reverse=True)[:20]

        midle_nodes.extend(top_neighbors_of_inner)  # midle_nodes COVER inner_nodes
        outer_nodes.extend(top_neighbors_of_inner)  # outer_nodes COVER inner_nodes

        for midle_node in midle_nodes:
            neighbors_of_midle = g.neighbors(midle_node)
            neighbor_of_midle_degrees = {n: g.degree(n) for n in neighbors_of_midle}

            top_neighbors_of_middle = sorted(neighbor_of_midle_degrees, key=neighbor_of_midle_degrees.get,
                                             reverse=True)[:30]

            outer_nodes.extend(top_neighbors_of_middle)  # outer_nodes COVER midle_nodes

    # Deduplicated
    midle_nodes = list(set(midle_nodes))
    outer_nodes = list(set(outer_nodes))

    # Subgraph with the nodes of interest
    subgraph = g.subgraph(outer_nodes)

    # Mapping dictionary from original graph indices to subgraph indices
    mapping = {subgraph.vs[node_idx].index: node for node_idx, node in enumerate(outer_nodes)}

    # Get vertices colors and sizes for the subgraph using the mapping dictionary
    vertex_colors = ['#3182bd' if mapping[subgraph.vs[i].index] in inner_nodes else (
        '#31a354' if mapping[subgraph.vs[i].index] in midle_nodes else '#e6550d') for i in range(len(subgraph.vs))]
    vertex_sizes = [10 if mapping[subgraph.vs[i].index] in inner_nodes else (
        4 if mapping[subgraph.vs[i].index] in midle_nodes else 1) for i in range(len(subgraph.vs))]

    # Plotting
    visual_style = {}
    visual_style["layout"] = subgraph.layout_fruchterman_reingold(seed=random.seed(42)) # Random Seeds
    visual_style["vertex_color"] = vertex_colors
    visual_style["vertex_size"] = vertex_sizes
    visual_style["edge_color"] = '#AAAAAA'
    visual_style["edge_width"] = 0.1

    return plot(subgraph, **visual_style)
```

Figure 3. visualization Algorithm

4. Graph Analysis on Social Media Network:

Once the data import and visualization algorithm development are completed, the preparatory phase for analysis is finished. We can proceed to analyze the network's graph structure from the following seven perspectives:

- Clustering Coefficient AND Degree Distribution
- The Most Influential Nodes
- Isolated Nodes
- Connected Components
- Average Shortest Path Length
- Diameter of the Network
- Community Detection

(1) Clustering Coefficient AND Degree Distribution

Once the data import and visualization algorithm development are completed, the preparatory phase for analysis is finished. We can proceed to analyze the graph structure of this network from the following seven aspects:

```
Global Clustering Coefficient: 0.006218559818028638
Average Clustering Coefficient: 0.1722579595214077
```

Figure 4. Clustering Coefficient Result

The Global Clustering Coefficient represents the proportion of closed triplets (three nodes forming a closed triangle) in the entire network. Its low value (0.0062) suggests a relatively low occurrence of closed triangles among the nodes in the network. This might imply the existence of fewer tightly-knit clusters or tendencies for nodes to cluster together within the network.

The Average Clustering Coefficient, with its relatively high value (0.1722), indicates that many nodes in the network form relatively dense subgraphs or small clusters around them.

In summary, the combination of these two metrics suggests that the network may exhibit dispersed global connectivity but potentially denser connections or communities among local nodes. Users in the network tend to form smaller groups, yet the overall connectivity might not be very tight.

Next, I computed and visualized the degree distribution (using the `degree_distribution()` method in igraph) on a logarithmic scale for better visualization of degree distribution characteristics. The results are shown in the following graph.

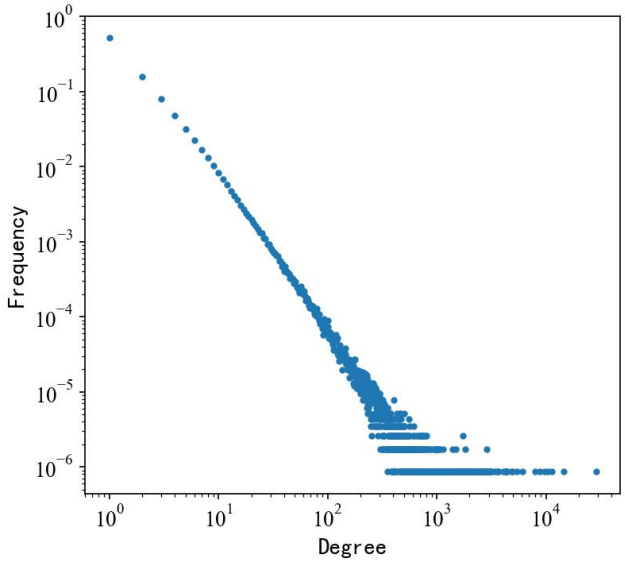


Figure 5. Degree Distribution Visualization Result

From the degree distribution, we can observe that this network exhibits a characteristic commonly found in many large-scale networks: a "Power-Law" degree distribution.

(2) The Most Influential Nodes

Then, I further utilized node degree as a measure of node influence and computed the top ten most influential nodes. The results are as follows:

```
Most Influential Nodes ID: [1072, 363, 35661, 106, 482709, 663931, 929, 808, 27837, 108624]
```

Figure 6. The IDs of the top 10 most influential nodes

I utilized the previously implemented visualization algorithm to visualize the influence of these selected nodes within the network. The results are displayed below:

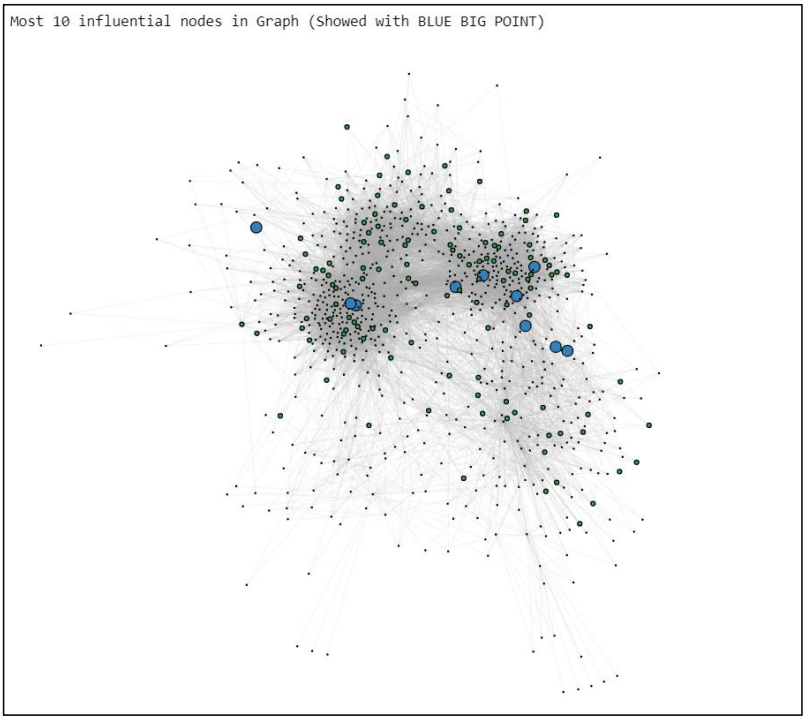


Figure 7. The Visualization of the top 10 most influential nodes

It's noticeable that the blue nodes in the graph represent the most influential nodes, which correspond to the most influential users within the network. It's apparent that their core spheres of influence overlap, as seen in the shared green nodes (neighbors).

(3) Isolated Nodes

Next, we can further explore the isolated nodes, which represent the "lonely users" within the social network. By using the `select(_degree=0)` method from igraph, we can obtain all nodes in the graph with a degree of 0. Then, we can count their number and print their IDs.

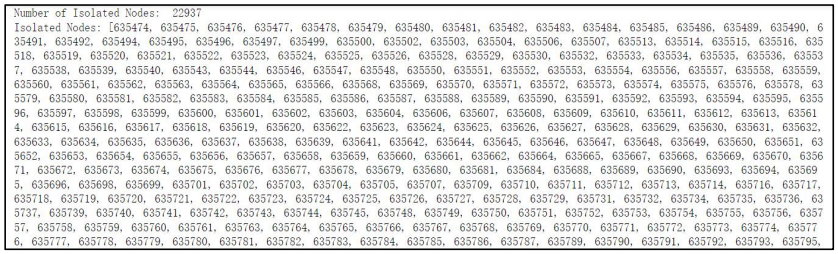


Figure 8. Isolated Nodes

It appears that there are a total of 22,937 isolated nodes, constituting roughly 2% of the entire 1,157,827 nodes in the network.

(4) Connected Components

Correspondingly, by utilizing the `connected_components()` method in igraph, we can obtain all connected components within the network. Upon further investigation, I found that within these connected components, the entire set of nodes from the edge-list forms a single connected component, which also happens to be the only and largest connected component.

```
Connected Components: Clustering with 1157827 elements and 22938 clusters
Number of non-isolated nodes in each non-isolated-connected-component: [1134890]
Total number of non-isolated nodes in all connected components: 1134890
```

Figure 9. An overview of the connected components

(5) Average Shortest Path Length

Later, I need to compute the Average Shortest Path Length of the network. Precisely calculating this graph feature incurs significant computational costs. Given that our network is complex and relatively dense, even the most efficient algorithms have complexities of $O(n^2)$. Consequently, directly using the igraph method to compute this value would result in unacceptable time overhead.

Therefore, to compromise on the computational precision, I'll resort to certain methods such as random sampling or Monte Carlo approaches to approximate the Average Shortest Path Length. This may not yield an exact average shortest path length but could provide a relatively quick estimation, particularly for large-scale networks.

Here, I attempted to employ a random sampling method for approximate computation. Initially, I explored three different sample sizes for computation: 20,000, 50,000, and 100,000. The outcomes are as follows:

```
approximate_avg_shortest_path_length on 20000 nodes: 6.358722744465105
approximate_avg_shortest_path_length on 50000 nodes: 3.713723837617643
approximate_avg_shortest_path_length on 100000 nodes: 6.515791883780898
```

Figure 10. Approximated ASPL for Different Sampling Capacities

After computing the avg_shortest_path_length for various magnitudes of randomly selected nodes, I observed that its range should fall between 5 and 7. With a total of 1,157,827 nodes, where only one non-isolated connected component exists - the largest connected component comprising 1,134,890 nodes - I iterated 11 times, computing avg_shortest_path_length by randomly sampling 100,000 nodes each time. Finally, I used the average of these computations as an estimation for avg_shortest_path_length.

```
Approximate Average Shortest Path Length: 6.00502367886632
```

Figure 11. Final Approximated ASPL Calculation

The approximate value I obtained for the average shortest path length, around 6 (precisely 6.00502367886632), suggests that there exist relatively short paths between most nodes in the network, allowing for relatively rapid information or influence propagation. This length also aligns well with the characteristic of a "small-world network," where despite its size, the average distance between nodes remains relatively short.

(6) Diameter of the Network

It is also unable to withstand precise calculations when I use diameter() directly， Thus, I attempted to combine a heuristic algorithm with BFS to create an approximation method. However, the computational overhead from the theoretical complexity remained unacceptable to me.

As NetworkX is the only library offering an approximate method to compute the diameter, I ultimately utilized the nx.algorithms.approximation.diameter() method, resulting in the following calculation:

```
import networkx as nx
file_path = 'Data_Q7/socialmedia.graph.txt'
G = nx.read_edgelist(file_path, nodetype=int)
diameter_estimate = nx.algorithms.approximation.diameter(G)
print("Approximate Diameter:", diameter_estimate)
```

```
Approximate Diameter: 24
```

Figure 12. The final approximate diameter

This approximation may not be entirely accurate, but based on this value and in conjunction with the earlier computation of avg_shortest_path_length, a diameter of 24 doesn't necessarily indicate poor network connectivity for large-scale networks. In certain networks, especially social networks, the internet, or complex networks, the diameter might be relatively large while still maintaining good global connectivity.

(7) Community Detection

Finally, community detection serves as a crucial analytical tool to reflect network characteristics. Since the network consists of a single largest connected component, I directly conducted community detection within this connected component.

In igraph, the community_multilevel() method implements a modularity-based community detection algorithm using **a multi-level optimization approach**. This algorithm is **an improvement upon the Louvain method**, optimizing modularity across multiple levels to identify community structures.

Its functioning is broadly as follows:

- Initialization: Assign each node to its own community, forming an initial community partition.
- Multi-level Optimization: The algorithm iteratively merges communities and checks the impact of each merge on modularity. It attempts to move each node from its current community to a neighboring community to improve modularity.
- Multi-level Structure: In each iteration, communities are merged to form a new hierarchical structure, and further optimization is performed at that level.
- Termination Criteria: The algorithm stops when modularity no longer increases or when a predefined number of iterations is reached.

This algorithm's advantage lies in its ability to handle large-scale networks while maintaining relatively high resolution, often exhibiting good performance in identifying suitable community structures.

Here, by utilizing this method, I identified a total of 6046 communities formed by all the nodes in the edge list. It's worth noting that the number of communities detected fluctuates between 5500 to 7500 in multiple test runs.

```
largest_component = g.components().giant()
communities = largest_component.community_multilevel()
print("Number of communities:", len(communities))
```

```
Number of communities: 6046
```

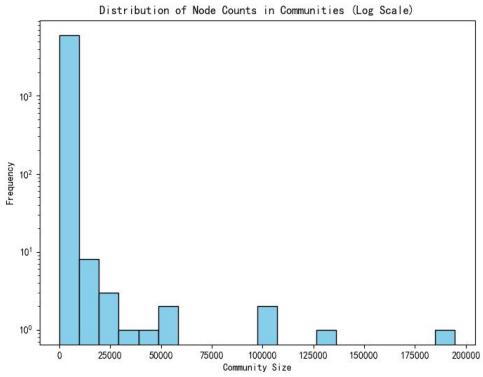


Figure 13. 6046 Community detected with their Node-distribution.

Next, I visualized the three largest communities, each showcasing distinct characteristics, which I summarized as follows:

- **1. Core-Homogeneous Loosely Connected Communities:** The core points within the community are uniformly distributed across various locations in the graph.

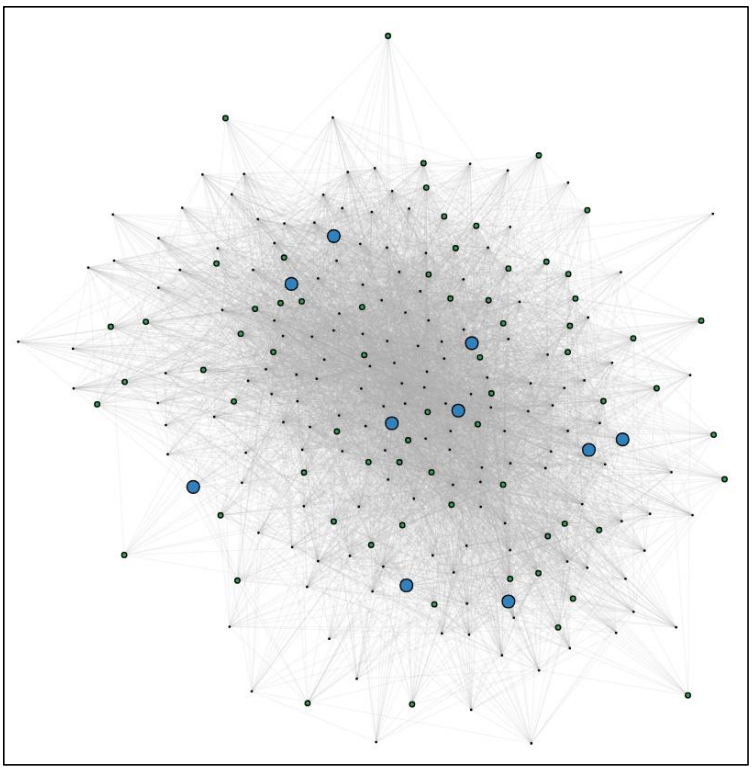


Figure 14. Core-Homogeneous Loosely Connected Communities

- **2. Core-Dense Communities:** The core points within the community are densely interconnected.

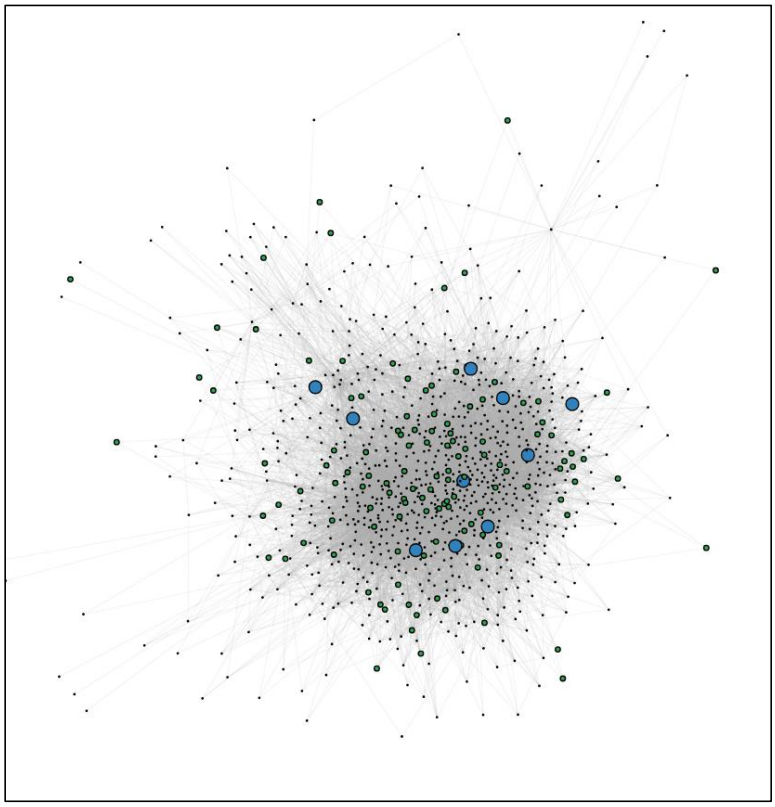


Figure 15. Core-Dense Communities

- **3. "Star-Satellite" communities:** The central core forms the largest 'star' community, surrounded by several 'satellite' communities orbiting around it.

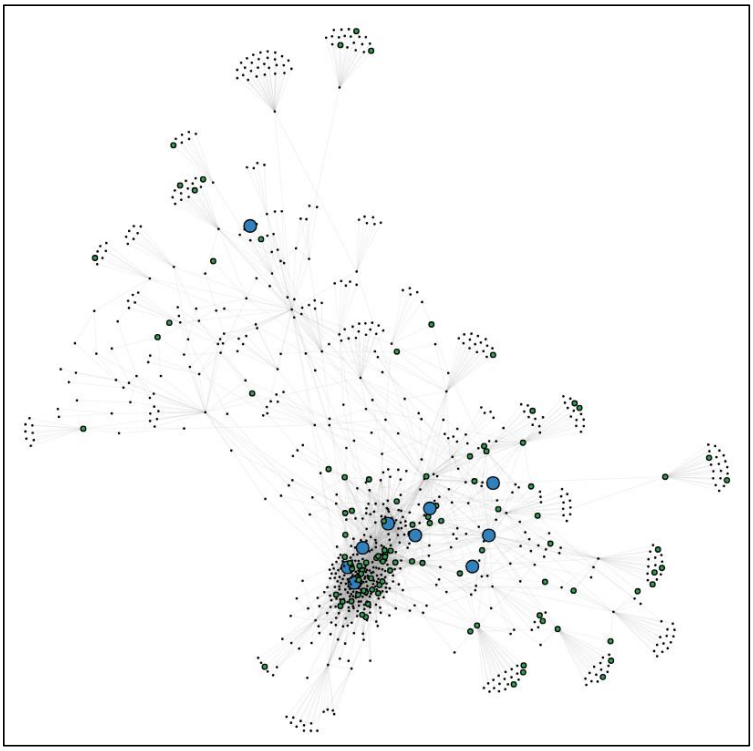


Figure 16. "Star-Satellite" communities