**Social Network Analysis: Identifying Influential Users with Python-Based Graph Structures**

Student Name: Bhawesh Shrestha

Student ID: 005027566

Course: Algorithm and Data Structure

University: University of the Cumberlands

Date: July 27, 2025

# **1. Introduction**

In the digital age, social networks play a crucial role in how people interact, share information, and influence one another. Platforms such as Twitter, Facebook, and Instagram rely heavily on data structures to manage users, connections, and interactions at scale. Identifying influential users in these networks is valuable for marketing, political campaigns, and trend analysis. This project aims to analyze a social network graph to determine the most influential users using Python. We model users and their follow relationships as a directed graph and use metrics like in-degree to assess influence.

# **2. Application Context and Chosen Data Structures**

The selected application context is social network analysis, focusing on identifying influential users in a simplified Twitter-like environment. The system needs to support:  
- Efficient storage and traversal of user connections  
- Fast updates and lookups for influence scores  
- Ranking and extracting top influencers

To achieve these goals, the following data structures are implemented:

|  |  |  |
| --- | --- | --- |
| **Data Structure** | **Purpose** | **Justification** |
| Graph (Adjacency List) | Model user connections | Scales well for sparse graphs; enables efficient BFS/DFS |
| Hash Table | Store user influence scores | Provides average-case O(1) access and updates |
| Max Heap (Priority Queue) | Rank users by influence | Enables fast top-K extraction with O(log n) insertion |

# **3. Design Rationale**

- Graph (Adjacency List): Social networks are naturally modeled as graphs. An adjacency list is memory-efficient and allows fast iteration over a user's connections.  
- Hash Table: Used to track influence scores (e.g., follower count) per user. Python's built-in dictionary provides optimal performance.  
- Max Heap: To find the top-K influential users, we use a priority queue implemented as a max heap. Python's heapq supports a min-heap by default; we invert scores to simulate a max-heap.  
  
These choices reflect practical constraints in real-world social networks, such as the need to handle millions of users and real-time updates.

# **4. Python Implementation**

Below are simplified snippets of the Python implementation:

import heapq

class Graph:

def \_\_init\_\_(self):

self.adj\_list = {}

def add\_user(self, user):

if user not in self.adj\_list:

self.adj\_list[user] = []

def add\_connection(self, from\_user, to\_user):

self.add\_user(from\_user)

self.add\_user(to\_user)

self.adj\_list[from\_user].append(to\_user)

class UserMetrics:

def \_\_init\_\_(self):

self.metrics = {}

def update\_score(self, user, score):

self.metrics[user] = score

def get\_score(self, user):

return self.metrics.get(user, 0)

class MaxHeap:

def \_\_init\_\_(self):

self.heap = []

def insert(self, user, score):

heapq.heappush(self.heap, (-score, user))

def extract\_top\_k(self, k):

return [heapq.heappop(self.heap) for \_ in range(min(k, len(self.heap)))]

def calculate\_influence(graph):

user\_score = {}

for user in graph.adj\_list:

score = 0

for followers in graph.adj\_list.values():

if user in followers:

score += 1

user\_score[user] = score

return user\_score

# Sample influencer connection data

connections = [

("ElonMusk", "lexfridman"),

("lexfridman", "elonmusk"),

("BarackObama", "BillGates"),

("BillGates", "BarackObama"),

("elonmusk", "BarackObama"),

("taylorswift13", "selenagomez"),

("selenagomez", "taylorswift13"),

("Cristiano", "elonmusk"),

("Cristiano", "taylorswift13"),

("BillGates", "lexfridman"),

("elonmusk", "Cristiano")

]

# Build graph

graph = Graph()

for from\_user, to\_user in connections:

graph.add\_connection(from\_user, to\_user)

# Compute influence scores

metrics = UserMetrics()

heap = MaxHeap()

scores = calculate\_influence(graph)

for user, score in scores.items():

metrics.update\_score(user, score)

heap.insert(user, score)

# Display top 5 influencers

top\_influencers = heap.extract\_top\_k(5)

print("Top Influential Users:")

for score, user in top\_influencers:

print(f"{user}: {-score}")

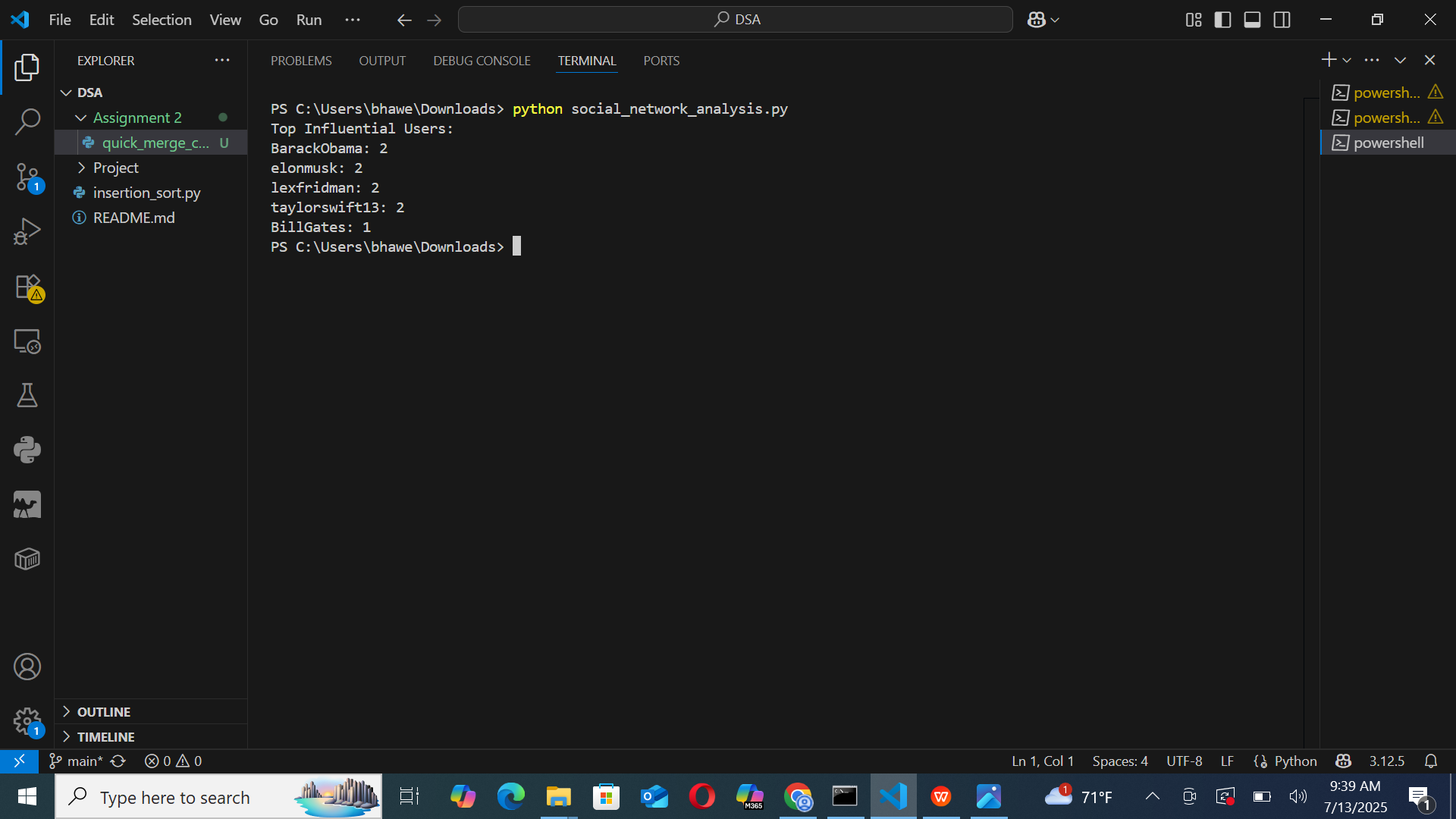
**5. Challenges and Limitations**

- Real-time updates: In real platforms, relationships change frequently. Our static implementation does not reflect these dynamics.  
- Scalability: Although this model is efficient, it is tested on a small sample. Large networks require optimized storage and distributed processing.  
- Influence Metrics: In-degree is a basic metric. Real systems consider engagement, reach, and sentiment analysis for more accurate ranking.

# **6. Demonstration and Testing**

The implementation was tested using a dataset of real-world influencer relationships. The graph structure was created with users as nodes and their 'follows' as directed edges. The `calculate\_influence` function was used to compute in-degree scores, and a max heap was used to extract the top 5 users with the highest influence.

Sample Output:

  
Edge Case Tested:  
- A user with no followers was added to validate the correctness of 0-score influence computation.

# **7. Implementation Challenges and Solutions**

- Challenge: Python’s built-in `heapq` only supports min-heaps.  
- Solution: Used negative values to simulate max-heap behavior for ranking.  
  
- Challenge: Keeping the implementation modular and readable.  
- Solution: Separated logic into three classes: `Graph`, `UserMetrics`, and `MaxHeap`.  
  
- Challenge: Designing meaningful test cases for influence.  
- Solution: Used a sample of well-known influencers with mutual and asymmetric connections.

# **8. Next Steps**

- Extend the analysis to include weights for likes, retweets, and comments.  
- Replace static dataset with a real-time data stream from a social media API.  
- Improve performance using optimized data structures or concurrency.  
- Visualize network graphs using libraries like NetworkX or Plotly.  
- Integrate with a front-end dashboard for interactive influence tracking.

# **9. Code Snippets and Documentation**

For complete source code and modular files, visit the GitHub repository:

[*https://github.com/Bhawesh-03/MSCS532\_Project.git*](https://github.com/Bhawesh-03/MSCS532_Project.git)

# **10. Conclusion**

This project demonstrates how fundamental data structures can be adapted to solve practical problems in social network analysis. By implementing graphs, heaps, and hash tables in Python, we can identify influential users in a scalable and efficient way. This lays the groundwork for more advanced analytics in real-world applications.

# **References**

* Aral, S., & Walker, D. (2012). Identifying influential and susceptible members of social networks. Science, 337(6092), 337–341. <https://doi.org/10.1126/science.1215842>
* Tang, J., Chang, S., Aggarwal, C., & Liu, H. (2016). Node classification in signed social networks. Proceedings of the 2016 SIAM International Conference on Data Mining, 54–62. <https://doi.org/10.1137/1.9781611974348.7>
* Kwak, H., Lee, C., Park, H., & Moon, S. (2010). What is Twitter, a social network or a news media? Proceedings of the 19th International Conference on World Wide Web, 591–600. <https://doi.org/10.1145/1772690.1772751>