**Social Network Data Structures: Proof of Concept Implementation**

Shyam Nath

University of Cumberland

Algorithms and Data Structures (MSCS-532-B01)

Student ID: 005032491

Project Phase 2 Deliverable 2

July 27, 2025

**Social Network Data Structures: Proof of Concept Implementation**

**Partial Implementation Overview**

The aim of this proof of concept (PoC) is to come up with part of the implementation of the data structures achieved in Phase 1 that will be applicable to a social network application. A Hash Table is mainly used as the primary data structure to handle user profiles, with a Graph (Adjacency List) as the second implementation used to model user relationships (followers/following). Also, a Priority Queue (Heap) is realized in order to find out the top influencers according to the number of their followers.

The Hash Table offers good insertion, deletion and look up which is important in scaling the user information. Complex social relations can be represented in the Graph structure, and each user node has a list of related user nodes (followers). The Priority Queue allows retrieving the top-K influencers by ranking them by the number of their followers (De et al., 2022).

In the PoC, we realized the necessary operations among which there were:

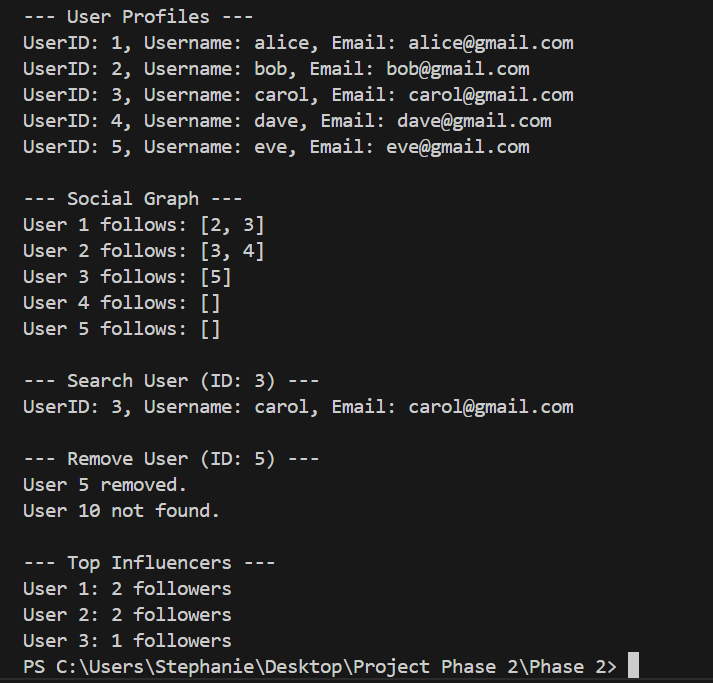
* Adding an user profile.
* Referring and deleting a user.
* Incorporation of links among the users.
* The showing of all users and their connections.
* Recovery of the top-K influencers.

The modular design will also allow the possibility of extending these components at a later stage to carry other functionalities such as posts, likes, and advanced search functions.

**Demonstration and Testing**

Python (main.py) Script was written to illustrate important functions handled by the data structures. Such functions were tested (Mattie, 2025):

1. Add Users into Hash Table
2. Show all users
3. Add Social Linkages in Social Graph
4. Find User
5. Remove a User
6. Retrieve Best-K Influencers

**Test Case Results:**

**Challenges in Implementation and their Solutions**

**Hash Table Collision Handling:**

Among the issues that were faced was the development of a good scheme to resolve the Hash Table collisions. Because collision handing is intrinsic to Python built-in dictionary, abstractions over low-level collision resolution such as chaining or open addressing were abstracted. Nevertheless, the uniqueness of the user IDs before insertion was made mandatory by performing validation checks.

**Solution:** Before inserting a new user, there was a simple existence check provided. The situation is similar when trying to add a user with an already existing ID: you get a warning display (Wardhani et al., 2024).

**Connection Management of Graph Structure:**

Graph was used with an adjacency list structure in which a user has a list of the users that he follows. There was a problem of having two-way awareness of connections in case of extending to the future (e.g., mutual friendships).

**Solution:** In this PoC unidirectional edges (followers) were used without making the structure more complex. To accommodate bidirectional relationships, the Graph can be enhanced to accommodate reverse look up to complement future phases.

**Effective Searching of Best Influencers:**

The use of the top-K top influencers meant needing to follow the whole graph and tally the number of followers to order the list, which would be slow when used at a large scale.

**Solution:** It employed Priority Queue (Heap) to have an efficient retrieval mechanism. heapq.nlargest() made sure that the K users having the most followers would be able to be gotten in O(N log K) time (Rai et al., 2024).

**Modularity and Scalability of Codes:**

The primary design challenge was to design the classes so they can be extended in the future in case it is decided to add posts, comments, and likes.

**Solution:** The classes were divided into independent modules: UserProfileTable, SocialGraph and PriorityQueue so that the minimum alterations are needed when the new feature has to be added.

**Next Steps**

While the current proof of concept demonstrates the foundational data structures of a social network application, several enhancements are planned for future phases to progress toward a fully functional system. The following steps outline the key areas of development:  
1. Extend User Profiles  
2. Implement Friendships (Bidirectional Connections)  
3. Content Management (Posts, Comments, Likes)  
4. Advanced Search Functionalities  
5. Persistent Data Storage  
6. Optimization and Performance Testing  
7. User Interface Development

**Code Snippets and Explanations**

**Hash Table Implementation (User Profiles):**class UserProfileTable:  
 def \_\_init\_\_(self):  
 self.table = {} # Hash Map: user\_id -> User object  
  
 def add\_user(self, user\_id, username, email):  
 if user\_id in self.table:  
 print(f"User {user\_id} already exists.")  
 return  
 self.table[user\_id] = User(user\_id, username, email)  
 def get\_user(self, user\_id):  
 return self.table.get(user\_id, None)  
**Graph Implementation (User Connections):**  
class SocialGraph:  
 def \_\_init\_\_(self):  
 self.graph = {} # user\_id -> list of followers  
 def add\_user(self, user\_id):  
 if user\_id not in self.graph:  
 self.graph[user\_id] = []  
 def add\_connection(self, from\_user, to\_user):  
 if from\_user not in self.graph:  
 self.graph[from\_user] = []  
 self.graph[from\_user].append(to\_user)  
**Priority Queue for Top-K Influencers:**import heapq  
def top\_k\_influencers(graph\_obj, k=5):  
 influence\_scores = graph\_obj.get\_influence\_scores()  
 return heapq.nlargest(k, influence\_scores.items(), key=lambda x: x[1])

**Link to the GitHub Repository**

<https://github.com/Snath32491/MSCS_Project_Phase-2.git>

**References**

De, S., Dey, S., Bhatia, S., & Bhattacharyya, S. (2022). An introduction to data mining in social networks. In *Advanced data mining tools and methods for social computing* (pp. 1-25). Academic Press.

Mattie, M. (2025). Comprehensive Implementation Process.

Rai, M., Parmar, H., & Sharma, S. (2025, February). Comparative Analysis of Sorting Algorithms: Time Complexity and Practical Applications. In *2025 3rd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)* (pp. 26-32). IEEE.

Wardhani, D., Chyan, P., Mallu, S., Suseno, A. T., Fauzi, A., Sanjaya, I. G. W., ... & Al Mudzakir, T. (2024). Pengantar Data Mining. *Penerbit Mifandi Mandiri Digital*, *1*(01).