**Name: AJAL RC**

**Student ID: 005039456**

**Algorithms and Data Structures (MSCS-532-B01)**

**Deliverable 2: Proof of Concept Implementation**

**Overview**

For this phase, I moved from theory to practice by creating a Python implementation of my social network analysis design. The SocialNetwork class combines several key data structures including an adjacency list (similar to Python dictionaries and sets) to represent user connections, a hash table for user profile information, and a priority queue (example with heapq) for ranking users by their connections. The code also allows for quick lookups, efficient network traversals, and scalable analysis, as recommended in recent graph processing literature (Cheng, Yu, & Qin, 2013; Sahu et al., 2017).

**Demonstration of Key Operations**

For the proof of concept, I wrote a script that tests each core operation of the social network class. Here are some of the basic operations in the algorithm:

**Basic Operations:**

1. **Adding Users:**

Each time new users are added, they are assigned an empty set of connections and a blank profile, coinciding with how a new profile is created in a regular social media platform. This operation ensures the network can grow as new users join (Sahu et al., 2017).

1. **Creating Connections:**

Connections are directional (like follows on Twitter) or graphical (like we see in Facebook or Instagram). Adding a connection updates the adjacency list to link user A with user B. The code checks for valid users and avoids duplicate connections, which makes sense as friendship cannot happen with an non-existing person, and the unique profiling is equally important.

1. **Adding and Retrieving Profiles:**

Each user can have a profile with details like name and bio. The system uses a dictionary (hash table) for constant-time updates and lookups, as suggested in research on high-performance key-value stores (Hao et al., 2018).

1. **Finding Influencers:**

To identify the most "influential" users, the program calculates degree centrality (number of outgoing connections). It then uses a max heap (priority queue) to find the top k users quickly.

1. **Breadth-First Search (BFS):**

BFS is implemented to explore the network and efficiently find the connections. BFS finds all users reachable from a starting point. BFS is one of the most effective operations, especially for finding connected communities or detecting the spread of information (Cheng, Yu, & Qin, 2013).

Here is the demonstration code for the implementation where the SocialNetwork class is called and core operations are performed.

A screen shot of a computer program

AI-generated content may be incorrect.  
  
Some of the operations as seen in the screenshot above are:

* 1. The user network is growing as new connections are added.
  2. Profile data is stored and retrieved as needed.
  3. The top influencers can be identified with simple function calls, making it a valuable tool for social network analysis.
  4. BFS allows connections to be reached effectively, allowing faster trackability and modifications.

**This is also what the output looks like: we are doing some core operations like adding new users and creating new connections between Alice, Bob, and Carol. I am also making sure the profile is set up correctly. We can also see that I can successfully set up the degree centrality and ensure that I can see all the neighboring connections for everyone.**

A computer screen shot of white text

AI-generated content may be incorrect.

**Implementation Challenges and Solutions**

1. **Avoiding Duplicates:**

One of the significant challenges was to ensure that connections are not duplicates. This also includes making sure that you cannot create a connection with yourself. To resolve this duplication issue, sets were implemented. Sets ensured each user and connection was unique—no wasted space or repeated work.

1. **Code Modularity:**

Social Networking requires many core operations, from adding a new profile to maintaining the degree of centrality. Most of these operations can be similar yet independent, so separating them is a good practice for easy maintainability and avoiding potential issues. Each feature was put in its own method, so it is easy to expand, debug, or swap out algorithms if better ones are found (Sahu et al., 2017).

1. **Error Handling:**

Another important challenge is handling errors properly. When there are issues with adding or modifying existing networks, we must error gracefully instead of crashing the whole site. The code gracefully handles missing users or invalid connections by printing helpful messages rather than failing silently.

**Next Steps**

The current proof of concept is functional for small networks. A couple of minor improvements could be made for the next phase.

1. Add more advanced analytics (like betweenness/closeness centrality) to measure influence and information flow better (Cheng, Yu, & Qin, 2013).
2. Allow users and connections to be deleted or modified. This makes sense because people greatly update their profile, including their profile picture or surname, after marriage, education status, and many more.
3. Begin scaling up by considering more efficient data structures or database storage, as real-world networks often have millions of nodes (Hao et al., 2018). The current implementation of the Hash table is only for a proof of concept but may not be the optimum solution.
4. Consider privacy, security, and the user interface for a more complete application, which will not be a small change per se, but will match a lot with the real implementation.

**Code Snippets and Documentations**

The full well documented implementation, including the read me can be found in my GitHub account here in the link: <https://github.com/ucajalrc/Deliverable-2>

**References**

Cheng, J., Yu, J. X., & Qin, L. (2013). Graph computation: Techniques and applications. Proceedings of the VLDB Endowment, 6(12), 1458–1459.

Hao, J., Li, J., Li, H., & Wu, K. (2018). Optimizing large-scale hash tables for key-value stores. IEEE Transactions on Computers, 67(4), 543–555.

Sahu, A. K., et al. (2017). The who, what, and how of high-performance graph processing. Proceedings of the 2017 Symposium on Cloud Computing, 475-489.