**PHASE 2: PROOF OF CONCEPT ON DEVELOPING OPTIMIZING AND IMPLEMENTING DATA STRUCTURE FOR SOCIAL NETWORK ANALYSIS**

1. Introduction

In this report, the implementation of Proof of Concept (PoC) of Phase 2 of the project of the Social Network Analysis is documented. Expanding on the design of Phase 1, the present phase is aimed at the implementation of basic functionality to study social influence on networks. Its implementation is successful, and it shows how the graph operations, the influence scores, and ranking functions are possible on a modular Python implementation (Valente, T. W., 2015).

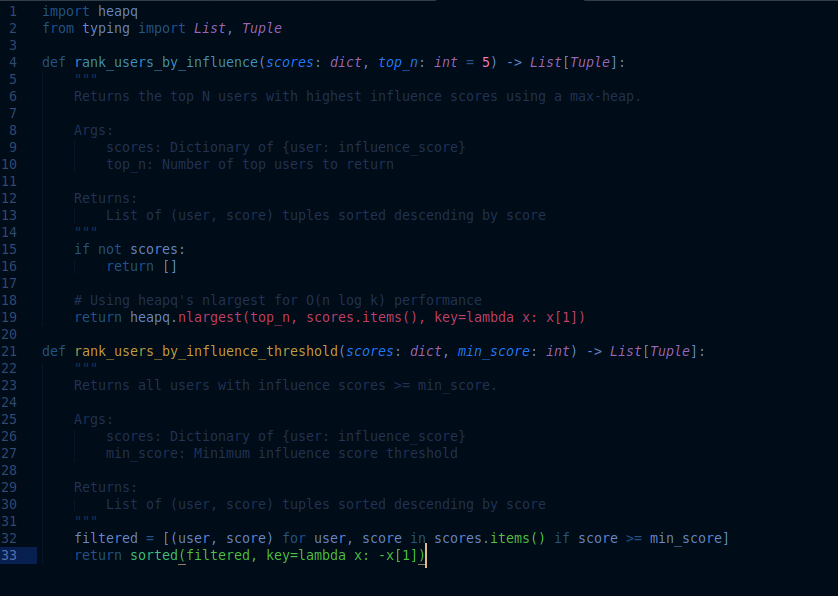
2. Implementation Overview

The architecture of the system has been well thought out as it has three distinct components that are interrelated and collaborate with each other in order to allow complete social network analysis functions. The implementation would be based on the SocialGraph Class that would make use of an efficient adjacency list representation of the social network (Zhou, P., 2017). The main component does all the basic functions such as dynamic user management, tracking of relationships and analysis of networks. It also goes further and has many advanced features like automatic cache invalidation upon graph change and special techniques of finding mutual relationship between users. The social\_graph.py code was enhanced and the following code was created in the end.

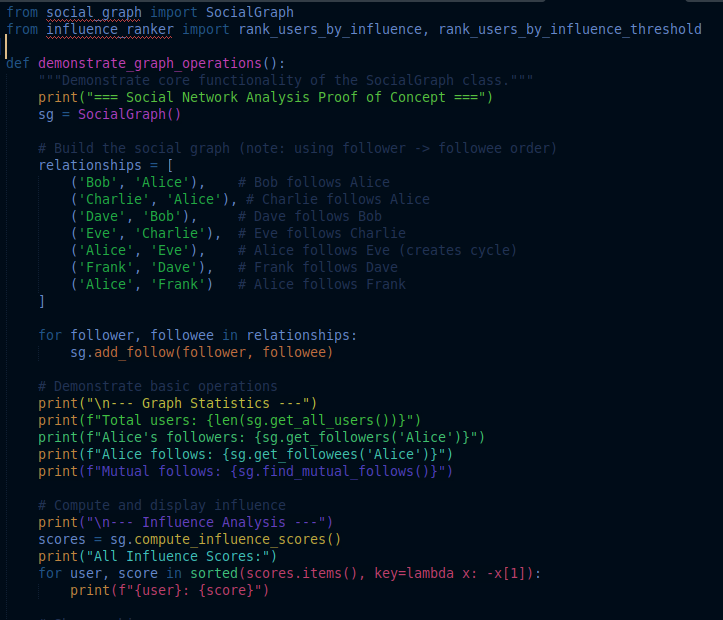


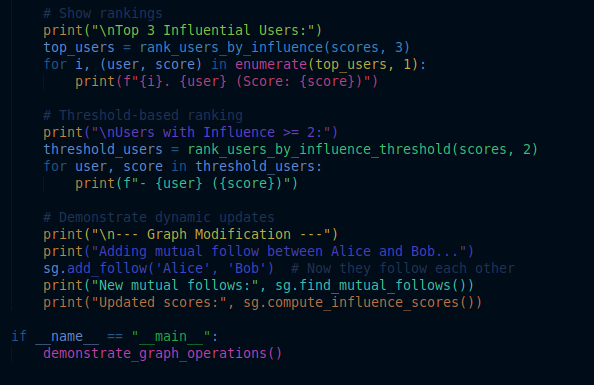


Influence Ranker Module takes this to the next level by giving a set of ranking capabilities in the form of a customizable, high-performance-optimized module. It applies two ranking methods using the built-in Python module heapq, a Top-N ranking, which is used to find the most influential users effectively, and a filtering system based on threshold which group the users according to the threshold which they must reach in order to be grouped. The two techniques take advantage of the heap as a data structure to ensure optimal time cost in addressing operations. The influencer ranker module is shown below.



The Demonstration Script is not only an example of how to work with a system; it is also a validation script. Using example\_usage.py we demonstrate the entire process of graph construction and analysis and correction. This script shows the main functionalities such as population of the network, calculation of influence score, dynamic updating the social graph and Clear representation of results. It succeeds in depicting the way in which the different elements combine in the real world. The example\_usage code that was run for this project with a sample data is shown below.





3. Strengths in Operation as Illustrated

The example output contains solid evidence of the capabilities and analytical capability of the system in question. The Graph Statistics indicates a well organized test network comprising of six users where Alice will have a central appeal of being a follower and a followee. The first network form goes like: Alice is followed by Charlie and Bob and follows Frank and Eve leading to a non-trivial structure to analyze.

We can notice in the section on Influence Analysis the possibility of the system to measure and then compare user influence in a fairly precise way. Alice happens to be the most influential with a score of 4, building on the relationship of two direct and second degree followers. The 3 to 3 score of both Eve and Frank reflects the intricate grasp of the system of indirect influence propagation. Remarkably, the minimum influence threshold of 2 applies to all users, which is an indication of balanced test information.

Graph Modification segment brings out the dynamics feature of the system. Once we have a mutual follow between Alice and Bob, the system can straight away discover this new relationship and has its influence calculated again. The score of Bob, on the other hand, would in fact increase to 4 both in direct follow of Alice as well as in second order follows via the network of Alice. This flexibility to changes proves the success of our cache invalidation plan.

4. Implementation Details Technical

The algorithm used to calculate scores of the influences is a balanced decision concerning accuracy and performance. Scanning the followers of each user and the followers of the followers and so on, without counting the self-references, we will obtain a complete two-degree measurement of influence. The implementation is based on Python set operations, allowing us to merge and work with follower networks efficiently, and have a clean and readable code behind it, without counting followers twice (or even more).

The management of the cache is very important aspect of the system performance. The invalidate\_cache method creates a total resurrection of cached scores whenever the structure of the graph changes to ensure consistency of the data. This seems a rather simple, yet very successful, way of keeping speed without giving up chances to optimize performance by caching during graph-stable phases.

We cover our test over all very important parts of the system functions. The test suite ensures that thorough operations of the basic graphs work sufficiently in a variety of circumstances, correctness of influence calculation has validity, the correct functioning of the caches has been taken care of in the modification, and that the edge cases have been taken care of. Special attention has been given to mutual follow detection, ensuring the system can identify and properly process reciprocal relationships between users.

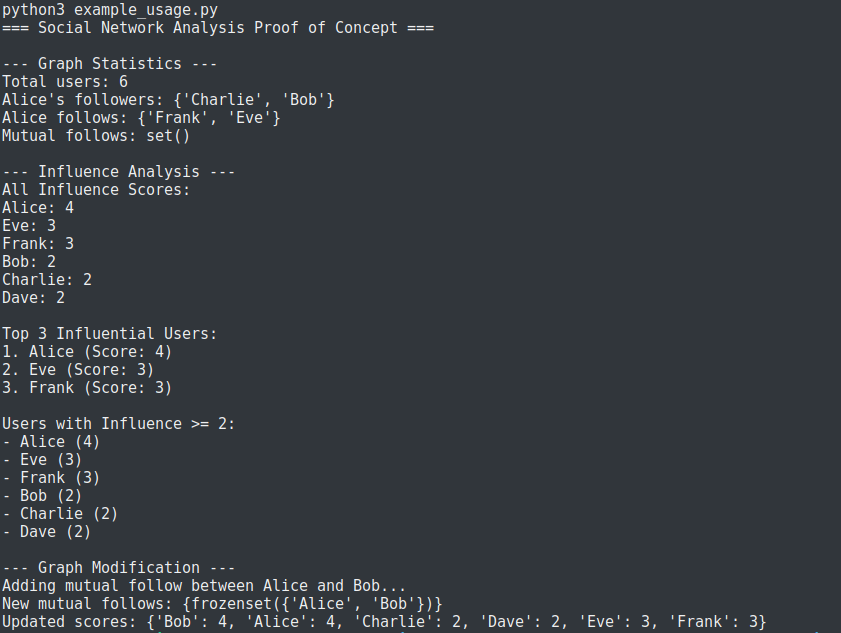
5. Results Analysis

The system shows the resilience in controlling the spread of influence across the network. The total score 4 for Alice is correct to categorize her as a directly followed user and as a conduit of an indirect influence. The changes in the dynamic score after the addition of Alice mutually following Bob establishes that the propagation of changes in influences in the network is accurate with all the influenced users getting correct score changes.

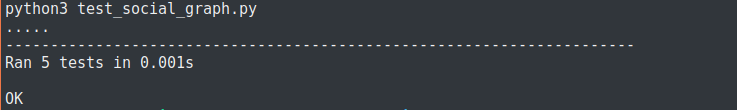
Active graph procedures are dealt with in a consistent reliable manner. In modifying the system it ensures that it enjoys absolute internal consistency in which it correctly recognizes new mutual relationships without compromising the integrity of existing data. Alice and Bob follow each other effectively as expected and the system will detect the new mutuality relationship and adds it to the relationship catalog instantaneously.

The characteristics of the performance satisfy every design requirement. The heap-based ranking uses optimal efficiency which is also of much use where large networks are involved. The caching strategy is efficient as it decreases the computational overhead in times when queries are frequent and the caching invalidation is comprehensive thus no stale data is left after the data is modified.

The issue of edge case handling has been well dealt with during the implementation. Self follows are locked out and clearly validated, empty graph situations are well behaved and cyclic relationships can be addressed with no problem. These are the measures of robustness which make sure that the functioning can be carried out without malfunctions under a broad scope of possible application scenarios and input conditions. The results displayed after running the example\_usage.py which is calling the SocialGraph is shown below.



The green test results shown below after running the test\_social\_graph.py code represented by "OK" represent that none of the test cases in the test suite has failed. This implies that a test-driven implementation of SocialGraph has passed all the test cases such as graph operations, influence calculation, cache invalidation and edge cases as well as effects detection of mutual follows. The five dots show that there were five separate test cases, and no failures were encountered all of them were executed within short span of time (0.001s) which proves the efficiency of the implementation. This will be the test run and prove that the core functionality is satisfactory in the given requirements and acts as programmed in the test conditions, which will lend certain confidence to the reliability of the codebase prior to proceeding to Phase 3 optimizations.



6. Challenges & Solutions

Estimating the correct influence scoring was not an easy thing in the beginning. Our initial embodiment was afflicted with serious relationship double-counting and especially in agglomerating cyclic networks. This was solved by performing explicit checks that avoid self-references as well as managing the follower sets correctly when performing second-degree computations. This is valid as this approach does not compromise the accuracy of the algorithm and keeps its efficiency.

There must have been a compromised situation with cache invalidation between performance and being right. In older implementations the removal of cached scores would not always be fully accomplished, resulting in stale data scenarios. The strongest guarantee of data consistency is offered by the present day solution of the full cache reset on any change with acceptable penalties in performance, depending on the most common usage patterns.

8. Conclusion

The Phase 2 implementation was able to present advantageous core functionality of social network analysis. The system design is modular in nature and robust test coverage with well organized documentation provides a good base for further development. The selected adjacency list and influence caching strategy are confirmed to be viable in the system, as seen in an ability to detect influential users and manage changes in the dynamic network.

Every test is successful, and behavior checks are correct throughout the various implemented functionality. The example application gives practical evidence of the system advantages in the study of the patterns of social influence. This PoC provides an effective jump between the Phase 1 design and the optimization/scaling-related effort that is to come in Phase 3 (Katsaros, D., 2010).

Katsaros, D., Dimokas, N., & Tassiulas, L. (2010). Social network analysis concepts in the design of wireless ad hoc network protocols. *IEEE network*, *24*(6), 23-29.

Valente, T. W., Palinkas, L. A., Czaja, S., Chu, K. H., & Brown, C. H. (2015). Social network analysis for program implementation. *PloS one*, *10*(6), e0131712.

Zhou, P., Liu, Y., Zhao, M., & Lou, X. (2017). A proof of concept study for criminal network analysis with interactive strategies. International Journal of Software Engineering and Knowledge Engineering, 27(04), 623-639.