COMP47500 – Advanced Data Structures in Java

Assignment No: 5

Date: 06/05/2024

Title: Leveraging bidirectional Graph to calculate for top social media influencers using Scoring system

Code Link : https://github.com/COMP47500-Adv-Data-Structures-in-Java/Assignment_5

Execute: main.java inside ie/ucd/csnl/comp47500

Assignment Type of Submission:			
Group	**Yes/No**	**List all of group members' details:**	**% Contribution** **Assignment Workload**
	Yes	Student Name: Prakash Jha Student ID: 2320077	25%
		Student Name: Satish Gaikwad Student ID: 23204142	25%
		Student Name: Chungman Lee Student ID: 23205535	25%
		Student Name: Abhijith Sathyendran Student ID: 23200124	25%

Table of Contents

1. Problem Domain Description:

The problem domain here is social network analysis, specifically the calculation of influence scores for social media accounts. The system serves as an analytical tool to calculate the reach of influencers in social networks.

The code is designed to model a social network as a graph, where vertices represent user profiles and edges represent interactions between users. The interactions can be of different types, such as "FOLLOW", "COMMENT", "LIKE", or "SHARE", and can be bidirectional or unidirectional.

The main tasks performed by the code are:

1. Creating User Profiles: User profiles are created with a name and age. These profiles are then used to create vertices in the graph.
2. Creating a Social Network Graph: A new social network graph is created as an instance of the SocialNetworkGraph class.
3. Adding Vertices and Edges: Vertices, representing user profiles, are added to the graph. Interactions between users, represented as edges, are also added. The type of interaction and whether it's bidirectional are specified when adding an edge.
4. Checking for Vertex and Edge Existence: The code checks whether a specific vertex or edge exists in the graph.
5. Calculating Influence Scores: The influence scores of the accounts are calculated, The influence score depends on the number and type of interactions a user has with other users.

The goal of this code is to analyze the influence of different social media accounts based on their interactions with others. This could be used for various purposes, such as identifying key influencers in a network or studying the spread of information or trends.

Key Features:

- **Social Network Graph Representation:**

  The codebase provides a comprehensive implementation of a social network graph, allowing the modeling of relationships and interactions between users.

- **Modular Design:**

  The codebase is modular, with separate implementations for vertices, edges, and the graph itself. This modular design promotes code reusability and maintainability.

- **Vertex and Edge Abstraction:**

  The Vertex and Edge interfaces abstract the concepts of vertices and edges, providing a flexible framework for representing different types of data and relationships within the graph.

- **Interaction Types Enumeration:**

  Interaction types are represented using an enum (InteractionType), allowing for easy categorization and management of user interactions such as likes, comments, shares, and follows.

- **Influence Score Calculation:**

  The codebase includes functionality to calculate influence scores for users based on their interactions within the social network. This feature provides insights into user influence dynamics.

- **Top Influencer Identification:**

  The project includes functionality to identify the top influencers in the social network based on their influence scores. This feature facilitates the identification of key users driving engagement and interaction within the network.

- **Graph Manipulation Operations:**

  Essential graph manipulation operations such as adding vertices and edges, retrieving edge lists, checking vertex and edge existence, and counting vertices and edges are implemented, providing core functionality for graph management.

- **Graph Visualization:**

  The printGraph() function enables the visualization of the graph's adjacency list, allowing users to inspect the structure and connections of the social network graph.

- **Scalability and Extensibility:**

  The codebase is designed to be scalable and extensible, allowing for the addition of new features and enhancements to accommodate evolving requirements or larger datasets.

- **Clear Documentation and Code Organization:**

  The codebase includes clear documentation and well-organized code structure, making it easy to understand, maintain, and extend. Each function is appropriately named and documented, enhancing code readability and usability.

Use case diagram:

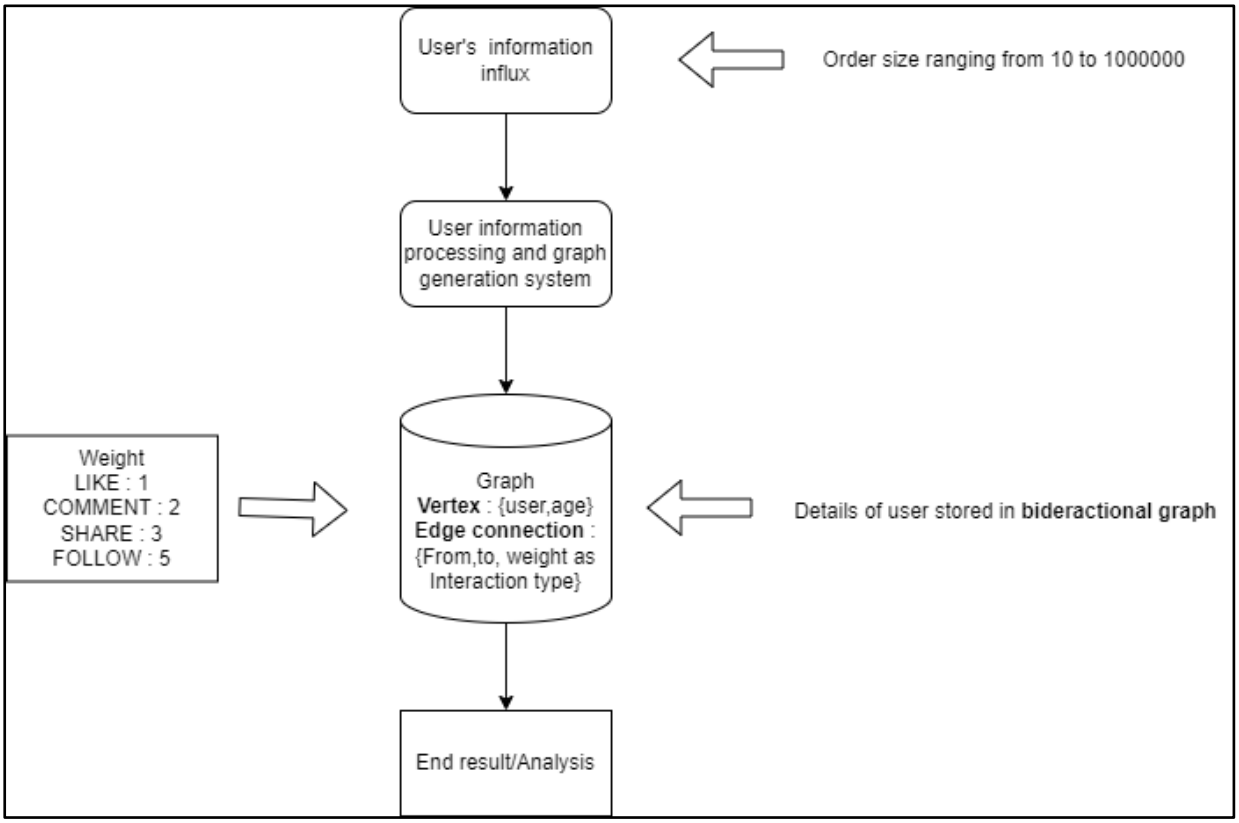


Figure 1: Use case Diagram

The above diagram explains how the algorithm works, it accepts the user's data like full name and age, and next the vertex is created of name and age.
The edge creation depends on 4 parameters:

- **From and to**: this is the user information(user name & age) to connect to the user vertex and form an edge
- **Weight**: the weight of an edge is dependent on whether the (from user) like, follows, comment or share the (to user's) post
- **Bidirectional**: in above point the interaction is only on from user -> to the user, to do the same, simple put bidirectional as true:

Lastly after creating the graph, the influencer's(user) scores are calculated and top 3 are displayed.

2. Theoretical Foundations of the Data Structure(s) utilised

The theoretical foundations of the data structures utilized in the provided code include:

1. Graph:

Definition: A graph is a collection of vertices (or nodes) and edges that connect pairs of vertices.
Theoretical Foundation: Graph theory, a branch of mathematics, deals with the study of graphs and their properties. It includes various concepts such as paths, cycles, connectivity, and algorithms for traversing and manipulating graphs.
Usage in Code: The Graph interface and its implementations (GraphImpl and SocialNetworkGraph) represent the abstraction of a graph data structure. They define methods for adding vertices and edges, retrieving edges from vertices, checking for the existence of vertices and edges, and calculating properties of the graph [1].

2. Vertex:

Definition: A vertex (or node) is a fundamental unit of a graph, representing an entity or object.
Theoretical Foundation: In graph theory, vertices represent individual entities or objects, and edges represent relationships or connections between them.
Usage in Code: The Vertex interface and its implementations (VertexImpl and UserVertex) represent vertices in the graph data structure. They encapsulate data associated with individual entities (such as user profiles) and provide methods for accessing and representing vertex data.

3. Edge:

Definition: An edge is a connection between two vertices in a graph, representing a relationship or interaction between them.
Theoretical Foundation: Edges define relationships between vertices and can have properties such as weight, direction, and type. Graph algorithms often operate on edges to analyze connectivity, paths, and other properties of the graph.
Usage in Code: The Edge interface and its implementations (EdgeImpl and InteractionEdge) represent edges in the graph data structure. They define methods for accessing edge properties such as weight and vertices connected by the edge.

4. InteractionType:

Definition: Interaction types represent the different ways users can interact with each other's content in a social network.
Theoretical Foundation: Interaction types categorize various actions users can perform within a social network, such as liking, commenting, sharing, and following. These interactions influence the structure and dynamics of the social network graph.
Usage in Code: The InteractionType enum defines constants for different interaction types, providing a structured way to categorize user interactions within the social network graph.

Operations implemented in the project:

- addVertex(Vertex<T> vertex): Adds a vertex to the graph.
- addEdge(Vertex<T> from, Vertex<T> to, int weight, boolean bidirectional): Adds an edge between two vertices in the graph.
- getEdgesFrom(Vertex<T> vertex): Retrieves a list of edges originating from a specified vertex.
- hasVertex(Vertex<T> vertex): Checks if a vertex exists in the graph.
- hasEdge(Vertex<T> from, Vertex<T> to): Checks if an edge exists between two vertices in the graph.
- getVertexCount(): Returns the number of vertices in the graph.
- getEdgeCount(boolean bidirectional): Returns the number of edges in the graph.
- printGraph(): Prints the adjacency list of the graph to the console.
- calculateInfluenceScores(): Calculates the influence scores for all users in the graph. Initialises the score for every vertex and reviews each edge originating from the vertex and allocates scores to the recipient based on the type of interaction.
- findKeyInfluencers(int topN): Identifies the top N influencers in the graph based on their influence scores. Iterates through all the vertices in the graph, adding user profiles to a list. Then sorts the list in descending order by influence score and selects the top N profiles.

Time complexity table for implemented operations:

V represents the number of vertices

E represents the number of edges connecting those vertices

addVertex(Vertex<T> vertex)	O(1)
addEdge(Vertex<T> from, Vertex<T> to, int weight, boolean bidirectional)	O(1)
getEdgesFrom(Vertex<T> vertex)	O(1)
hasVertex(Vertex<T> vertex)	O(1)
hasEdge(Vertex<T> from, Vertex<T> to)	O(n)

Time complexity table for HashMapImpl class's operations (implement AbstractMap):

getVertexCount()	O(1)
calculateInfluenceScores()	O(V+E)
findKeyInfluencers()	O(V+E)
addInteractionEdge()	O(1)
printGraph()	O(V+E)
getAdjacencyList()	O(V)
getEdgeCount()	O(V+E)
hasEdge()	O(E)
getEdgesFrom()	O(1)

3. Analysis/Design (UML Diagram(s))

DESIGN CONSIDERATIONS: Sequence Diagram

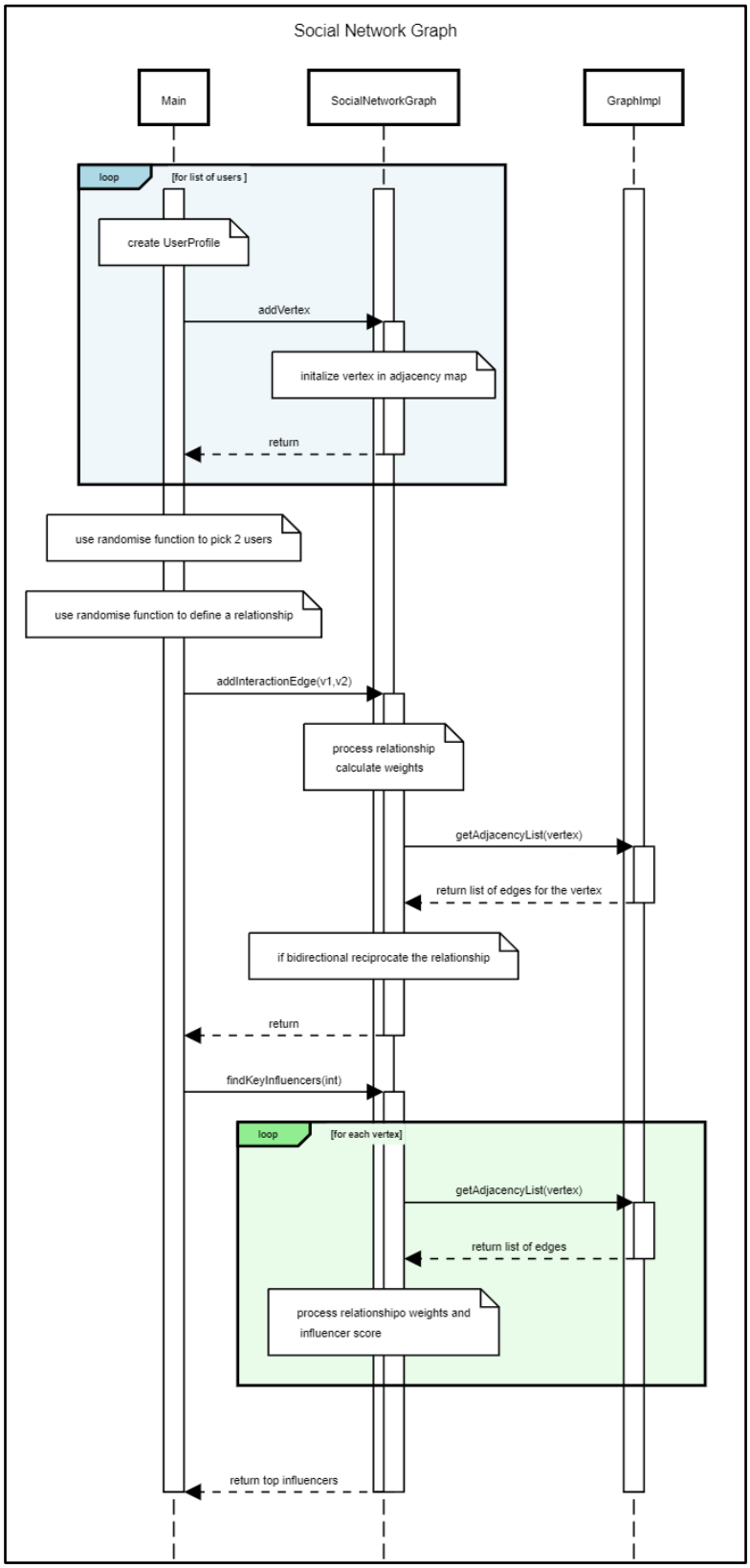


Figure 2: Sequence Diagram

Class Diagram: [PlantUML link to view class diagram]

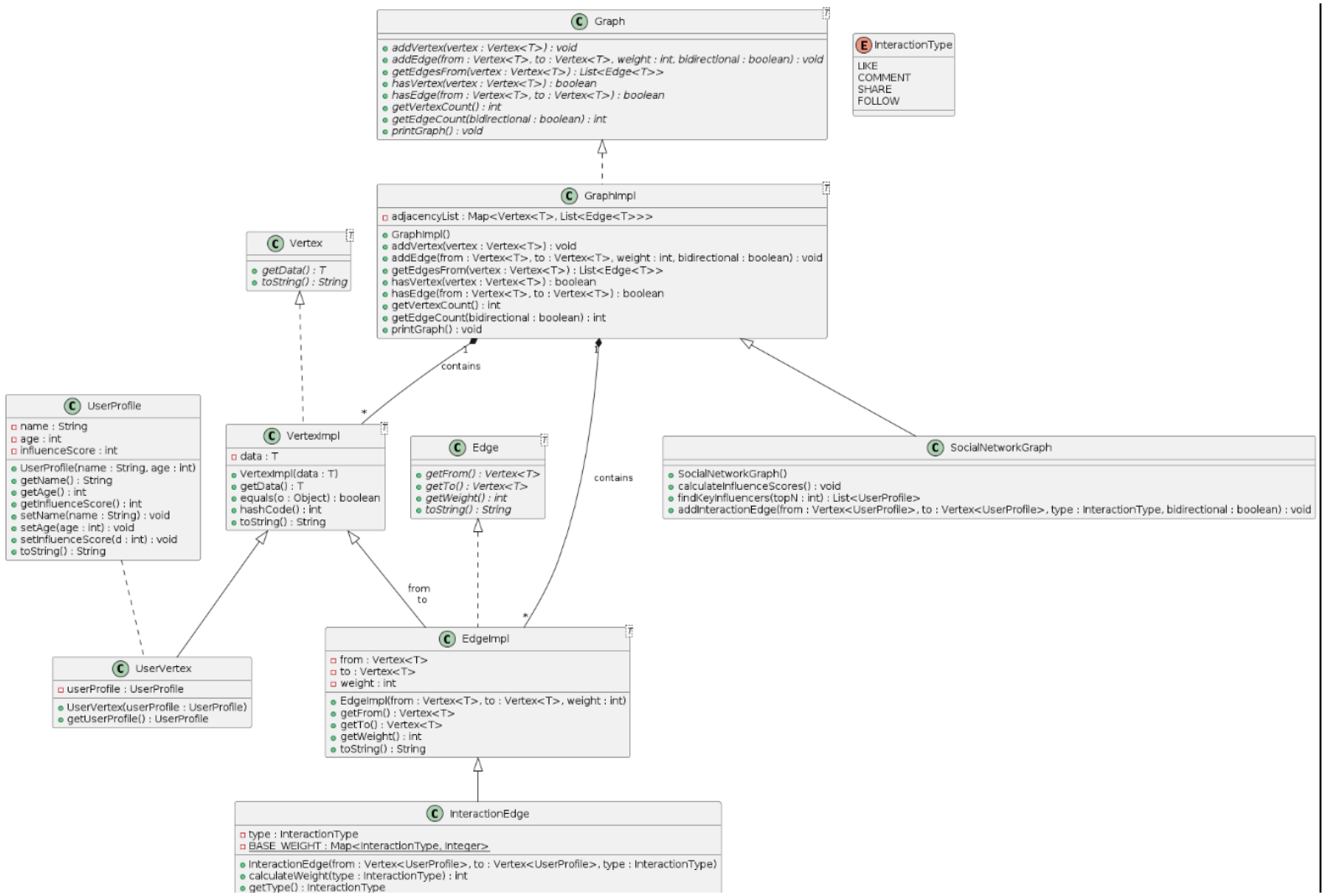


Figure 3: Class Diagram

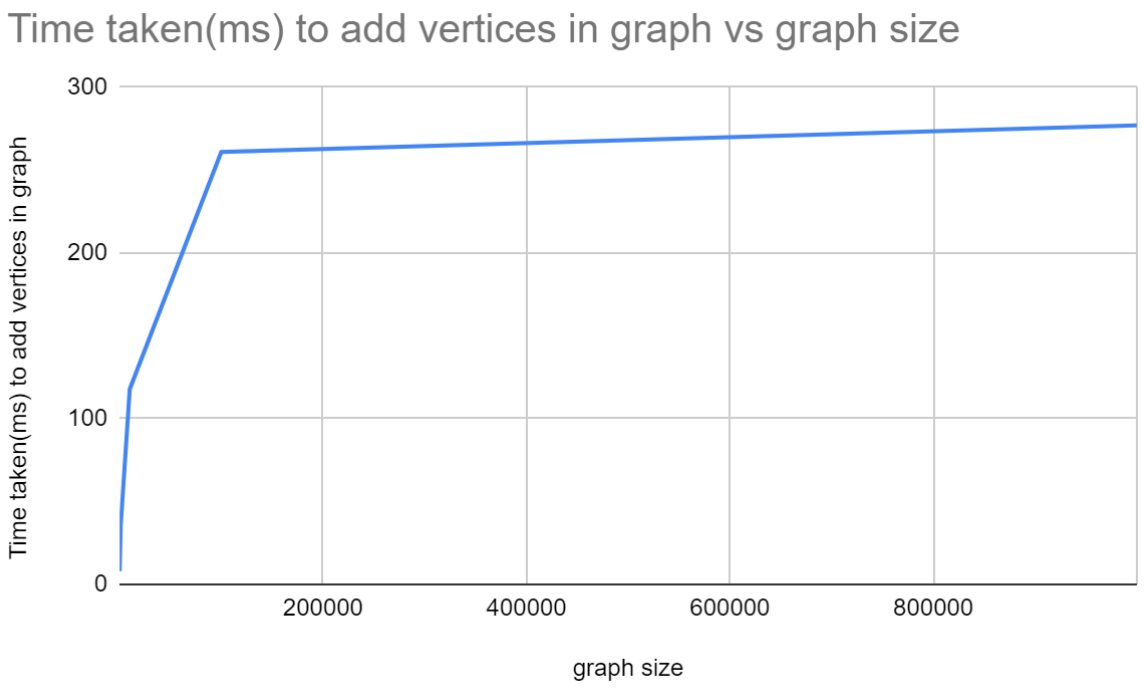
4. Code Implementation:

GitHub (link): https://github.com/COMP47500-Adv-Data-Structures-in-Java/Assignment_5

5. Set of Experiments run and results:

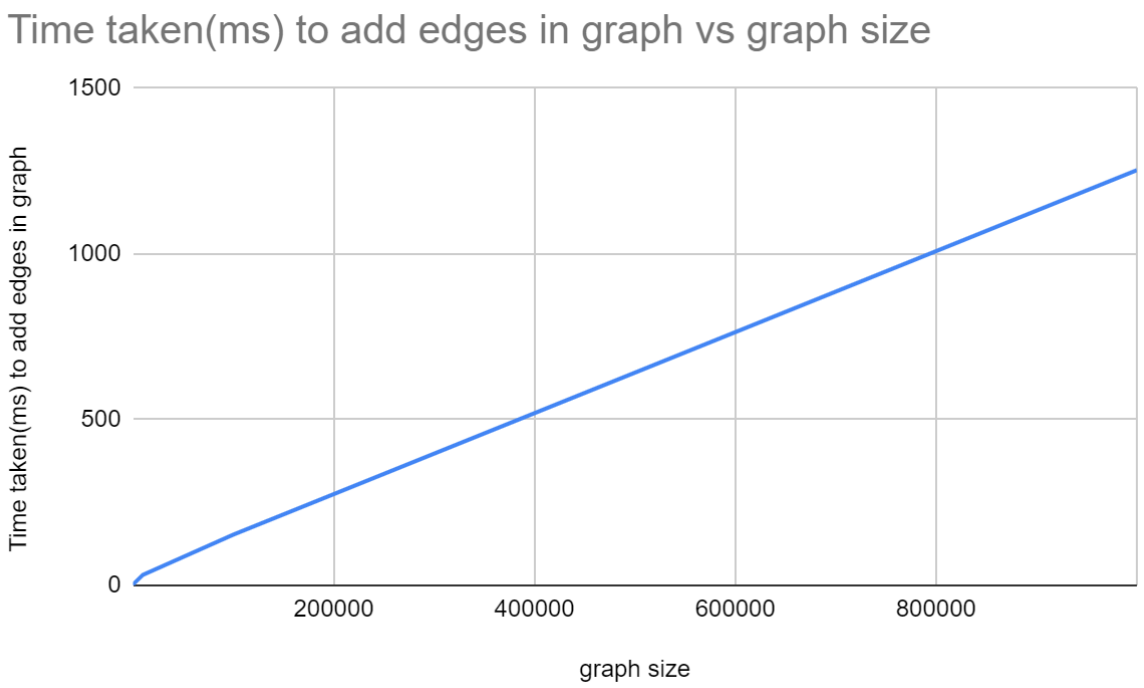
graph size	Time taken to add vertices in graph	Time taken to add edges in graph	Time taken to calculate influence scores using traversal algorithm	Time taken to find top 3 influencer
10	**8**	**3**	**1**	**39**
100	**10**	**1**	**1**	**2**
1000	**36**	**6**	**3**	**5**
10000	**118**	**32**	**23**	**17**
100000	**261**	**153**	**91**	**84**
1000000	**277**	**1253**	**294**	**78**

Graph 1:



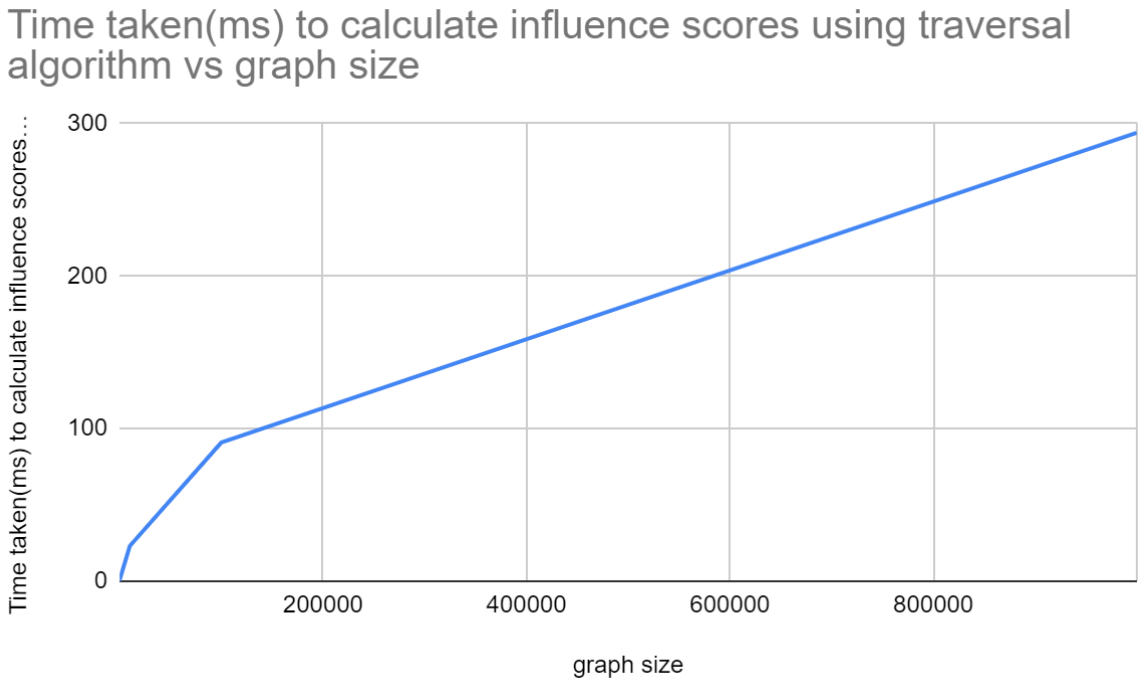
The above graph indicates the time taken to add vertices in a graph, as we can see with the increase in size the time also increases up until one point then it remains linear as it depends mostly on invoking the new vertices.

Graph 2:



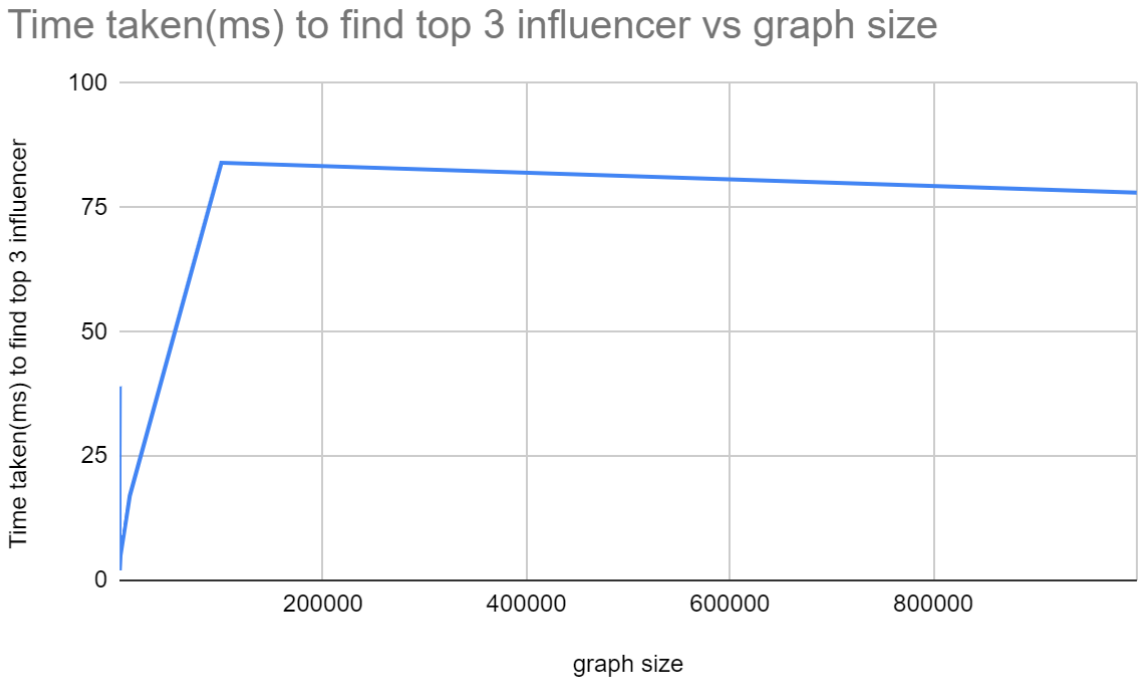
The above graph indicates the time taken to add edges in the graph vs graph size, the single operations are O(1), but as the operations increase so does time complexity linearly [2].

Graph 3:



The above graph indicates the time taken to calculate the scores of each user and it is a combination of multiple traversal algorithms: which Calculates the influence scores for all users in the graph. Initialises the score for every vertex reviews each edge originating from the vertex and allocates scores to the recipient based on the type of interaction. This leads to a somewhat linear time complexity.

Graph 4:



The above graph gives the analysis of size vs time taken to find the top 3 influencers, this graph may depend from system to system as it first Identifies the top N influencers in the graph based on their influence scores. Iterates through all the vertices in the graph, adding user profiles to a list. Then sorts the list in descending order by influence score and selects the top N profiles.

Execution output:

```
_____________________________________________________________________
processing graph with vertices: 10
Time taken to add vertices in graph 7 ms
Time taken to add edges in graph 2 ms
Time taken to calculate influence scores using traversal
algorithm 0 ms

Top Influencer:
Deborah Wehner IV with score 21
Diane Marks with score 9
Lyle Roob I with score 8

Time taken to find top influencer 35 ms
_____________________________________________________________________
processing graph with vertices: 100
Time taken to add vertices in graph 3 ms
Time taken to add edges in graph 1 ms
Time taken to calculate influence scores using traversal
algorithm 2 ms

Top Influencer:
Deborah Wehner IV with score 21
Percy Walsh with score 18
Shawn Brakus with score 16

Time taken to find top influencer 1 ms
_____________________________________________________________________
processing graph with vertices: 1000
Time taken to add vertices in graph 17 ms
Time taken to add edges in graph 4 ms
Time taken to calculate influence scores using traversal
algorithm 4 ms

Top Influencer:
May Reichel with score 22
Deborah Wehner IV with score 21
Stella Haley with score 20

Time taken to find top influencer 4 ms
_____________________________________________________________________
processing graph with vertices: 10000
Time taken to add vertices in graph 66 ms
Time taken to add edges in graph 28 ms
Time taken to calculate influence scores using traversal
algorithm 22 ms

Top Influencer:
Marguerite Gusikowski with score 27
Chad White with score 24
```

```
Dr. Clifford Langworth with score 23

Time taken to find top influencer 13 ms

_____________________________________________________________
processing graph with vertices: 100000
Time taken to add vertices in graph 220 ms
Time taken to add edges in graph 137 ms
Time taken to calculate influence scores using traversal
algorithm 53 ms

Top Influencer:
Jenna Durgan Jr. with score 31
Mark Lebsack-Hartmann with score 29
Ted Friesen PhD with score 29

Time taken to find top influencer 66 ms

_____________________________________________________________
processing graph with vertices: 1000000
Time taken to add vertices in graph 223 ms
Time taken to add edges in graph 928 ms
Time taken to calculate influence scores using traversal
algorithm 126 ms

Top Influencer:
Elbert Reinger with score 108
Kristina Heller with score 103
Salvatore Terry with score 101

Time taken to find top influencer 87 ms
```

6. Video of the Implementation running

Recording link:
https://drive.google.com/file/d/1ACt1Otg-6l2XQNLo69IST9b1OpOHgNS5/view?usp=sharing

Comments: uploaded video recording in google drive.

7. References

[1] Graphs (Java SE 10 & JDK 10). (n.d.). https://techdevguide.withgoogle.com/paths/data-structures-and-algorithms/#sequence-6

[2] The MIT Press, Massachusetts Institute of Technology. (2024, January 12). Book details - MIT Press. MIT Press. https://mitpress.mit.edu/9780262533058/introduction-to-algorithms/