Social Network Analysis - Project Report

1. Introduction

This project simulates a social network where people can connect, display mutual friends, suggest friends, and find the shortest path between two people in the network. The network is represented using an adjacency list, and different functionalities are implemented using graph algorithms to explore the relationships between people in the social network.

2. Objectives

The objectives of this project are:

- To simulate a social network where users can interact and build relationships.
- To implement features like adding people, creating connections, finding mutual friends, suggesting friends, displaying profiles, and finding the shortest path between users.
- To explore graph-based algorithms to represent and manipulate the social network.

3. System Requirements

Software Requirements:

- C++ compiler (e.g., GCC or Clang)
- Any text editor or IDE (e.g., Visual Studio Code, Code::Blocks)
- Standard C++ libraries (e.g., vector, unordered_map, queue)

Hardware Requirements:

- A system with a minimum of 2 GB RAM and 1 GHz processor.
- Disk space for code and project files.

4. Graph Data Structure

The social network is represented using an undirected graph. Each person in the network is

represented as a node (vertex), and a connection between two people is represented as an edge between two nodes. The graph is implemented using an adjacency list, where each node has a list of its adjacent nodes (i.e., its connections). This structure allows efficient traversal and manipulation of the network.

5. Code Explanation

The code for this project is organized into a class named `SocialNetwork` that contains various methods to manage the network, such as adding people, adding connections, removing connections, and displaying mutual friends or profiles. The network is represented using an adjacency list (unordered_map of strings and vectors). Key features include:

- `addPerson`: Adds a person to the network.
- `addConnection`: Establishes a connection between two people.
- `removeConnection`: Removes a connection between two people.
- `displayMutualFriends`: Displays mutual friends between two people.
- `suggestFriends`: Suggests new friends for a person based on their connections.
- `displayPersonProfile`: Displays the profile of a person.
- `findShortestPath`: Finds the shortest path between two people in the network.

6. Testing and Results

The functionality of the system was tested using a set of people and connections. The results were validated by adding, connecting, and removing people from the network. Additionally, the shortest path and mutual friends functionalities were tested by providing different user inputs. Sample results include:

- Mutual friends between two people: List of mutual friends displayed.
- Shortest path between two people: Path is shown in the format person1 -> person2 -> ... -> personN.

- Friend suggestions: List of suggested friends based on indirect connections.

The system performed well for typical inputs and edge cases (e.g., no mutual friends, no path between people).

7. Conclusion

This project successfully simulates a social network, allowing users to interact with the network by adding people, creating connections, and exploring relationships. The graph data structure was effectively used to manage and manipulate the network. The program meets the defined objectives and provides a user-friendly interface for testing the features of the social network.

8. Future Improvements

Future improvements for this project include:

- Adding features such as messaging between users.
- Implementing more advanced algorithms for friend suggestions using recommendation systems.
- Optimizing the graph representation for large networks.
- Implementing a user interface (GUI) for a better user experience.
- Extending the system to handle more complex network dynamics like friend groups and blocked users.